Alma Mater Studiorum
Università degli Studi di Bologna

Advanced Research Center on Electronic Systems for
Information and Communication Technologies

XXVI Ph.D Course in Information Technologies

Application Platforms for the Internet of Things: Theory, Architecture, Protocols, Data Formats, and Privacy

Settore Concorsuale: ING-INF/03

Settore Scientifico disciplinare: 09/F2

Candidate:
Matteo Collina

Advisors:
Prof. Alessandro Vanelli-Coralli
Prof. Giovanni Emanuale Corazza

Coordinator:
Prof. Claudio Fiegna

March 2014

Contents

Introduction

The last 20 years of research in the pervasive computing area have seen very important steps towards the realization of Mark Weiser's vision of ubiquitous computing [1]: a world were technology vanishes in the background.

The advent of the World Wide Web revolutionized the way we work, communicate, socialize, and how business is done: the Internet has become one the most pervasive technology. Recently, smartphones and mobile broadband enabled us to carry the Internet in our pocket, seamlessy integrating it in our lives. However, everyday objects remain disconnected from the virtual world, while the Internet of Things (IoT) movement is exploring how to interconnect them. This technology shift is supposed to be greater than the advent of mobile phones, and in [2] a 2020 scenario where non-phone interconnected devices will be 10 times the phone devices (50 vs 5 billions) is foreseen. Other predictions value the IoT market to $309 billion by 2020 [3]. However, this work began before all this predictions were made.

The presented dissertation is the outcome of a three-year research program aimed to explore and solve the issues in the Internet of Things field. All the presented chapters adopt an application engineering point of view, presenting the new and old problems that we face in building

the Internet of Things.

Original Contributions

This dissertation explores the Internet of Things field: a junction between electronics, telecommuncation and software engineering. While many challenges still are unsolved in the electronics, this work focuses on how to build *the* Internet of Things from an architecture and network perspective. This dissertation analyzes and presents solutions for the major problems of such a global system: interoperability and privacy. In order to do so, we also analyze the communication between the things and the users, to fully understand and improve the state-of-art communication protocols.

Internet of Things application development happens in silos. As few best practices have been defined, engineers usually pick the best technologies for the problem under investigation, often using proprietary and closed communication protocols. Moreover, even if they decide to use a standard protocol, there are still many competing standards from different organizations. Thus, this field is extremely fragmented and some standards are more popular than others in some specific niches, and vice versa. In fact, there is the common belief that *"only if we can solve the interoperability problem we can have a real the Internet of Things"* [4]. In this dissertation, we discuss our solution to this compelling issue, the *Ponte* project, as presented in Chapter 2.

The major achievement presented in this work is the *Ponte* project [5], which is in the process of being released as an Eclipse Foundation project [6], and was presented at several developer-focused confer-

ences across the globe, such as Eclipse Day @ Googleplex (Mountain View), Distill by Engine Yard (San Francisco) and EclipseCon France (Toulouse). *Ponte* is the outcome of a research activity that identified a set of primitives for IoT applications. We argue that each IoT protocol can be expressed in term of those primitives, thus solving the interoperability problem at the application protocol level. Moreover, the primitives are network and transport independent and make no assumption in that regard. *Ponte* began with an early work of a cross-protocol bridge, *QEST* [7]. This work was partially funded by Mavigex Srl, and is an outcome of the ongoing collaboration between our research unit and the company.

As this dissertation aims to present a guideline for IoT application developers, it includes an analysis of the application protocols latency in different circumstances. Particularly, Chapter 3 discuss the latency and throughput of IoT protocols on a high delay/high error rate link. Moreover, Chapter 4 analyzes the latency of Real-Time web protocols over a GEO satellite link. This last work is part of the ESA SatNex III project and it is presented in [8].

Privacy issues follows the rise of interconnected devices count. Even if we care, privacy violations are exposed to the public usually after years of the fact. Thus, it is clear that *the* Internet of Things must ensure resilience to attacks, data authentication, access control and client privacy [9].

We argue that it is not possible to solve the privacy issue without solving the interoperability problem: enforcing privacy rules implies the need to limit and filter the data delivery process. In Chapter 5, after a brief digression on the possible data formats and semantics for IoT applications, this dissertation presents a novel approach for filtering data that

can increase the throughput by a factor of ten, as presented in [10]. This work was partially funded by the FP7 Project GAMBAS and it was lead by the Digital Enterprise Research Institute, Ireland.

Thesis Outline

This dissertation is organized as follows. Chapter 1 analyzes the possible architecture of an IoT application and discusses the role of each communication protocol in such architectures. Chapter 2 proposes a novel model for an IoT system and identifies a set of primitives that allows to solve the interoperability problem. Chapter 3 and Chapter 4 focus on the latency of binary and Web protocols for Soft Real-Time applications, such as the IoT. Chapter 5 proposes a solution for the privacy issue in the Internet of Things.

Chapter 1

Protocols and Platforms

The Internet of Things (IoT) is envisioned as the next industrial revolution. The ubiquity of sensors and actuators will allow ordinary people to interact with their environment, allowing business and institutions to provide on-demand real-world services through digital means. Unlike the World Wide Web, the Internet of Things is not based on a set of interoperable technologies, but every application pick the best technologies to overcome its technical challenge in the best way according to some criteria. Thus, multiple protocols for any layer of the OSI stack [11] were developed, leading to different application platforms that are not interoperable without a platform-dependent bridge.

This chapter is organized as follows: Section 1.1 categorize IoT applications into four common platform architectures that have different requirements in term communication protocols. Section 1.2 analyze the various options for connecting the things at the MAC layer. As for the network and transport layer, some applications might use the IP stack, whereas others might use application specific protocols, and all these solutions are investigated in Section 1.3. On top of the IP stack it is possible to

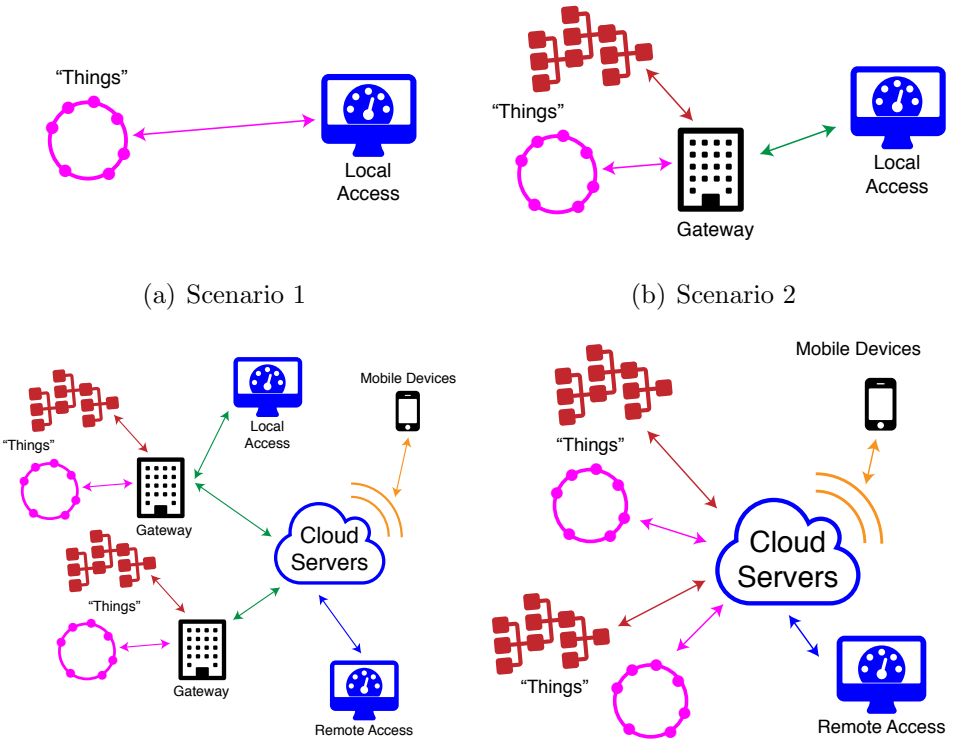
transfer some standards-to-be application protocols, which are discussed in Section 1.4. Finally, Section 1.5 summarize our findings.

1.1 Platform Architectures

Any Internet of Things application is based on one of following platform architecture, depending on how the user can access the virtual representation of a thing, which is often called node. Every of this platform architecture have different requirements in term of device-device communication, group communication, routing, quality of service, IP compatibility, and remote push availability. In the following paragraphs, we discuss what are the different requirements for each of the architectures.

1.1.1 Local User

Fig. 1.1(a) shows the first architecture, which we identify as "Local User". In this architecture, the User access the things directly, usually using an off-the-shelves handheld devices, such as a smartphone or a tablet. In order to support such on-the-go access, these devices must be compatible with what most handhelds supports, which limits the possible choices. The best in class protocols that are supported by the majority of handhelds are Bluetooth Low Energy (BLE) [12] and the 802.11.x family (WiFi) [13]. Even if new standard emerge, the applications based on this architecture will always be limited by mainstream technologies. Finally, in this scenario the things do not communicate between each other.



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

Figure 1.1: Network topologies in the Internet of Things. (a) shows a setup where "Things" are able to communicate directly to users. (b) shows a simple setup where different things communicate locally with users through gateway. (c) shows the most common setup where different Things communicate with users connected to a remove server through a gateway. (d) shows a setup where "Things" are able to communicate with remote users without the need of gateway.

1.1.2 Gateway User

Fig. 1.1(b) shows the second architecture, which we identify as "Gateway User". In this architecture, users and things might be connected to a local Gateway using different protocols, possibly at all level of the OSI [11] stack. Thus, the Gateway is responsible to mediate the between the users and the things.

In this architecture, the nodes might send messages between each other, or provide connectivity to others in case some nodes cannot communicate with the Gateway directly. As an example, in the 802.15.4 [14] MAC protocol the nodes can be deployed in various topologies, as we will discuss in section 1.2.

1.1.3 Remote User through Gateway

Fig. 1.1(c) shows the third architecture, which we identify as "Remote User Through Gateway". In this architecture, users are not directly connected to the Gateway: this its extremely frequent as most gateway do not have a public IP and domain name for security reasons. Thus, users need a Remote Server to proxy the connection to the Gateway, which is trusted and can handle a higher level of security. This remote server is often deployed in the so-called "cloud". Finally, if the mobile application want to support push notification [15] it cannot be contacted directly from the Gateway, as mobile devices are not connected with a public IP address and they must initiate the communication to the Remote Server. The link between the Gateway and the Remote Server is discussed in Sec. 1.1.5.

1.1.4 Remote User

Fig. 1.1(c) shows the fourth architecture, which we identify as "Remote User". In this architecture, the things are directly connected to the Internet, which means that non-IP network protocol cannot be used, e.g. Bluetooth. Moreover, there is no local access to the things, that must be always accessed from the cloud. The major issue with this architecture is privacy: we need to protect the communication and limit how the sensed data can be used. Moreover, in this architecture the communication between things is more expensive than the precedent two cases with the Gateway involved.

1.1.5 Gateway-Remote Server link

The direct link between two things is obviously not enough to create a global Internet of Things: this new paradigm will come to the full potential when every thing is connected. In urban areas, most things will be connected through cables, WiFi or some other low latency / low error links. In remote or inaccessible areas, things will be connected through cellular network, e.g. GSM, or through a GEO satellite link. The influence that these interconnections have on IoT application is studied in Chapter 3 and 3.

1.2 MAC Protocols

Any Internet of Things application must specify how the things are interconnected, starting from the bottom of the OSI stack [11]: numerous MAC protocols are available for use depending on the application requirements. In this dissertation the PHY protocol are not considered, as

for an application developer point of view these are usually bound to a specific MAC protocol.

The Internet of Things pose numerous challenges in creating the basic interconnections between two things: network topology, cost, latency, application throughput and security are the most common directions in which MAC protocols are evaluated. The most used MAC protocols for the IoT are 802.15.4 [14], WiFi [13], and Bluetooth Low Energy [12].

1.2.1 802.15.4

IEEE 802.15.4 [14] is the most widespread protocol for Wireless Sensor Networks (WSN). In 802.15.4 the nodes connects to a Coordinator in a star, tree or mesh topology. The Coordinator might be application-specific or provide Internet connectivity to the nodes.

Estimating application latency in 802.15.4 networks is application specific, because the packets to be receive are polled by the nodes at specific intervals. As most nodes are battery-operated, they might sleep for hours between turning on the radio again. However, the latency for sending such message is in the order of 2.4 ms and 6.02 ms [16], plus any retransmission needed in case of errors. In [16], the authors measured that an 802.15.4 network can reach 163 kbit/s maximal throughput.

802.15.4 optionally supports the Advanced Encryption Standard (AES) algorithm [17], both for authentication and authorization. The keys can be shared with the whole network of sensors (network shared keying), or can be shared by a pair of nodes (pairwise keying), or can be shared only with a group of nodes (group shared keying).

1.2.2 WiFi

In [18], the authors have measured that low-power Wi-Fi (LP-WiFi) provides a significant improvement over typical Wi-Fi on both latency and energy consumption counts. According to the authors, LP-Wifi consumes approximately the same power as 802.15.4 for small packets but it performs better for large packets. Thus, it is possible that a LP-Wifi approach will emerge as a solution in some sensor network applications.

1.2.3 Bluetooth Low Energy

Since its first version, Bluetooth has been used to control from cars to wirstbands and in general most of our personal devices. Bluetooth imposes a star topology to networked things, placing the user-controlled device as the center. Bluetooth has a very short range, and it is suitable for Personal Area Network devices. As an example, Bluetooth is used for headsets, speakers, printers, and quantified self devices.
Low Bluetooth Energy (BLE) offers a low power alternative to the standard Bluetooth, reducing latency to 6ms and application throughput to 236.7 kbit/s [19]. However, it reduces the power consumption while connected to 0.024 mA, giving an expected battery life of 1 year over a coin battery [20].
BLE chips are cheaper than 802.15.4 chip, but it requires more processing power [21]. Moreover, BLE does not support the Internet Protocol (IP), but it discussed within IETF [22] and in [23] a first implementation is discussed.

1.3 Network and Transport Protocols

The network and transport layers of any Internet of Things applications are extremely important to achieve interoperability between different solutions. On one hand, the user is always connected through the Internet stack, which right now involve the Internet Protocol (IPv4) [24] as network protocol and the Transmission Control Protocol (TCP) [25] as transport protocol. On the other hand, the things might be connected through different protocols, such as ZigBee: if the thing has no Internet support, then it is responsibility of a gateway to expose on the local or global network.

1.3.1 IP

The impressive number of real-world things that we aim to connect creates challenges for the whole Internet: at the network level, the major issue of IPv4 is its byte address field length, which is only 32 bits. As of today, all the possible addresses are allocated [26]. The next version of the Internet Protocol, called IPv6, uses 128 bits for its address field, which allows plenty of addresses for all the possible things. However, IPv6 headers are much larger than IPv4, and the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [27] standard specifies how to compress them to fit in a 802.15.4 frame. A similar technique is proposed to allow the transmission of IPv6 packets over Bluetooth Low Energy [28]. The IP stack allows two transport protocol, TCP and UDP.

TCP

The Transmission Control Protocol (TCP) [25] creates a communication channel between two remote parties, a client and a server. TCP is the basis of the Worl Wide Web, as it creates a reliable communication channel between the parties by involving retransmissions. In [29], the authors discuss the reasons why TCP is not sufficient as a transport protocol for the IoT: connection setup, congestion setup, and data buffering makes TCP expensive to send end-to-end messages on battery-powered devices that are in a sleeping state. Thus, TCP cannot be used on sensors that have an estimated battery lifetime of years.

UDP

The User Datagram Protocol (UDP) [30] offers the minimum set of features for a transport protocol: application multiplexing, via port numbers, and integrity verification, via checksum, of the header and payload. The main difference with TCP is that UDP is not reliable: the application is responsible for handling the retransmissions of lost messages. Thus, IoT applications can customize the trade-off between reliability, congestion control, and battery consumption.

1.3.2 ZigBee

ZigBee is a network, and application protocol suite that aims to solve the industry and home automation problem. Thus, it is not compatible with the Internet stack, and requires a gateway. ZigBee uses 802.15.4 as its MAC layer, thus it supports start, tree, and mesh topologies. At the network level, ZigBee supports network routing through the Ad hoc

on-demand distance vector (AODV) routing algorithm [31]. The ZigBee standard includes no transport layer, but it have several application profiles that specifies the functionalities of the things: these profiles dictates the available data across different vendors.

1.4 Application Protocols

At the application layer, thing-driven approaches leverage binary protocols and data formats that are specifically designed for machine to machine communications. These protocols and data format introduce little overhead, minimize battery consumption but are usually not reused in other fields. The benchmark against which all these protocols should measure is HTTP, as it is extremely familiar to the developers.
The most widespread open protocols specifically designed for the IoT are MQTT [32] and the Constrained Application Protocol (CoAP) [33], which are based on TCP and UDP, respectively. MQTT is a classical publish/subscribe protocol, while CoAP is a request/response protocol based on the REST pattern. Both MQTT and CoAP support the same primitives: MQTT focuses on sending and receiving updates, while supporting basic syndication; CoAP focuses on syndication, while supporting a basic notification mechanism [34].

1.4.1 HTTP

Hypertext Transfer Protocol (HTTP) [35] is the basis of the Web, and it is used also to integrate different software applications using the Representational State Transfer pattern [36], where every resource is globally identified by an Uniform Resource Identifier (URI) [37]. As of today,

thousands of businesses offer REST APIs for creating new applications. It is also important to note that HTTP is a text-based protocol with many data types being transferred in text format. HTTP is designed to support caching and several approaches exist to syndicate data. Recent advances, such as WebSockets [38] and Server-Sent Events [39], allow to build soft real-time Web applications.

1.4.2 MQTT

The MQTT [32] protocol is fast, lightweight, power efficient and implements various levels of Quality of Service (QoS). MQTT is based on TCP, so it provides standard TCP delivery reliability, in addition to its own QoS mechanism. MQTT implements a classic publish/subscribe (pub-/sub) pattern with a central broker. The protocol revolves around the concept of *topic*, where clients might publish updates or subscribe to for receiving the updates from other clients. The MQTT community claims that a pub/sub protocol is what is needed to build a true IoT. MQTT can also tunnelled over a WebSocket, thus allowing web client to communicate with the nodes with extremely low latency.
MQTT libraries have been provided for all major IoT development platforms, and for several programming languages (C, Java, PHP, Python, Ruby, Javascript) and for the two major mobile platforms (iOS and Android). The MQTT protocol is being standardize at the Advancing open standard for the information society (OASIS) consortium [40] [41].

1.4.3 CoAP

The Constrained Application Protocol (CoAP) [33] is an implementation of the Representational State Transfer pattern [36] (REST) and it is similar in HTTP from a high-level point of view. However, it is implemented over UDP and it is binary. Thus, it significantly reduces the overhead for battery-powered devices while guaranteeing HTTP compatibility through a proxy. CoAP supports a basic notification mechanism, the observe option [34], which is similar to the HTTP Server-Sent Events [39]. Both these techniques create a unidirectional stream of notifications.

1.4.4 MQTT-SN

The MQTT-SN protocol, formerly MQTT-S, is the port of the MQTT protocol over the UDP transport [42]: it is semantically compatible, and a MQTT-SN device can connect to a standard MQTT broker via a protocol translator. MQTT-SN is optimized for the implementation on low-cost, battery-operated devices with limited processing and storage resources such as wireless sensor devices connect via 802.15.4 and 6LowPan. At the moment of this writing the license of the MQTT-SN protocol is unclear [42].

1.5 Conclusion

In this chapter, we analyzed all the possible architectures and protocols to support any Internet of Things applications. This state-of-art review served as a basis for all the research work in the following chapters: in Chapter 2 we present a novel approach for bridging between all the

application protocols presented in Section 1.4; in Chapter 3 we present an analysis of the latency for those protocols in various delay and error conditions; finally, in Chapter 4 we analyze the cost of directly controlling a remote device over a long round trip time link using Real-Time Web technologies.

Chapter 2

Architecture and Primitives of an IoT Platform

The IoT builds upon three main pillars: embedded software in sensors/actuators, Internet protocols, and data management. Each of these fields has its own set of programming approaches, development paradigms and standards. As may be natural at the start of a new field, some groups have tried to force IoT applications to follow only one of the possible paradigms. We argue that the IoT does not need more development standards or paradigms, which developers must learn before being able to work in this field. Instead, it needs a way to make the existing approaches work together. The IoT success requires interoperability.

As of today, there is a single global system that could act as a starting point for the IoT, as it is highly interoperable, integrating all different kinds of information sources, and well understood by developers worldwide to create a multitude of powerful applications: the World Wide Web. We use the Web to work, for entertainment and to interact socially: tomorrow, we will use the Web to interact with the physical world. Thus,

our goal with the *Ponte* Project is to support existing Web developers in developing IoT applications, abstracting out binary protocols and data formats incompatibility. We aim to make the IoT usable in real life by lowering the barrier for existing developers. *Ponte* is not another layer between the application and the device, but it is a replacement for a component already in place: the broker.

Interoperability between protocols and devices is not a new problem, albeit it is a key factor for the success of an IoT system. In this chapter, we approach this problem in a new way by defining several primitives to supports interoperability, independently from the used technology. We argue that the primitives allow to support the most common protocols of today, MQTT and CoAP, and those of tomorrow. Then, we present a novel architecture for bridging between the Web and the things, by supporting both the Representational State Transfer (REST) [36] and publish/subscribe [43] patterns. Moreover, we discuss the various options regarding data format in the Internet of Things, providing a recommendation. Finally, we introduce our implementation of that architecture in the context of the *Ponte* project, which is also an Eclipse Foundation project [6]. Thanks to *Ponte*, we evaluate both the bridging and the protocol themselves, a novel analysis that has not been done before.

2.1 Related Work

The Internet of Things survey prepared by Atzori et al. [29] highlights three main research areas of the Internet of Things: devices, Internet, and Semantic Web. The devices area is mainly related to physical things. The Internet area focuses on protocols for interconnection. The Semantic Web

area addresses how to integrate and process the data coming from the IoT.

In [44], the authors highlight how the application logic is moving from the devices to the cloud, allowing mashups regarding both the configuration and the sensing/actuating phases.

The Web of Things (WoT) movement [45] envisions a world where every thing host its own web server. However, hosting a server on every device is unfeasible if the devices are battery-powered, because the Web is based on a 100% duty cycle. Thus, the WoT leverage the syndication capabilities of the Web to cache the latest data. Besides, we usually interact with public-facing Web applications, while a 'thing' is usually not accessible from the Internet, as – due to security and privacy reasons – it has no public IP address. Moreover, the authors of [46] highlight the challenges in integrating multiple WoT hub.

Several cross-protocol architectures have been proposed for the IoT, such as [7] or [47]. In both the approaches, the authors focus on specific protocols and data formats, without abstracting a general interaction model. In addition, the latter does not cover publish/subscribe on the Web which is possible in our architecture.

In [48], the authors evaluate the performance of a system based on an embedded implementation of Constrained Application Protocol (CoAP) [33] that uses Efficient XML interchange [49] as the data format. Even though they consider only local networks, their results of CoAP performance are coherent with ours.

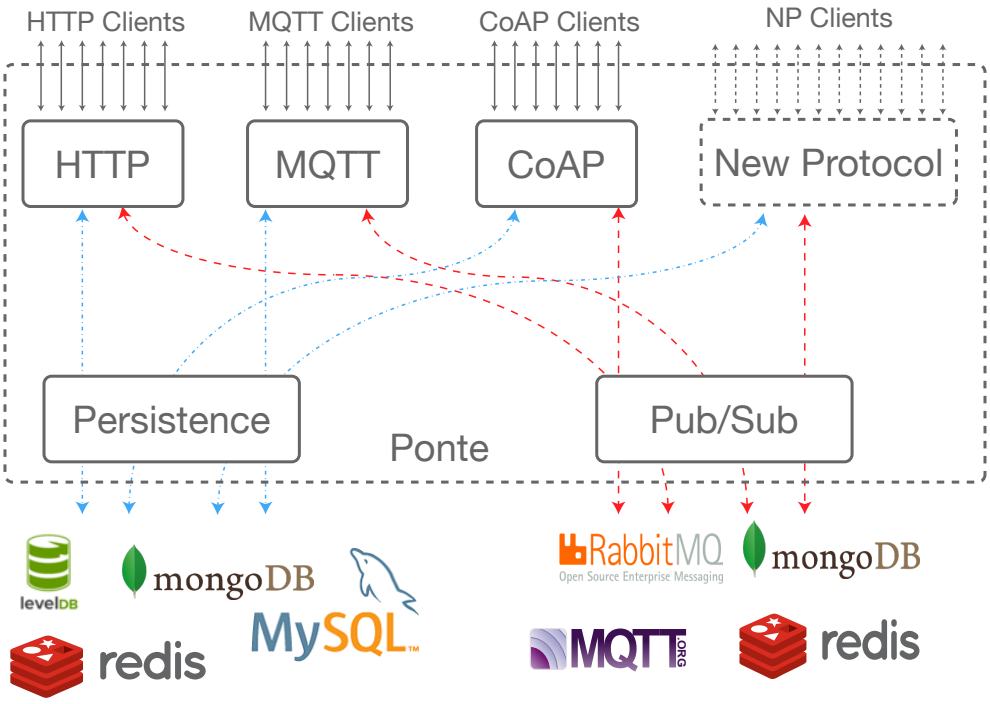


Figure 2.1: Architecture of a bridge for the Internet of Things that enables communication between devices supporting different protocols. This architecture can be deployed on top of various databases and brokers, and can be extended to support more protocols. The list of supported databases and brokers is not exhaustive.

2.2 Protocols for the Internet of Things

Hypertext Transfer Protocol (HTTP) [35] is the basis of the Web, and it is used also to integrate different software applications using the REST pattern, where every resource is globally identified by an Uniform Resource Identifier (URI) [37]. As of today, thousands of businesses offer REST APIs for creating new applications. It is also important to note that HTTP is a text-based protocol with many data types being transferred in text format. HTTP is designed to support caching and several

approaches exist to syndicate data. Recent advances, such as WebSockets [38] and Server-Sent Events [39], allow to build near real-time Web applications.

Thing-driven approaches leverage binary protocols and data formats that are specifically designed for machine to machine communications. These protocols and data format introduce little overhead, minimize battery consumption but are usually not reused in other fields. The most widespread open protocols for the IoT are MQTT [32] and the Constrained Application Protocol (CoAP) [33], which are based on TCP and UDP, respectively. MQTT is a classical publish/subscribe protocol, while CoAP is a request/response protocol based on the REST pattern. Both MQTT and CoAP support the same primitives: MQTT focuses on sending and receiving updates, while supporting basic syndication; CoAP focuses on syndication, while supporting a basic notification mechanism [34].

2.3 Primitives for the IoT

We argue that any Internet of Things system protocol must solve a very delimited set of problems: data delivery, discovery and duty cycle. In the IoT, an enormous quantity of data must be collected, delivered and syndicated. In order to support real-time interaction, the users need to retrieve the latest thing status, and then all updates from there. The discovery of new thing and data is also extremely important, as users or other things might not know directly what things to query, e.g. a building might be composed of hundreds of things. Finally, many devices in the IoT are battery-powered: their duty cycle is less then 50%, so Users needs

a way to send offline commands. In the following, we name the 'status of a thing' as a resource, as it is possible to support virtual devices.
In order to solve these problems in the most generic way, we argue that it is possible to define a set of primitives that can be used to support any IoT protocol:

1. storing and looking up a resource, as we need syndication to support offline behavior and REST clients;

2. publishing a resource update, and subscribing to one or more resource updates, as we want real-time notification of real-world events, with wildcard support for discovery;

3. storing and forwarding offline updates, in order to receive notifications when a device is offline.

These primitives are based on two separate components that have been usually left separated: persistences and messaging. In particular, the first primitive requires the persistence provided by a database, while the second requires messaging, e.g. a publish/subscribe broker. The third needs both.
Fig. 2.1 shows an architecture that realizes the primitives: it combines persistence and pub/sub messaging to support HTTP, MQTT and CoAP, but it can be extended to support more. On the implementation side, different databases can be used for storage data, while different message queues, or brokers, can be used to deliver real-time or offline messages. In the following paragraphs we discuss how the various protocols map to the defined primitives.

2.3.1 HTTP Support

HTTP is an implementation of the REST pattern, where a resource can be manipulated using standard verbs: GET, PUT, POST and DELETE. This allows to syndicate the things resources in a very straighforward way over a database, by indexing their URIs. Moreover, it can support Server-Sent Events [39], as a way of delivering resource updates in a very efficient way. HTTP do not support offline messaging natively.

2.3.2 MQTT Support

MQTT offers three types of messaging: normal, retained and offline. Normal messages is a classical publish/subscribe implementation, where multiple listeners receives messages published on a specific topic by multiple publishers. Retained messages are more interesting: it is possible to set a message that can be stored inside the broker and deliver when a new subscribe is made. Offline messages allows a subscriber to receive the messages that where published when it was offline, thus the messages need to be stored inside the broker.

MQTT messaging can be mapped one-to-one to our primitives: retained messages are used to support the syndication of a resource (primitive 1), while normal messages can be used for updates (primitive 2). Moreover, it supports natively offline messaging (primitive 3). Even though MQTT supports all primitives, there is no separate API for accessing retained messages.

2.3.3 CoAP Support

CoAP is very similar to HTTP, as it allows to GET, PUT, POST or DELETE a resource. Storing the state of a resource is usually done in memory, in the file system or in a database. As previously noted, it also specifies an 'observe' option [34] in which resource updates are sent to the client, as soon as they happen. Finally, it is being discussed how to support offline devices in CoAP [50].

2.4 Implementation

Ponte is the implementation of the architecture described in Section 2.3. One one hand, *Ponte* can be used as the device-facing server in any IoT system, replacing the pub/sub broker or REST api provider. On the other hand, *Ponte* supports different backends, to better integrate with enterprise systems.

Ponte is implemented in Javascript over the Node.js [51] framework, which is based upon V8, the Chrome Javascript virtual machine. Node.js implements the reactor pattern [52]: a particular evented I/O system where every computation is executed in response to an event, and this approach allows to build highly concurrent network applications. In order to support the MQTT protocol, we used the MQTT.js implementation [53]. As for supporting CoAP, we developed a CoAP library for Node.js [54]. How the messages are delivered and stored is described in Sec. 2.4.2.

Ponte supports full customization and can be embedded in another Node.js application. Thanks to a simple event-based API, developers can customize *Ponte* to application specific features. As an example, it

is common practice to define HTTP/CoAP URI or MQTT topic formats to identify a 'class' of devices. Thus, *Ponte* simplifies the architecture of a IoT application.

Our primitives for IoT protocols can be implemented on top of a combination of a databases and a message broker. Thus, our architecture can be implemented on top of several different databases and brokers. As Fig. 2.1 shows, our implementation supports LevelDB [55], MongoDB [56], Redis [57] as databases, but it can be extended to support traditional SQL databases. Moreover, our implementation supports an embedded broker based on the Trie data structure [58], and RabbitMQ [59], MongoDB and Redis as messages brokers, but it can be extended to support others. Our implementation is available at [60] and it is also an Eclipse Foundation project [6].

2.4.1 API

In Ponte, every thing is identified by an HTTP URI, a CoAP URI and a MQTT topic. All the different protocols can be used to interact with the given resource. Moreover, it is possible to add content-negotiation for HTTP and CoAP.

Table 2.1 shows all the possible ways of interacting with a given resource over all protocols. As expected, *Ponte* follows the REST pattern for both HTTP and CoAP. MQTT topics are mapped to resources using retained messages, but it also support non-retained updates.

Ponte does not support streaming of updates over the Web using WebSockets [38], Server-Sent Events [39] or Ajax Long-Polling [61]. However, it supports MQTT over WebSocket: recent browsers can use MQTT directly from Web applications.

2.4.2 Message Delivery and Persistance

Fig. 2.2 highlights the flow of messages from the clients to *Ponte* and vice versa. The messages received from the clients are first passed through the persistance layer, which it store them in the database and queue them for delivery for offline clients. Then, the messages are passed to the pub/sub layer, which forward them to the connected clients that are subscribed to that topic. The multiple supported broker in the pub/sub layer are used to build clustered systems. The default pub/sub broker is embedded and trie-based [58].

A client can receive messages for two reasons: because the resource changed at that point in time, or because a resource changed while the device was offline. In MQTT, messages are also stored for offline delivery if the client fails to acknowledge them and it is disconnected. The multiple supported databases in the persistance layer are used to accomodate different load needs and to build clustered systems. The default database is LevelDB [55].

2.5 Evaluation

In order to verify the feasibility of our approach, we measured the latency introduced by *Ponte*. Latency is a key requirement for IoT applications, as we want our systems to react to real-world events as soon as they happen.

We consider two main scenarios. In the first, 10000 clients updates a resource state simultaneously, and one client receive the update: this scenario is called 'many-to-one'. In the second, one client send an update that must be delivered to 10000 clients: this scenario is called 'one-to-

many'. These scenarios are edge cases of our architecture, as we expect the normal operation to be less instensive in the number of deliveries for a single resource.

As our goal is to measure the impact of *Ponte*, the resources and capabilities of devices are irrelevant for our simulation: we consider the remote devices ideally connected, e.g. with negligible network latency and packet loss probability, and we emulate this situation by running all the clients on the same machine. All measurements are taken on a Ubuntu 12.04 virtual machine kindly provided by LiberoCloud, namely a 'Large' instance with 2 virtual cores and 8GB of RAM, which is a very typical setup that is offered by every cloud computing vendor. Table 2.2 shows the kernel parameters that were changed during the evaluation to support more than 10000 concurrent TCP connections and UDP exchanges. For all measurements, we deployed a *Ponte* configuration with the LevelDB database and the embedded Trie-based broker.

2.5.1 Many to One Scenario

Fig. 2.3 and Fig. 2.4 show the 'many-to-one' scenario. The first notable behavior is that HTTP has a worse performance compared to MQTT and CoAP. The test makes 10000 simultaneous requests to a single server. The HTTP clients open 10000 new TCP connections which are queued up to be served. In contrast, MQTT clients are already connected: we choose this setup as it is the most common for both the protocols. In this setup MQTT offers a better latency than the other protocols. However, our implementation performs slightly worse then the Open Source broker Mosquitto [62], which is provided as reference. Considering CoAP, we see three main levels of latency which are due to the exponential backoff in

case of retransmission. Overall, *Ponte* offers a good latency through all protocols in this conditions.

2.5.2 One to Many Scenario

Fig. 2.5 and Fig. 2.6 show the 'one-to-many' scenario. In particular, delivering to MQTT is extremely performant as all 10000 clients are already connected and waiting for our message. Moreover, *Ponte* configured with the LevelDB database and Trie broker performs better than Mosquitto, which is provided as reference implementation. Delivering to CoAP gives very different results: sending 10000 confirmable packets at the same time produces worse latency, as UDP and CoAP have no flow control. Moreover, some packets are lost and then the exponential backoff rentransmit for confirmable messages is triggered.
We consider our result satisfying, as they prove our architecture is feasible. As said, normal operation is usually in the middle of the two experiments, but both of them shows the approach is sound. The major issue regards high-throughput CoAP networking, as we measured increased latency when the retransmission is triggered due to the lack of congestion control.

2.6 Conclusion

The Internet of Things promise to blend the boundary between real and virtual worlds to improve our life through the use of digital technologies. Software, algorithms and data will help us creating a sustainable economy for the next century. However, we still need to lower the entrance barrier for developing new IoT applications.

Our work defines a set of primitives for an IoT system, and then propose an architecture and its implementation, *Ponte*, to allow developers to interact with the devices easily. *Ponte* supports various protocols, HTTP, MQTT and CoAP, providing a coherent interface between them that does not require the addition of a new layer between the devices and the application. Moreover, we analyze the interoperability problem and we propose a solution to simplify the interaction between the things and data that is already published on the web, such as Linked Open Data. Finally, we show that our approach can handle 10000 concurrent clients on a single server.

Our work can be extended in several directions. Firstly, the interoperability problem needs a commmon data format. We think that adopting Semantic Web technologies might allow to automatically transform and adapt the data to and from the things, solving the evolvability of the transmitted data. Secondly, enforcing privacy is a key research topic in the IoT: by centralizing the data handling in a single place, *Ponte* simplify the development of privacy preserving algorithms. These two directions are discussed in Chapter 5. Thirdly, a peer-to-peer custom database/message broker is needed to support *Ponte* without depending on external software.

Table 2.1: Ponte API over multiple protocols

Protocol	Function	Type	Identifier
MQTT	any changes	retained publish	/{resource/path}
MQTT	any updates	subscribe	/{resource/path}
MQTT	updates to multiple resources	subscribe	/{resource/path}
CoAP	any changes	PUT	/r/{resource/path}
CoAP	last published value	GET	/r/{resource/path}
CoAP	any updates	GET with observe	/r/{resource/path}
CoAP	deletes	DELETE	/r/{resource/path}
HTTP	any changes	PUT	/resources/{resource/path}
HTTP	last published value	GET	/resources/{resource/path}
HTTP	deletes	DELETE	/resources/{resource/path}

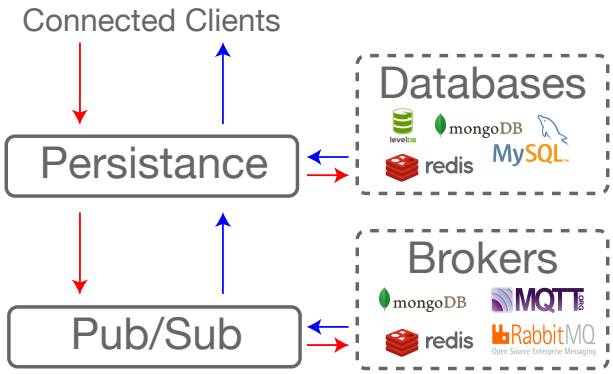


Figure 2.2: Internals of Ponte. All incoming messages transit through a persistence layer that saves them to disk if it is a change of state or there is a matching offline subscription. Then, they pass to the pub/sub layer that routes them to the relevant subscribers. The persistance layer directly sends the offline messages to the reconnecting clients. The list of supported databases and brokers for the persistance and pub/sub layers is not exhaustive.

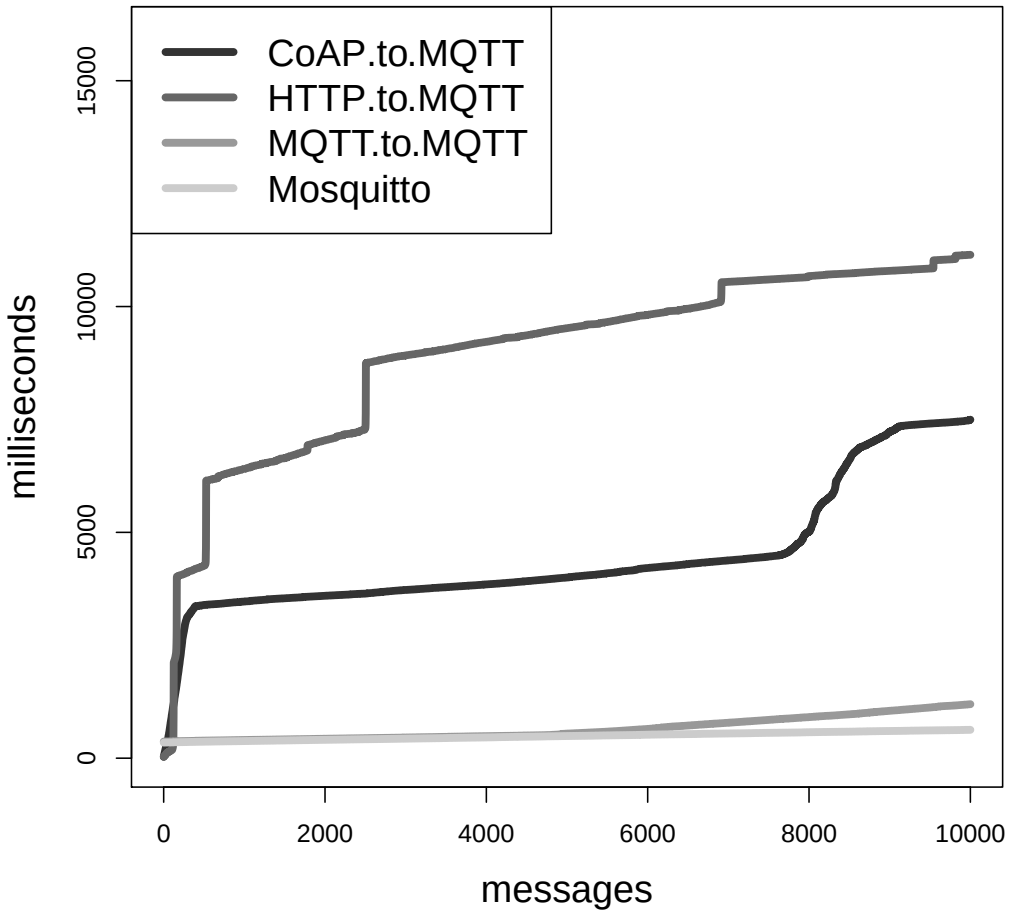


Figure 2.3: Receiving time to MQTT of sending 10000 simultaneous messages to one client. This graph plots the receiving time of each of the 10000 messages. The Mosquitto broker is added for comparison with a pure-MQTT broker.

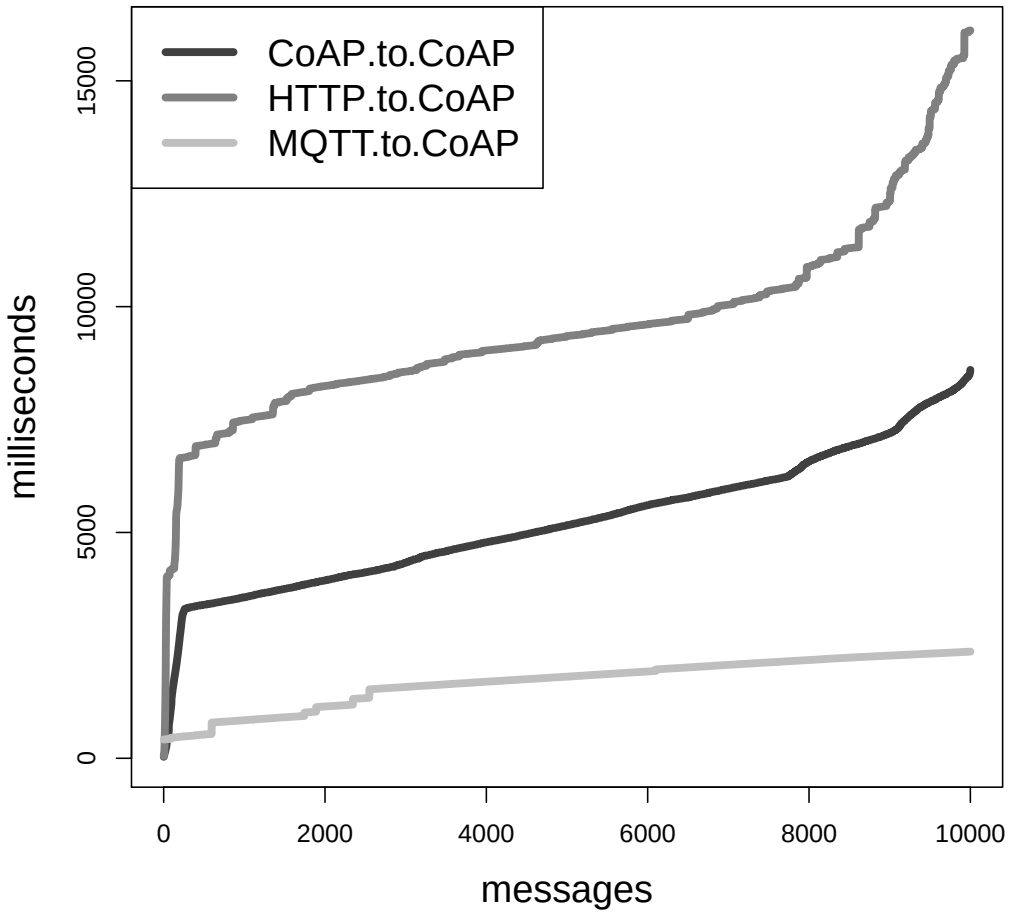


Figure 2.4: Receiving time to CoAP of sending 10000 simultaneous messages to one client. This graph plots the receiving time of each of the 10000 messages. As CoAP is based on UDP, there might be lost messages due to lack of congestion control: the protocol implements a retransmission scheme based on exponential backoff.

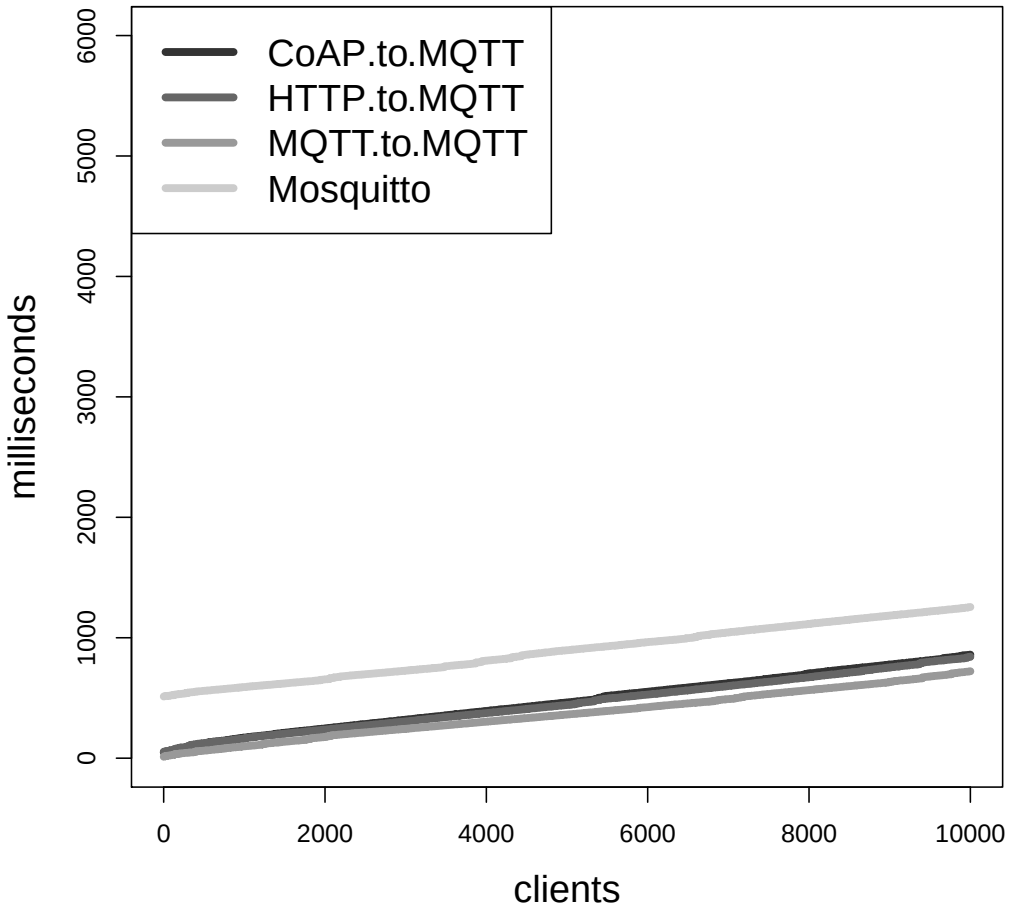


Figure 2.5: Receiving time of one messages to 10000 clients using the CoAP protocol. This graph plots the receiving time of the message in each of the 10000 clients. The Mosquitto broker is added for comparison with a pure-MQTT broker.

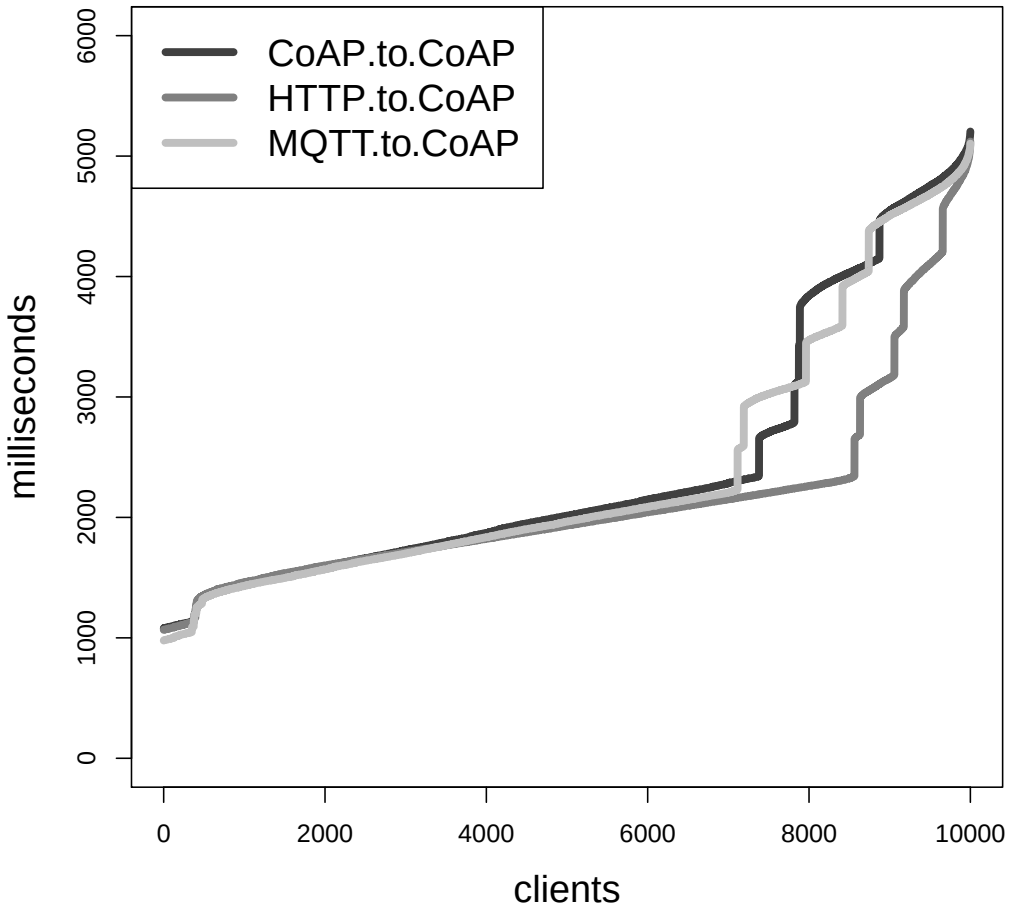


Figure 2.6: Receiving time of one messages to 10000 clients using the CoAP protocol. This graph plots the receiving time of the message in each of the 10000 clients. As CoAP is based on UDP, there might be lost messages due to lack of congestion control: the protocol implements a retransmission scheme based on exponential backoff.

Parameter	Value
net.core.rmem_default	536870912
net.core.wmem_default	536870912
net.core.rmem_max	536870912
net.core.wmem_max	536870912
net.core.netdev_max_backlog	100000
net.ipv4.udp_rmem_min	52428800
net.ipv4.udp_wmem_min	52428800
net.ipv4.tcp_rmem	4096 16384 33554432
net.ipv4.tcp_wmem	4096 16384 33554432
net.ipv4.tcp_mem	786432 1048576 26777216
net.ipv4.tcp_max_tw_buckets	360000
net.ipv4.tcp_max_syn_backlog	10000
vm.min_free_kbytes	65536
vm.swappiness	0
net.core.somaxconn	100000

Table 2.2: Kernel Parameters used in the Tests. The Linux kernel was tuned to not drop UDP packets on reception and to have the best responsiveness for TCP.

Chapter 3

Internet of Things Protocols Analysis over Error and Delay prone Links

The Internet of Things (IoT) paradigm envisions a world where everything is connected and can be remotely monitored and controlled. From forests to factories, we can improve efficiency and reduce costs if relevant data is collected and analyzed. Moreover, remotely controlling our homes can drive a new wave of energy efficiency gains. However, in order to connect everything, we need devices that can run on batteries for years, and this requires protocol optimisation as well. Internet of Things (IoT) application layer protocols are gaining popularity in a wide range of scenarios where low-cost, low-power or resource constrained devices are present. The most diffused protocols are MQTT [32] and the Constrained Application Protocol (CoAP) [33], and both are designed to be low overhead and constrained device friendly.

MQTT is a publish/subscribe messaging protocol built on top of

the Transmission Control Protocol (TCP) [25] and designed to be lightweight. Moreover, MQTT supports offline messaging to handle disconnected clients. MQTT is in standardization within the Advancing Open Standard for the Information Society (OASIS) consortium [40] [41].
CoAP is a request/response protocol which loosely follows the Hyper Text Transfer Protocol (HTTP) [35], but over the User Datagram Protocol (UDP) [30] instead of TCP. Like HTTP, CoAP is being standardized within the Internet Engineering Task Force (IETF). In order to ensure message delivery, CoAP features a retransmission mechanism based on exponential back-off and a maximum of four retransmission attempts.
MQTT and CoAP address different use cases, and to make them interoperable we built the *Ponte* platform [5, 6]. Ponte is a multi-transport Internet of Things broker supporting MQTT, CoAP and HTTP and allows to publish messages from MQTT/CoAP enabled devices to HTTP and vice versa, connecting together the world of things and the Web.
MQTT for Sensor Networks (MQTT-SN) [42] is a variant of MQTT built on top of UDP. MQTT-SN is semantically compatible and can be connected to any MQTT broker through a simple protocol bridge. In [63], the authors have measured that a MQTT-SN performs slightly better than CoAP over 802.15.4 networks. However, MQTT-SN is by far less popular than CoAP, and almost no libraries for it exists. Thus, we do non consider MQTT-SN in this study.
In remote areas lacking in fixed terrestrial network infrastructure, the only available Internet access technology is often represented by satellite links, affected by losses and significant delay. At this time there is no information about MQTT and CoAP performance over satellite IP links in the literature: in this chapter we compare these two IoT protocol in

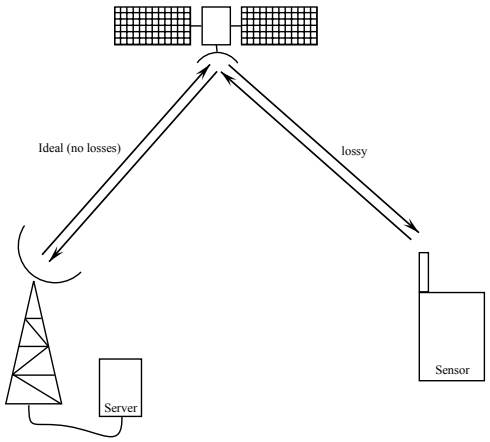


Figure 3.1: Reference architecture for our measurement campaign. We assume that the link between the server and the satellite is error-free, while the link between the satellite and the sensor is error-prone. Both links are subject to delays.

terms of latency and throughput in order to be able to choose the most suitable protocol, depending on the application requirements. MQTT and CoAP as application protocols for smartphone applications are studied in [64]. However, it does not consider links with high round-trip time, especially in presence of packet losses. Our study is focused on evaluating the latency of both application protocols at the increase of link delay and packet loss.

3.1 Application Layer Testbed

In order to compare MQTT and CoAP we consider the network architecture as described in Fig. 3.1: one or more *devices* connect to a server running MQTT or CoAP by the means of a single satellite link.
The satellite connection showed in Fig. 3.1 is emulated using *Dum-*

Table 3.1: Measurement Campaign Parameters

Variable	Value
P_e	$10^{-2}, 10^{-1}$
d_{link}	$[0, 150]$
A	$2 \cdot 10^{-4}, 1 \cdot 10^{-3}, 0.01, 0.1$

mynet [65]. *Dummynet* is a live network emulation tool, originally designed for testing networking protocols: It is able to simulate packet loss, delay and bandwidth limitations. In our simulations, links are affected by delays, while downlink in the Forward Link (FL) and uplink in the Reverse Link (RL) are affected by delay and losses.

The goal of our measurement campaign is to verify in what conditions it is better to use CoAP or MQTT. Thus, the considered simulation parameters are packet loss probability P_e, link delay d_{link}, offered traffic from the devices A. In order to represent different channel conditions the loss probability P_e takes values in the interval $[10^{-2}, 10^{-1}]$, as these are common error rates for low-SNR wireless links used by IoT applications [66, 67]. Finally, delay d_{link} varies from 10 ms to 150 ms. The end-to-end delay is $d_{end-end} = 2 \times d_{link}$.

We evaluate four scenarios depending on the amount of offered traffic through the protocols, under two packet loss probability, as shown in Table 2.2. The first two scenarios $(2 \cdot 10^{-4}, 1 \cdot 10^{-3})$ emulate a case where a single constrained node transmits/receives messages. While in the third and fourth scenarios $(0.01, 0.1)$, we consider a network of 50 devices, transmitting their messages through a single gateway provided by a satellite IP link: this scenario is common in remote areas, where a sin-

gle connection can be reused by multiple devices, potentially connected through multi-hop wireless network.

In order to evaluate consistently MQTT and CoAP, we use confirmable messages in CoAP. The sender should receive an ACK within a timeout, or it starts retransmitting with exponential back-off. CoAP limits the numbers of such retransmissions to 4. In particular CoAP's back-off mechanism is controlled by two parameters [68]: retransmission timeout (RTO) and retransmission counter. The initial value of RTO is set to a random number within the interval [ACK_TIMEOUT, ACK_TIMEOUT*ACK_RANDOM_FACTOR]. The default value is 2 seconds for ACK_TIMEOUT and 1.5 seconds for ACK_RANDOM_FACTOR. The retransmission counter goes from zero to MAX_RETRANSMIT (default value is 4). Retransmission timer starts when a message is sent and is doubled when the timer expires and no ACK is received. The time between the first transmission of a confirmable message and the instant in which no further acknowledgments are expected is called exchange lifetime and is given by

$$\begin{aligned} &\text{EXCHANGE\_LIFETIME} = \\ &\text{ACK\_TIMEOUT} * (2^{\text{MAX\_RETRANSMIT}} - 1) * \\ &* \text{ACK\_RANDOM\_FACTOR} + \\ &+ 2 * \text{MAX\_LATENCY} + \text{PROCESSING\_DELAY} \end{aligned} \tag{3.1}$$

and, with default parameters its value is 247 seconds. In equation 3.1, MAX_LATENCY is the maximum time that a datagram needs, in order to be delivered to destination (default value 100 seconds); PROCESSING_DELAY is the time needed for a device to process a confirmable message and send the corresponding ACK (by default set to

ACK_TIMEOUT).

In the following analysis we are including in our measures only the messages that are effectively transmitted, not the ones that were never received by the server. This might happen for CoAP which, after four unsuccessful retransmissions, reports to the application the server unreachability. On the other hand, MQTT guarantees packets delivery as it is based on TCP.

3.1.1 Implementation

In order to minimize the differences in our measurements due to different software stacks, we base our MQTT and CoAP server upon the *Ponte* Project and on the Node.js framework [51] (version 0.10.20). The clients are emulated using MQTT.js [53] (version 0.3.7) and node-coap [54] (version 0.5.3), which are high-level clients for MQTT and CoAP written in node.js. We motivate the use of the node.js framework because it excels at I/O, while providing a high-level development language and framework. The *Ponte* project support the interoperability between the most common IoT protocols, HTTP, MQTT and CoAP [5, 6]. Fig. 3.2 shows the architecture of *Ponte*: it combines persistence and pub/sub messaging to support HTTP, MQTT and CoAP, but it can be extended to support more. On the implementation side, different databases can be used for storage data, while different message queues, or brokers, can be used to deliver real-time or offline messages.

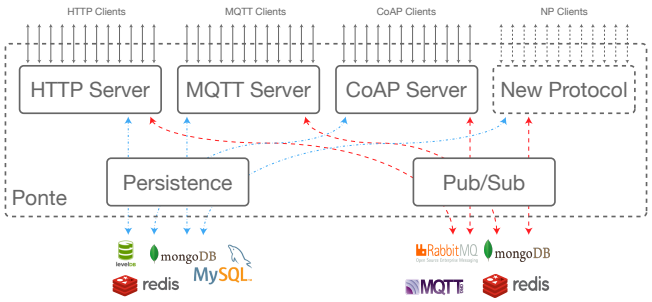


Figure 3.2: The *Ponte* architecture. *Ponte* bridges various Internet of Things application protocols. *Ponte* can be deployed on top of various databases and brokers, and can be extended to support more protocols. The list of supported databases and brokers is not exhaustive.

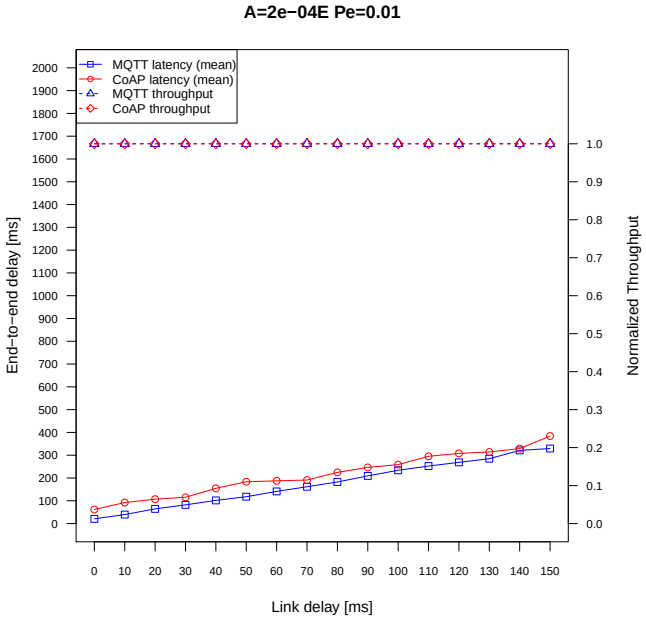
3.2 Results

Fig. 3.3, 3.4, 3.5, and 3.6 show the results of our measurements campaign regarding the application protocol latency. For each combination of delay, probability of packet loss, and offered traffic the presented values in the figures are the average of 1000 repetitions.

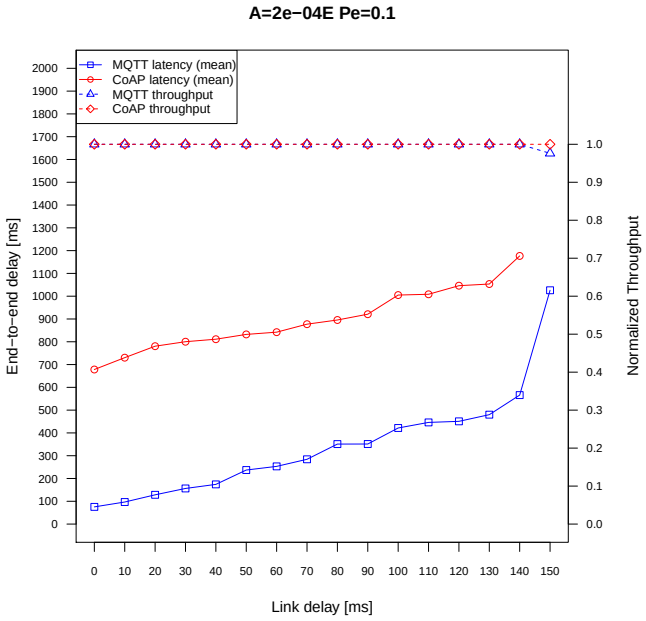
All measurements were taken on a virtual machine gently provided by LiberoCloud, namely a 'Small' instance with 1 virtual core and 1GB of RAM. The VM operating system is Ubuntu 12.04. All major cloud providers offers similar setups. However, we think our results are not influenced by the underlining stack, as our emulation occupied very little of the allocated resources.

3.2.1 Low Throughput

Fig. 3.3 and 3.4 shows the results with the single node scenario. In that regard, Fig. 3.3(a) and 3.4(a) shows that MQTT and CoAP performs equally with low packet loss. However, Fig. 3.3(b) and 3.4(b) highlights

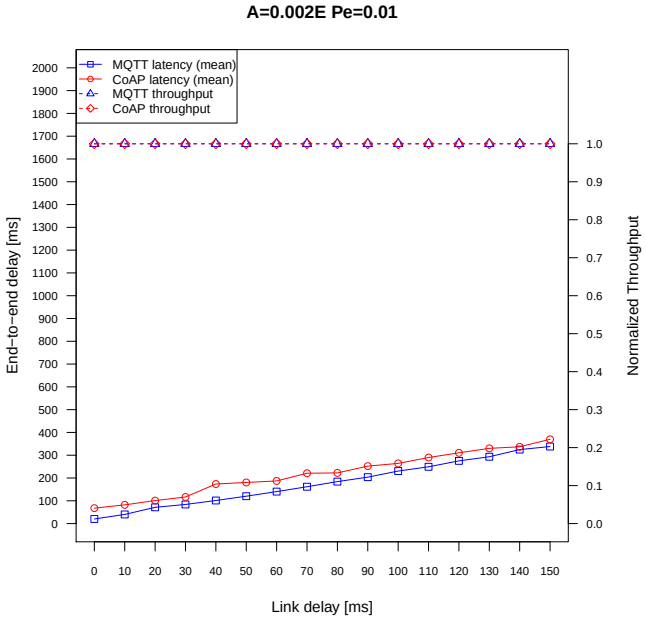


(a)

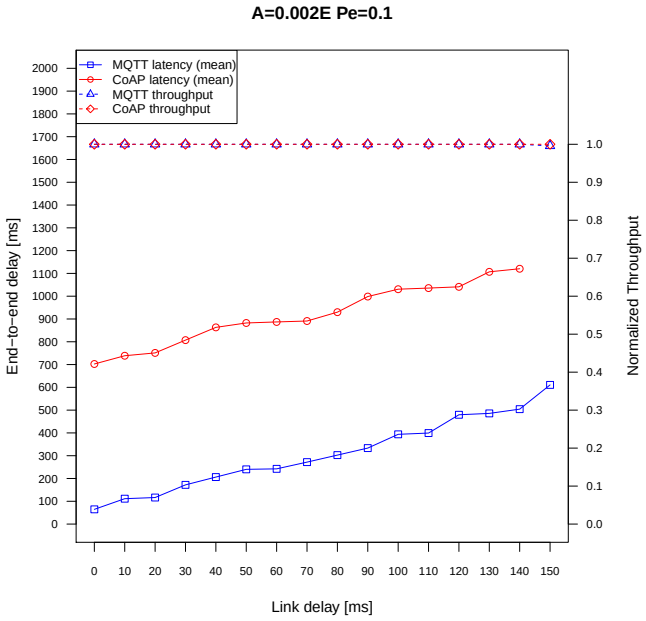


(b)

Figure 3.3: Latency on the Low-Throughput scenario for MQTT and CoAP, which is the case of a single node sending a message every interval of time. These experiments have a lower error probability of 0.01.

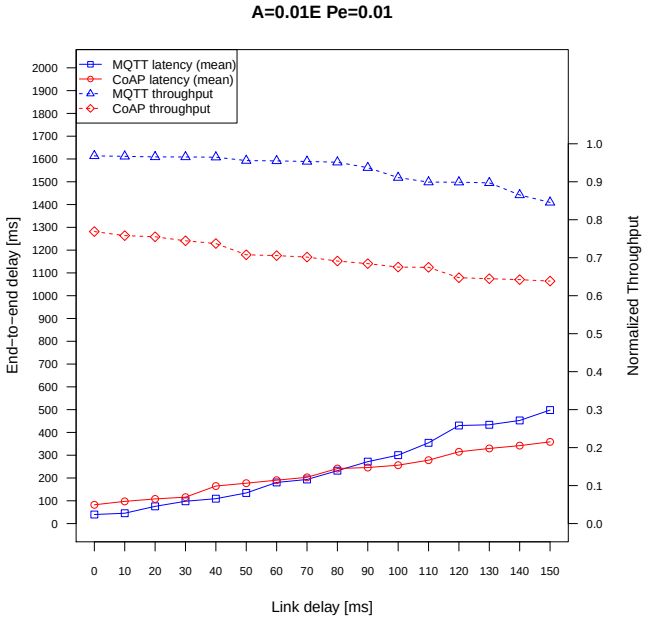


(a)

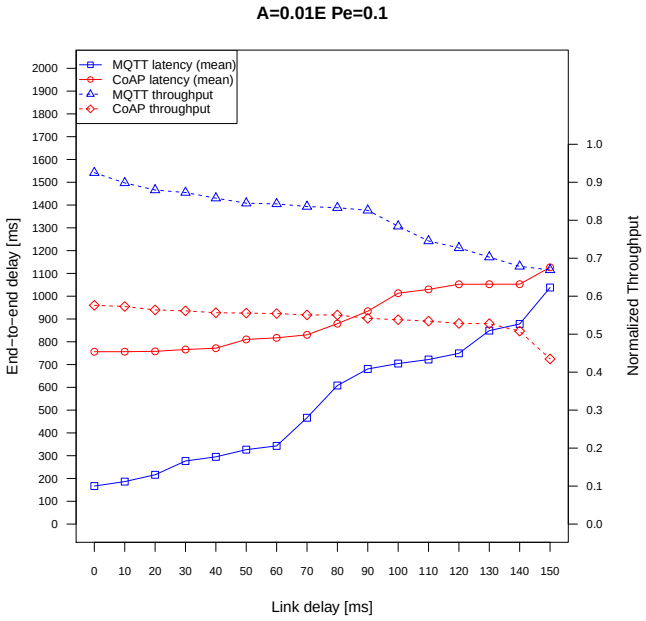


(b)

Figure 3.4: Latency on the Low-Throughput scenario for MQTT and CoAP, which is the case of a single node sending a message every interval of time. These experiments have a higher error probability of 0.1.

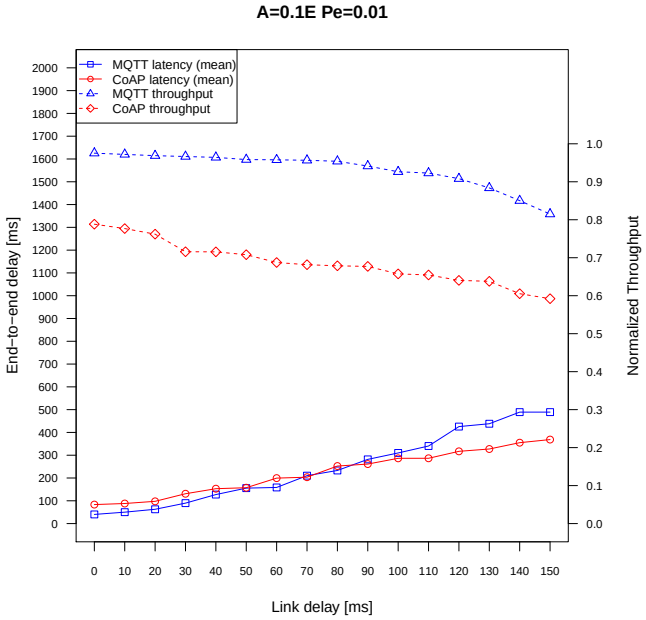


(a)

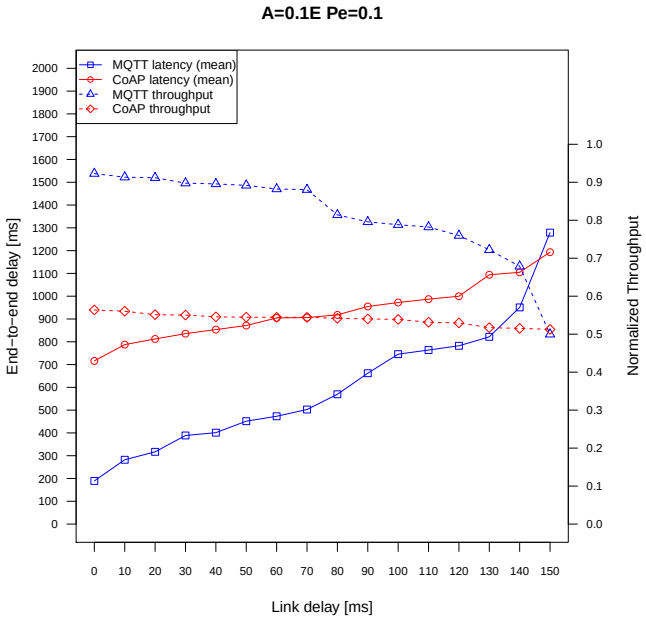


(b)

Figure 3.5: Latency on the High-Throughput scenario for MQTT and CoAP, which is the case of multiple nodes sending messages (50 nodes in this graphs). These experiments have a lower error probability of 0.01.



(a)



(b)

Figure 3.6: Latency on the High-Throughput scenario for MQTT and CoAP, which is the case of multiple nodes sending messages (50 nodes in this graphs). These experiments have a higher error probability of 0.1.

that MQTT performs better in a low offered traffic setup with a more consistent packet loss. We explain this behavior due to the different rentrasmission mechanism of TCP and CoAP: TCP has a more effective rentrasmission scheme that evaluates the maximum Round Trip Time (RTT), while CoAP uses a fixed value of 202 seconds, which follows from a MAX_LATENCY parameter of 100 seconds. The MAX_LATENCY variables is defined by the spec arbitrarily. Moreover, CoAP first retransmission occur between 2 and 3 seconds, which is higher if compared to TCP. Thus, MQTT performs better.

3.2.2 High Throughput

Fig. 3.5 and 3.5 shows the results with the multiple node scenario with an higher offered traffic. In that regard, Fig. 3.5(a) and 3.6(a) shows that CoAP performs slightly better than MQTT at the increase of delay with low packet loss probability. However, Fig. 3.5(b) and 3.6(b) highlight that MQTT performs better than CoAP, in terms of latency, with high offered traffic and high packet loss probability. With both high and low packet loss probability, MQTT offers the best performance in terms of throughput, in the presence of high offered traffic. The worse performance of CoAP in terms of throughput is due to its retransmission mechanism: The parameter ACK_TIMEOUT, by default set to 2 seconds, makes a node wait at least 2 seconds for an acknowledgment before trying to retransmit; In case of a loss, it is necessary to wait at least 2 seconds before a retransmission can occur. Although this phenomenon is not appreciable in the latency graph due to the high number of averaged realisations, the delay introduced by the back-off mechanism causes a throughput degradation.

3.3 CoAP Parameters Tuning

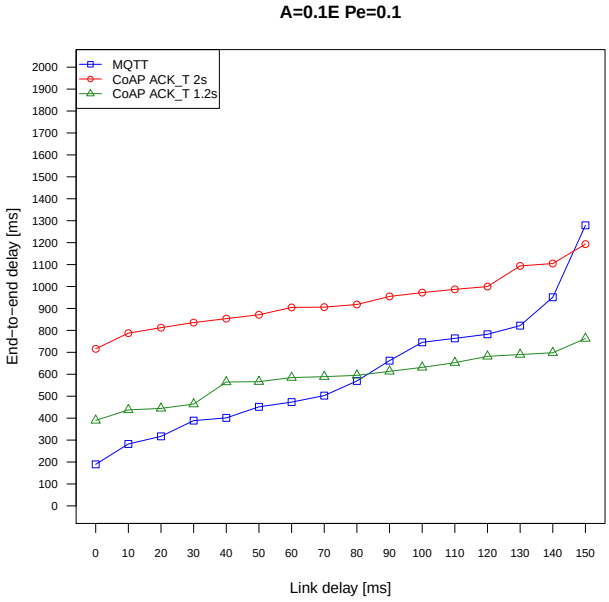
In order to improve CoAP latency and throughput, we reduced the ACK_TIMEOUT from 2 s to 1.2 s and ACK_RANDOM_FACTOR from 1.5 to 1.2. By reducing these parameters, CoAP starts retransmitting sooner and thus the mean latency and normalized throughput should improve. Our experiments, which are highlighted in Fig. 3.7, confirm our hypothesis. Moreover, by tuning the parameters for fast rentransmission, CoAP offers a better latency and same throughput of MQTT over links with delays higher than 90 ms under high traffic conditions.

3.4 Conclusions and Future Work

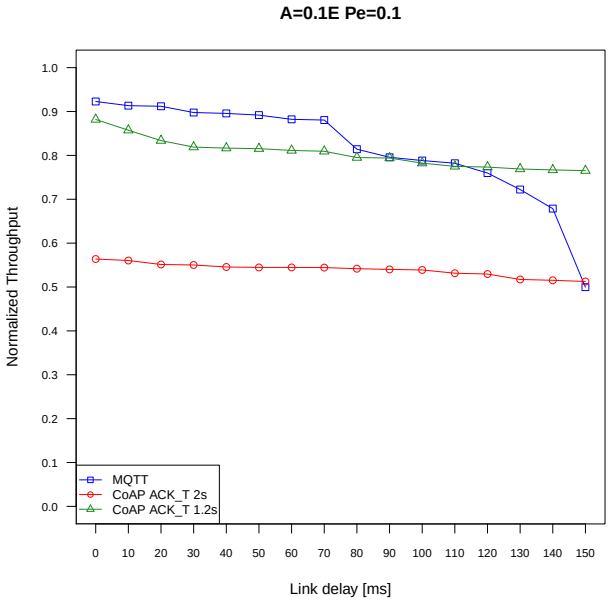
Internet of Things (IoT) application layer protocols, such as MQTT and CoAP, expose different behaviour in high throughput, delay, or lossy conditions. In this chapter, we analyzed the latency and throughput of MQTT and CoAP, two protocols suited for low-cost, low-power and resource constrained devices.

The goal of this research work is to understand in what conditions it is better to use CoAP or MQTT in respect to the increase of delay and packet loss. As our results show, MQTT offers higher throughput and lower latency than CoAP in high offered traffic scenario, in the presence of high percentage of packet loss and delay. However, we showed that by tuning the operational parameters of CoAP, it is possible to overtake the results achieved by MQTT. Thus, tuned CoAP offers better latency and throughput, even if some packets might be completely lost, as only 4 retransmission occur, while TCP guarantees message reception.

Based on our results, it is possible to assess a guideline in choosing the



(a)



(b)

Figure 3.7: Latency and throughput of the CoAP protocol with ACK_TIMEOUT from 2 s to 1.2 s and ACK_RANDOM_FACTOR from 1.5 to 1.2. (a) shows the latency of MQTT, CoAP with the defaults, and CoAP with the tuned parameters. (b) shows the throughput of MQTT, CoAP with the defaults, and CoAP with the tuned parameters.

application protocol for an IoT application depending on the network characteristics. As MQTT performs better in the low throughput scenario with a single device, we suggest the use of MQTT in presence of high delay and a single device. However, if the offered traffic increases, CoAP with the tuned parameters offers the outlined advantages. Finally, it is better to consider that CoAP might support sleepy devices better than MQTT, especially if their duty cycle is hourly or daily, as there is no cost of TCP connection handshaking.

Our work can be extended in several directions. Firstly, it is possible to study how the CoAP parameters impact the latency and throughput, then we might identify some dynamic configuration of these parameters. Secondly, we might study more protocols, such as MQTT-SN. Thirdly, we might want to verify how TCP variants specifically designed for satellite networks affects MQTT performance.

Chapter 4

Latency Analysis of Real-Time Web Protocols over a Satellite Link

The World Wide Web has changed the way we work, play, and live. At the beginning, the Web was used mostly by the research and academic community and consisted of static resources. Researchers would write HTML pages by hand and would distribute them by means of Web servers. Then, when new Web technologies and increased server computational capacity were made available, customized Web pages dynamically generated by programs running on Web servers became popular. Today, Web pages generated by well-crafted programs that allow people to collaborate over the Web are the norm.

At the beginning of the Web, Web pages were delivered to the client ready to be rendered and every user interaction required a page reload. Now, the user resources to display Web pages have grown to such an extent that users can run fully-fledged Web applications within the browser and

pages can be modified without the need of a reload. This is achieved by running Javascript code at the browser. Thanks to the Asynchronous JavaScript and XML (AJAX) technique [69], applications adopting the so called 'Web 2.0' approach can update a Web page in response to user commands or poll the server for changes at regular intervals. AJAX network transfers are faster and lower-overhead compared to the legacy techniques based on HTML. Moreover, the JavaScript Object Notation (JSON) [70] is increasingly preferred to the more complex XML as a transfer language.

The main limiting factor of the modern Web applications is the network-induced latency, which is strictly related to the Round Trip Time (RTT) beteween client and server. As the RTT increases, the responsiveness of Web applications degrades. This is particularly critical when the Internet access is through satellite, as the RTT might be several times the one of terrestrial connections (RTT is about 600ms for a GEO satellite). Performance Enhancing Proxies (PEPs) [71] have been designed to maximise throughput and efficiency of traditional HTTP connections, but might not cope well with the dynamic Web traffic generated by interactive applications. To guarantee usability of Web applications also to satellite users is therefore essential to improve the responsiveness of HTTP.

This chapter aims at evaluating some of the most promising techniques for real-time Web applications in presence of long RTTs, such as those found on a GEO satellite link, by carrying out a series of experiments on real platforms. The techniques considered are used in real-world applications to provide users powerful Web front-ends. Initially, front-end developers focused primarily on page manipulation by means of advanced application program interfaces (APIs), such as the jQuery [72] library.

More recently they addressed more sophisticated frameworks, such as the Model View Controller (MVC) [73], that offer well-known design patterns. The most popular of the new frameworks is Backbone.js [74]. The Web applications that are built with these technologies are comparable, both in terms of features and capabilities, to the traditional desktop applications, but with the additional requirement that the applications depend on remote data, which has to be fetched via AJAX. In order to increase responsiveness, Web applications are allowed to fetch data from the server automatically, anticipating the user inputs. These methods are supported by a number of techniques, protocols and libraries and have gone through refinements along the years. However, these technologies are never been tested with high-delay links, such as those of a GEO satellite and the aim of this chapter is to fill this gap. In this chapter we focus on the following three techniques, which are the most common in the field: Ajax Long-Polling (ALP) [61], Server-Sent Events (SSE) [39] and WebSocket (WS) [38]. ALP is a very efficient polling technique, while SSE is a unidirectional data streaming protocol. WS is the cutting-edge standard that provides a bidirectional data streaming protocol. Even though WS achieves the lower latency, it is still not clear if it will be widely supported, hence the need to assess other widespread techniques.

The chapter is organized as follows. Section 4.1 illustrates the state of art protocols and techniques for building Real-Time Web applications. Section 4.2 describes both the testbed setup for the evaluation of the protocols and the experiments to assess the latency of the various protocols. Section 4.3 presents and discusses the results of the experiments. Finally, in Section IV we draw our conclusions.

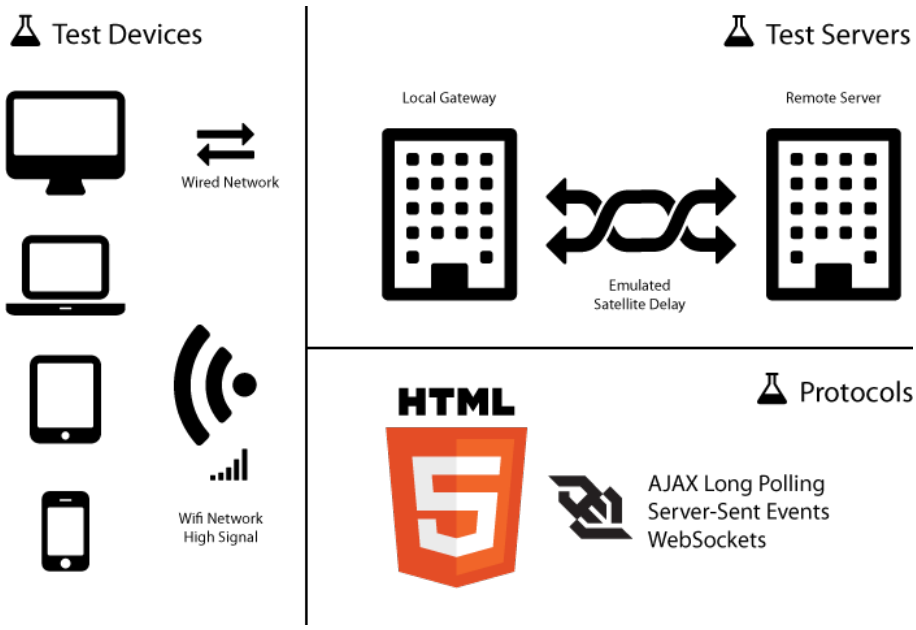


Figure 4.1: High-level architecture of the testbed. Browsers run on either desktop or mobile devices; desktops are directrly connected to the router via Ethernet, mobile devices through a 802.11 WLAN; the delay introduced by the GEO satellite link is emulated by means of Dummynet, running on the router. The tested protocols are Ajax Long-Polling, Server-Sent Events and WebSockets

4.1 Real-Time Web Protocols and Techniques

The development of technologies for pushing data from the server to the browser has encountered several difficulties, mostly due to the lack of support by Web servers and browsers. The core idea behind Server Push is to maintain open a communication channel between the browser and the Web server and to reduce as much as possible the overhead when using this channel w. Since HTTP 1.1 [35] is essentially a request/response

protocol, it does not support this feature natively. In order to alleviate the slow responsiveness of HTTP 1.1 in the presence of real-time interactive Web applications, in recent years a series of proposal have been developed.

Ajax Long Polling (ALP) [61] exploits a loophole inside the HTTP 1.1 spec. After receiving a HTTP request, a Web server is not required to respond immediately; rather it can defer few instants the reply. By changing the timing of request/responses, it is possible to simulate a continuous client-server communication channel. ALP has been used for several years by the most common browsers. Note that ALP keeps open the underlying TCP connection, by using the keep-alive feature of HTTP/1.1. Unfortunately, ALP has a very high overhead due to the transfer of a full HTTP header at each request.

A second technique to implement real-time Web applications is Server-Sent Events (SSE) [39]. Although promoted by W3C, it has seen a limited spread so far. SSE is part of the HTML5 specifications. It defines both the Javascript API and the HTTP payload format to channel data in a HTTP request. Unlike ALP, the response HTTP body is not terminated after delivering a payload, so new events can be forwarded to the client without the overhead of retransmitting the HTTP headers. SSE is a backward compatible specification and most browsers (excluding Microsoft Internet Explorer™ – IE) implement this. Both ALP and SSE allow the server to deliver continuously changing data to the clients. However, servers lack of proper ways to deal with user-generated updates. The only technique that can be adapted to ALP and SSE is an AJAX call, which is in fact a complete HTTP request. Since the number of connections that browsers can open to a given domain is limited, updates cannot be sent concur-

rently. This may cause head of line (HOL) blocks when the rendering of resources already received is blocked by other requests.
In order to support collaborative Web applications, WebSocket has also been included in the HTML5 specifications. Similarly to SSE, WebSocket consists of a Javascript API and a packet data unit (PDU) to be encapsulated in HTTP payloads. Unlike SSE, both the request and the response are left open, thus leading to a full-duplex channel. In addition, the WebSocket specification is much more flexible allowing to transfers of both data and control packets. Web servers are starting to support the WebSocket specification in 2013. In order to provide an evaluation on also the future of the Web, we are also evaluating the SPDY v3 protocol [75], as it is the basis of HTTP 2. SPDY is proven to outperform HTTP/1.1 over a satellite link [76] for normal Web navigation. However, Real-Time Web applications have different traffic patterns than normal Web sites. Thus, this work complements the work of Cardaci et al [76] with the investigation of a different application scenario.
The next Section will describe how the three techniques described above, i.e. ALP, SSE and WS have been evaluated on an emulated GEO satellite link.

4.2 Testbed Layout and Experiment Description

The aim of the experiments is to evaluate how the Real-Time Web protocols (i.e. ALP, SSE and WS) and their different implementations perform in a GEO satellite scenario without losses. In particular, we aim to evaluate for the first time these protocols in a high delay environment,

Browser	Version
Google Chrome	27.0.1453
Firefox	21.0.0
Safari	6.0.4
iOS (Mobile Safari)	6.0.0
Android (Chrome)	18.0.1025

Table 4.1: The various Browser version used in the experiments

considering both desktop and mobile platforms. Moreover, for the sake of generality, the satellite link has been deliberately emulated as a simple delay.

Fig. 4.1 shows the testbed layout. The testbed consists of three parts:

- Several Web clients running different browsers on different hardware platforms (e.g. a Nexus 7' tablet, an iPad 2 tablet and a Mac OS X laptop). The mobile devices are connected through a dedicated Wi-Fi network. Table 4.1 details different client architecture and Web browsers.

- A satellite gateway also emulating the GEO satellite link. This is emulated by inserting the typical propagation delay of GEO satellite (RTT=600ms) by means of the Dummynet [65] emulation tool.

- A Web server, which is run by node.js (version 0.10.2) [51]. There are not intermediaries, such as Apache [77] or Nginx [78], between the server and the client to achieve the least possible latency.

Server and clients run on virtual machines (VMs), generated by using

VirtualBox [79]. The host is a Macbook Air 2011 equipped with 1.8 GHz Intel Core i7 and 4GB 1333Mhz DDR3. The computer has two cores with hyperthreading capabilities sufficient to running concurrently multiple OSes. The Web server VM file-system image is based on the Ubuntu 12.10 GNU/Linux distribution. The maximum amount of VM RAM is 512MB, which is largely more than what needed by the Web server (around 70MB).
The Wi-Fi network is emulated through the Mac OS X "Network Sharing" feature. Both the WiFi network and the satellite link in our experiment do not introduce packet losses. Although this condition may not always be verified in practice, we believe that this is the most common case In order to evaluate protocol performance, we focused on three different scenarios:

- Chat Scenario, the client sends and subsequently receives one message;
- Client Update Scenario, the client sends 100 messages and then it receives one message;
- Server Update Scenario, the client sends one message and then receives 100 messages.

In all our experiments the content of the message is not significant and the payload size is always 39 bytes. Each experiment consists of 50 consecutive transactions over the same TCP connection. Each experiment is in turn repeated five times.
The browser on the client always initiates the exchange and computes the application latency, the transaction's duration. At the end of each

experiment, the browser sends the 50 transaction times to the server, which in turn saves them to disk. The three application-layer protocols (ALP, SSE, and WS) are evaluated on top of HTTP/1.1. ALP and SSE are also evaluated on the SPDY v3. In order to have a benchmark, all experiments are repeated also without the satellite delay.

4.3 Results

The following bar charts show the results of our measurement campaign, averaged over all the transactions. For each test results without GEO satellite delay (Fig. 4.2, 4.4, 4.6, and 4.8), which are our benchmark reference, are displayed first, followed by results with GEO satellite delay (Fig. 4.3, 4.5, 4.7, and 4.9).

Fig. 4.2 and Fig. 4.3 show the latency of ALP respectively without and with a GEO satellite delay. The figures show that Safari is, on the average, less robust than the other desktop browsers to the satellite delay increase. Overall, since the RTT is about 600ms, our measurements show that the ALP protocol requires between two and four RTTs to complete a transaction.

In our experiments on the iOS and Android platforms we observed the delay-jitter increasing when using the Wi-Fi connection without cross-traffic. This indicates that WiFi specific stack characteristic, such as the power saving features of mobile device, may affect significantly packet transmission timings. Future investigations could back this hypothesis.

Fig. 4.4 and Fig. 4.5 show the latency of SSE protocol respectively without and with a GEO satellite delay. SSE performs better than ALP: SSE uses only one or two RTTs in the case of desktop platforms. Two RTTs

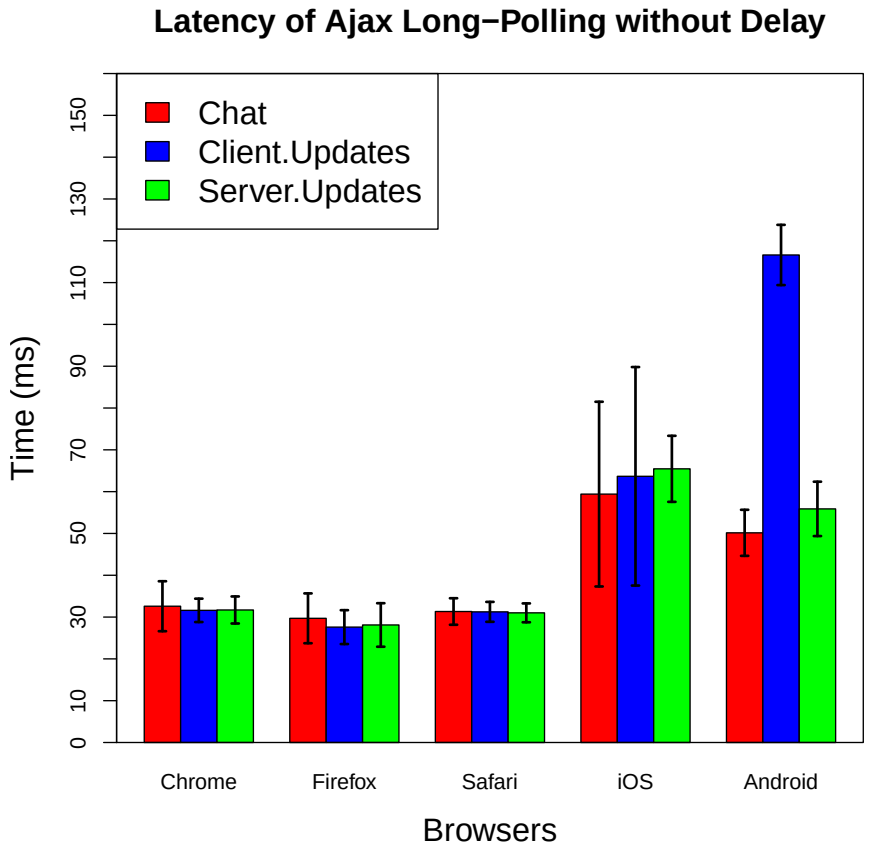


Figure 4.2: Application latency of the Ajax Long-Polling protocol without any emulated delays across all tested browsers. Chrome, Firefox and Safari accessed the server directly, while the iOS and Android browsers through Wi-Fi.

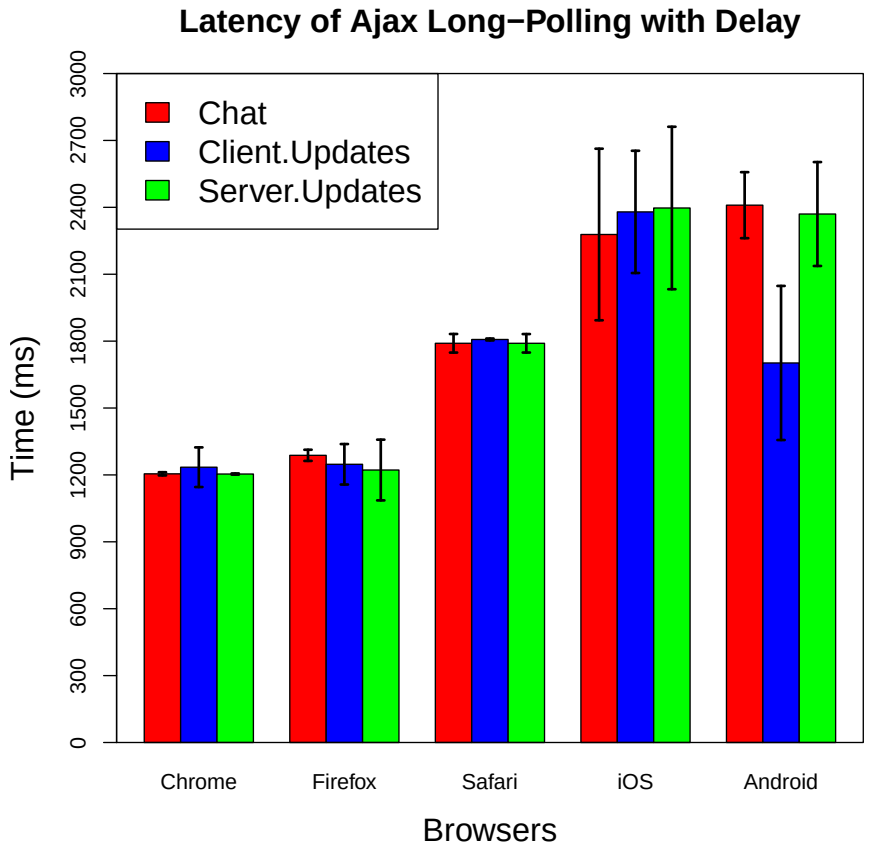


Figure 4.3: Application latency of the Ajax Long-Polling protocol with the GEO satellite delay (RTT=600ms) emulated delays across all tested browsers. Chrome, Firefox and Safari accessed the server directly, while the iOS and Android browsers through Wi-Fi.

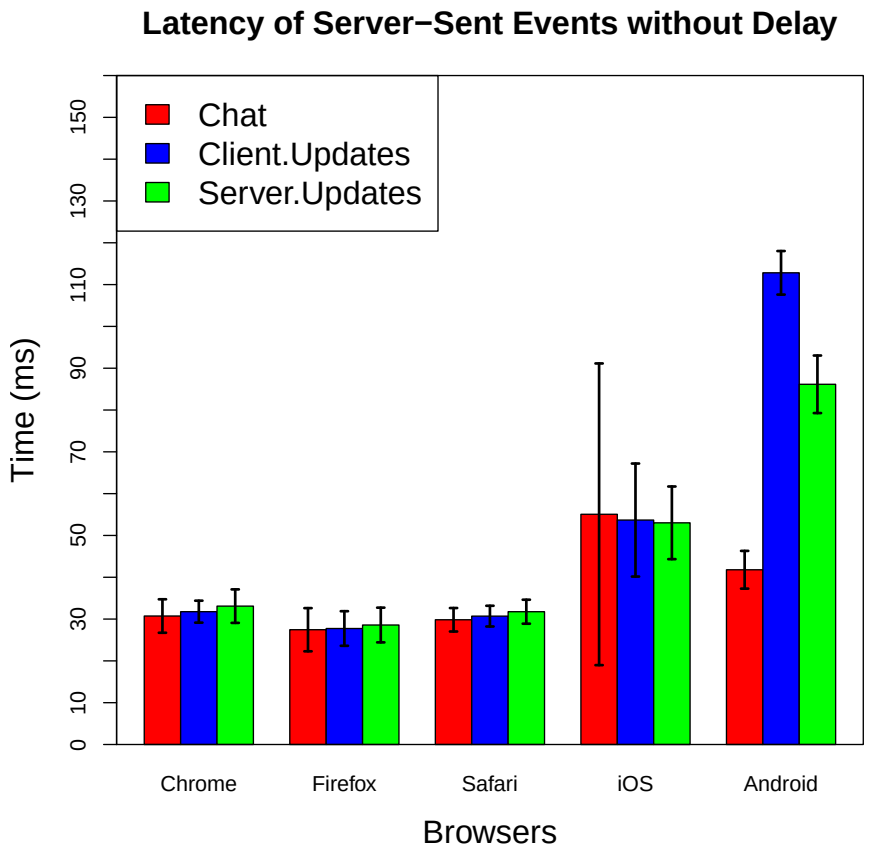


Figure 4.4: Application latency of the Server-Sent Events protocol without any emulated delays across all tested browsers. Chrome, Firefox and Safari accessed the server directly, while the iOS and Android browsers through Wi-Fi.

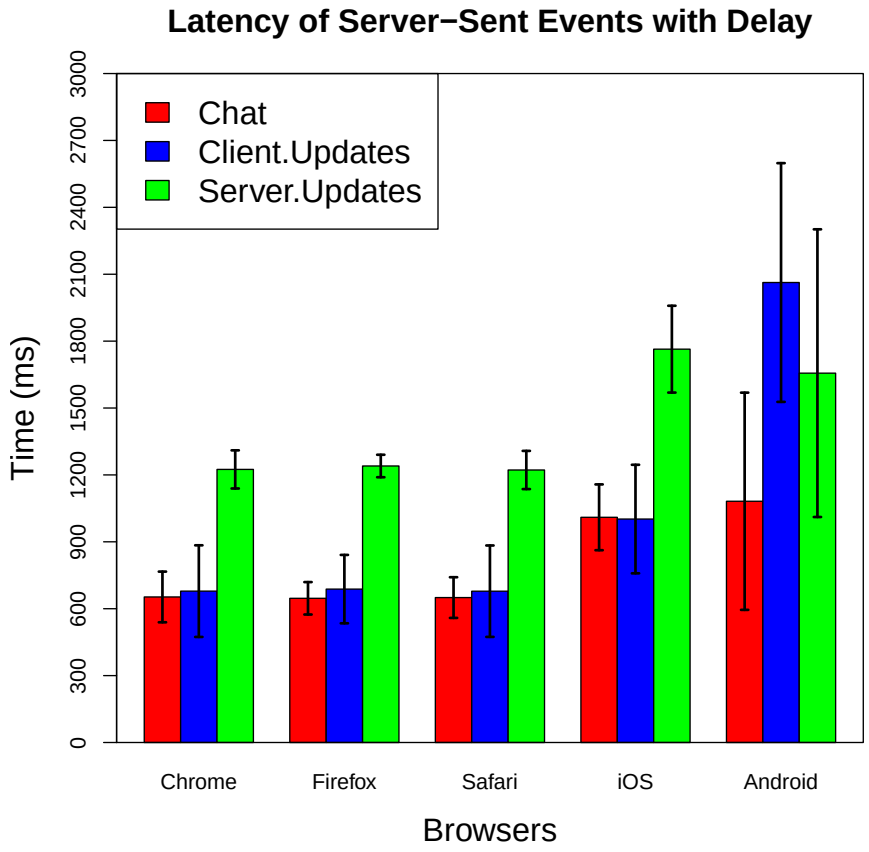


Figure 4.5: Application latency of the Server-Sent Events protocol with the GEO satellite delay (RTT=600ms) emulated delays across all tested browsers. Chrome, Firefox and Safari accessed the server directly, while the iOS and Android browsers through Wi-Fi.

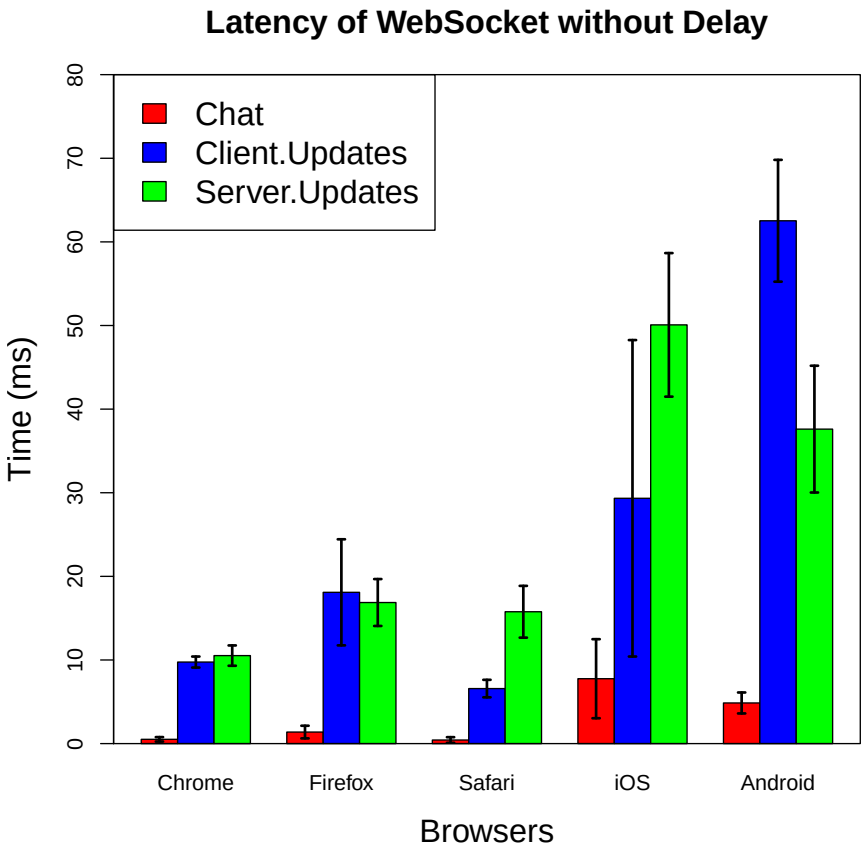


Figure 4.6: Application latency of the WebSocket protocol without any emulated delays across all tested browsers. Chrome, Firefox and Safari accessed the server directly, while the iOS and Android browsers through Wi-Fi.

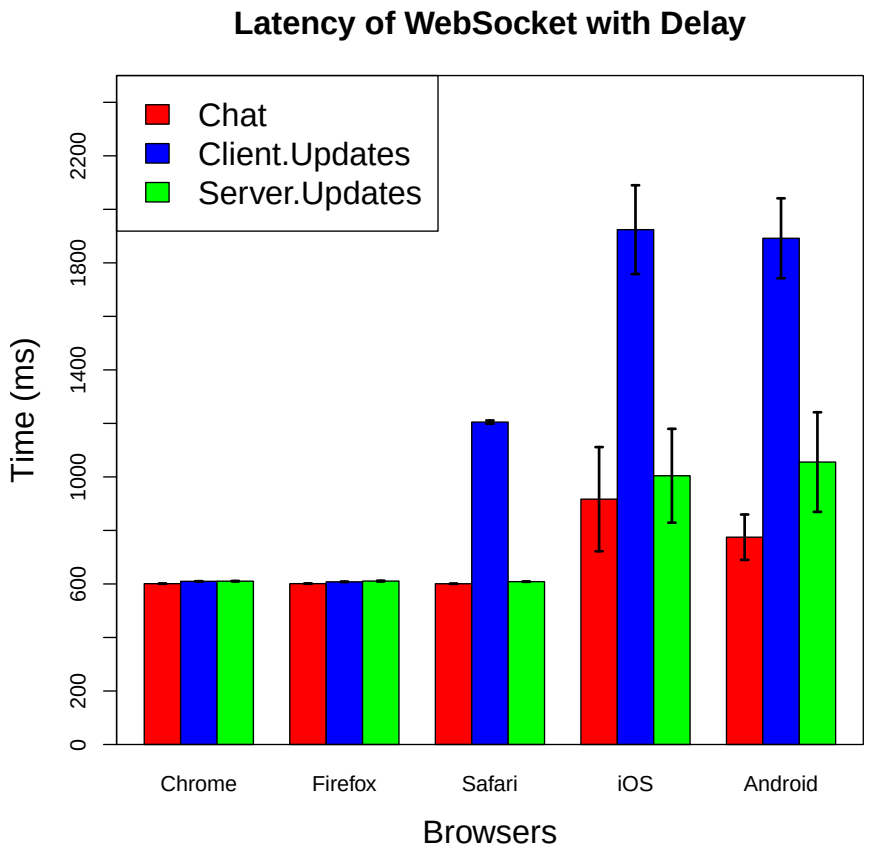


Figure 4.7: Application latency of the WebSocket protocol with the GEO satellite delay (RTT=600ms) emulated delays across all tested browsers. Chrome, Firefox and Safari accessed the server directly, while the iOS and Android browsers through Wi-Fi.

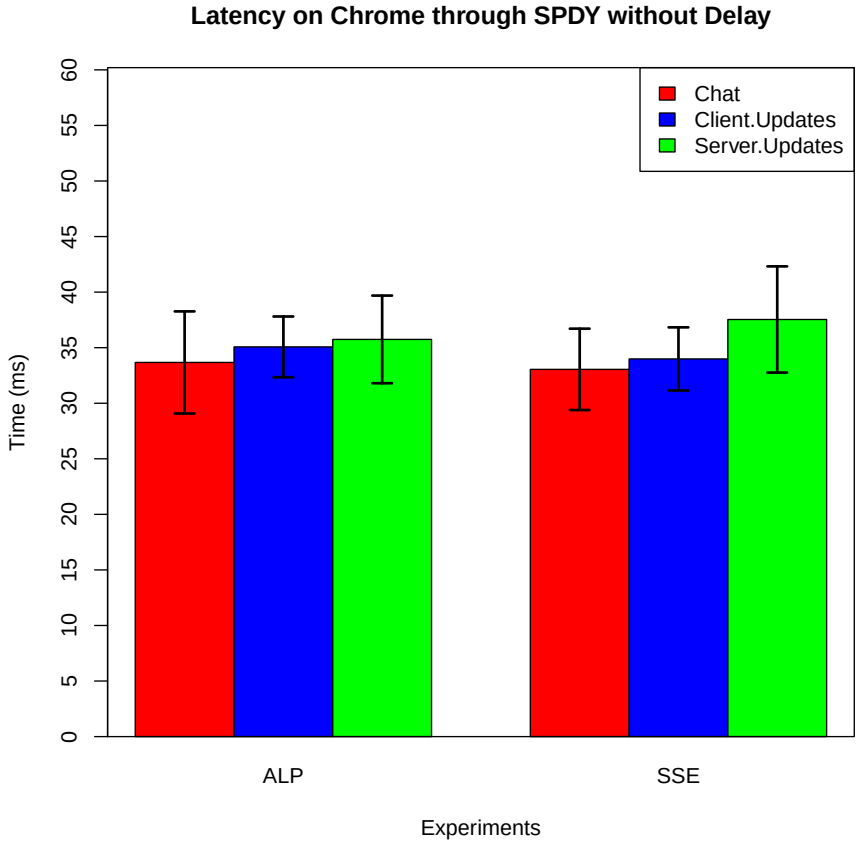


Figure 4.8: Application latency of Ajax Long-Polling and Server-Sent Events protocol without the GEO satellite delay on Google Chrome through SPDY. The WebSocket protocol is not compatible with SPDY, so it was not included.

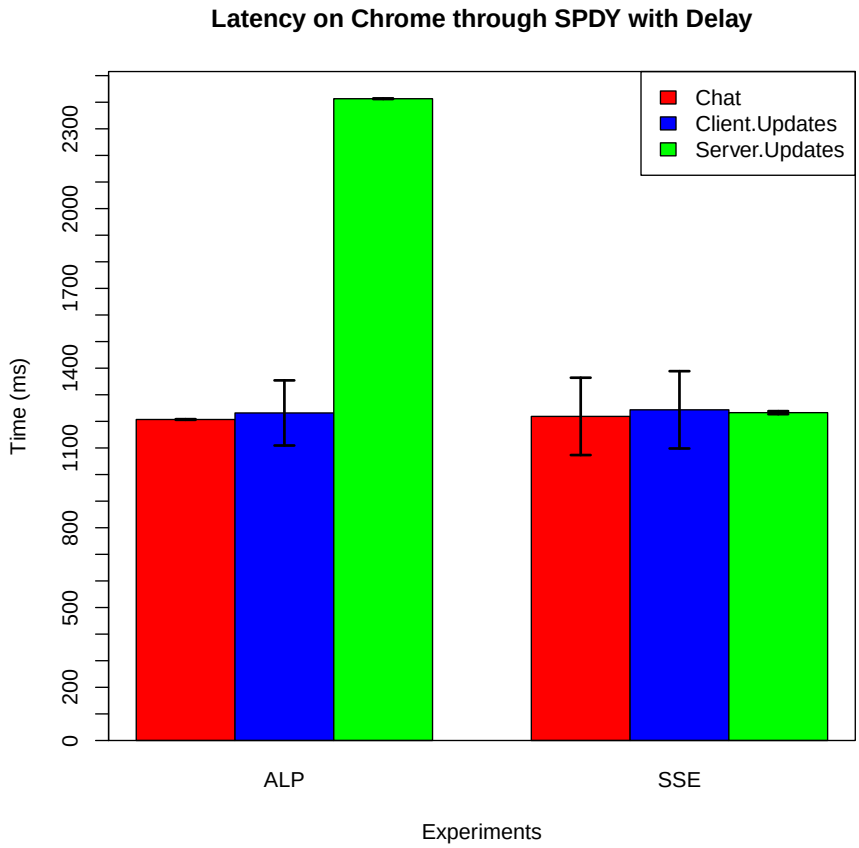


Figure 4.9: Application latency of Ajax Long-Polling and Server-Sent Events protocol with the GEO satellite delay (RTT=600ms) on Google Chrome through SPDY. The WebSocket protocol is not compatible with SPDY, so it was not included.

are observed only in the Server Update scenario, i.e. when 100 messages are transmitted from the server to the client. A similar qualitative behavior, although with a larger bias, is observed in the case of mobile platforms. The poor performance of SSE in Server Update scenario is investigated later in chapter, see 4.3.1. More specifically, in our case SSEs messages are a single line of text, prefixed by 'data: '. This causes an escaping procedure to be performed at every message. Unfortunately, this procedure slows down the production of messages at the server, whose transmission can be completed only one RTT later, in a second TCP segment.

A different situation is illustrated in Fig. 4.6 and Fig. 4.7, which show the latency of WebSocket respectively without and with a GEO satellite delay. WS outperforms all the other protocols completing the transaction in one RTT in almost all experiments on desktop platforms. Moreover, WS offers a communication channel with a more predictable latency, as the variance is negligible in desktop platforms and significantly reduced in mobile platforms.

Finally Fig. 4.8 and Fig. 4.9 show the latency achieved with Google Chrome desktop browser when ALP and SSE use SPDY/3 for transport, respectively without and with satellite delay . We can see that both application-layer protocols achieve worse performance with SPDY than when WS works in conjunction with HTTP/1.1. A possible explanation, which however needs further investigation to be confirmed, may be that SPDY was optimized for typical Internet Web browsing and is not yet able to efficiently cross-interact with real-time protocols.

The results of our experiments show that WebSocket (WS) introduces the least latency for Real-Time Web applications. However, WS support

is still not widespread amongst Web servers and Web proxies so the use of the other techniques may be required. Real-Time Web applications need to be aware of the latency of the protocol they are using for communications. We suggest that Real-Time Web applications estimate application latency by adding a "heartbeat transaction" that measure the time taken to send and receive a message to the server. After estimating the application latency, the application can then decide if it can provide a sufficient quality of service. Otherwise, the application can either switch to an offline operation mode by using a store-and-forward technique, or in the worst-case halt the application.

4.3.1 Enhancing Server-Sent Events

We argue Server-Sent Events can deliver a much increased performance, as it leverages two HTTP connections to implement a full duplex channel. In particular, the client sends updates through a standard AJAX call. The client initiates receiving data by issuing a HTTP GET request, and then the server appends the 'events' to the response body, using the SSE data format. As the updates can be sent using the HTTP keep-alive header that allows a client to reuse the same HTTP connection for multiple requests, the clients uses only two HTTP connections. Those connections have usually fairly wide TCP transmission windows compared to the data being transmitted. In order to understand why SSE underperforms, it is important to study the behaviour at the TCP level. In the server updates scenario, the 100-messages are divided across two TCP exchanges, one of 132 bytes and one with the remaining, thus requiring at least two RTT. In the chat and client updates scenario, only the 132-bytes long one is present, thus requiring only one RTT. This is

caused by Nagle's Algorithm, which combines a number of small outgoing messages to send them all at once. Specifically, as long as there is a sent packet for which the sender has received no acknowledgment, the sender should keep buffering its output until it has a full packet's worth of output, so that output can be sent all at once. Thus, disabling Nagle's algorithm improves SSE latency. However, this is normally not disabled for HTTP connections.
Fig 4.10, Fig 4.11, and Fig 4.12 show our results without Nagle's Algorithm. Both ALP and SSE showed better latency, and SSE has slightly worse latency compared to WebSocket in all scenarios. However, disabling Nagle's Algorithm on the server increased the variance for ALP and SSE techniques, especially in Firefox and Safari.

4.4 Conclusions and Guidelines

The Real-Time Web protocols offer a new way of communicating and cooperating across the globe. Thus, the satellite community should support these technologies in GEO satellite networks. In this chapter we evaluated the performance, in terms of latency, of most promising protocols for real-time Web applications using a range of scenarios. Our previous experiments show that mobile Web applications suffer longer latencies over a GEO satellite link. However, we showed that these longer latencies are not due to a software issue with the mobile browsers, but they are related to radio interferences or WiFi congestion. A possible mitigation to this problem can be the use of TCP splitting techniques. Finally, we proved that WebSocket offers the best latency over a GEO satellite link, as it avoids protocol-specific delays in the data flow. When WebSocket

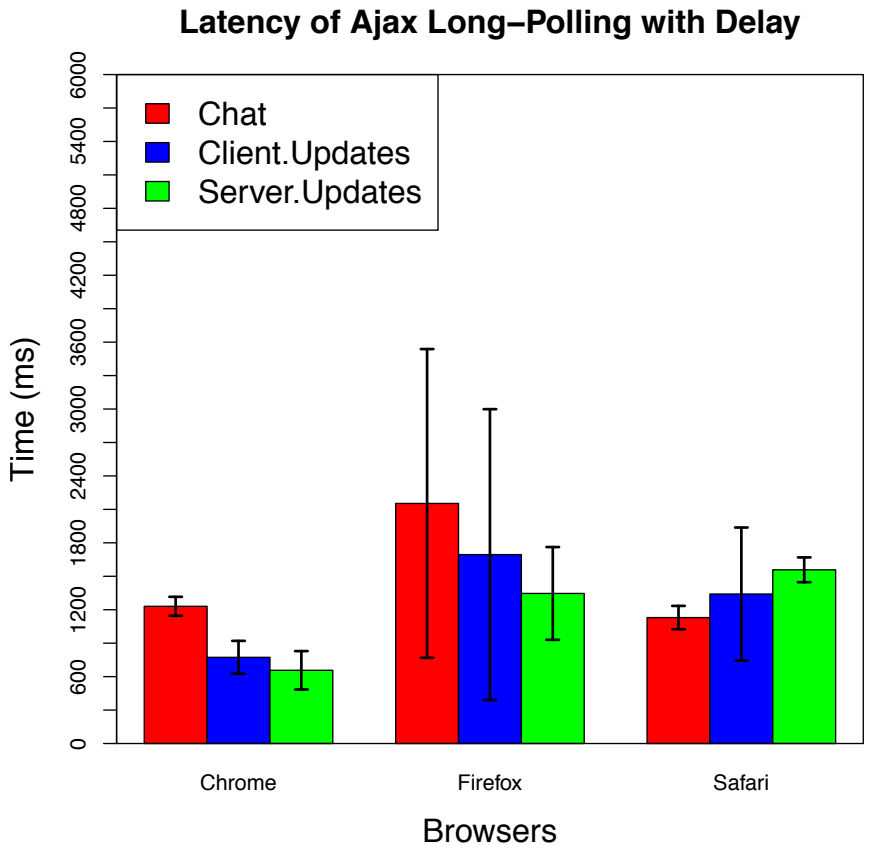


Figure 4.10: Application latency of the Ajax Long-Polling protocol with the Nagle's algorithm disabled. with the GEO satellite delay (RTT=600ms) emulated delays across all tested browsers.

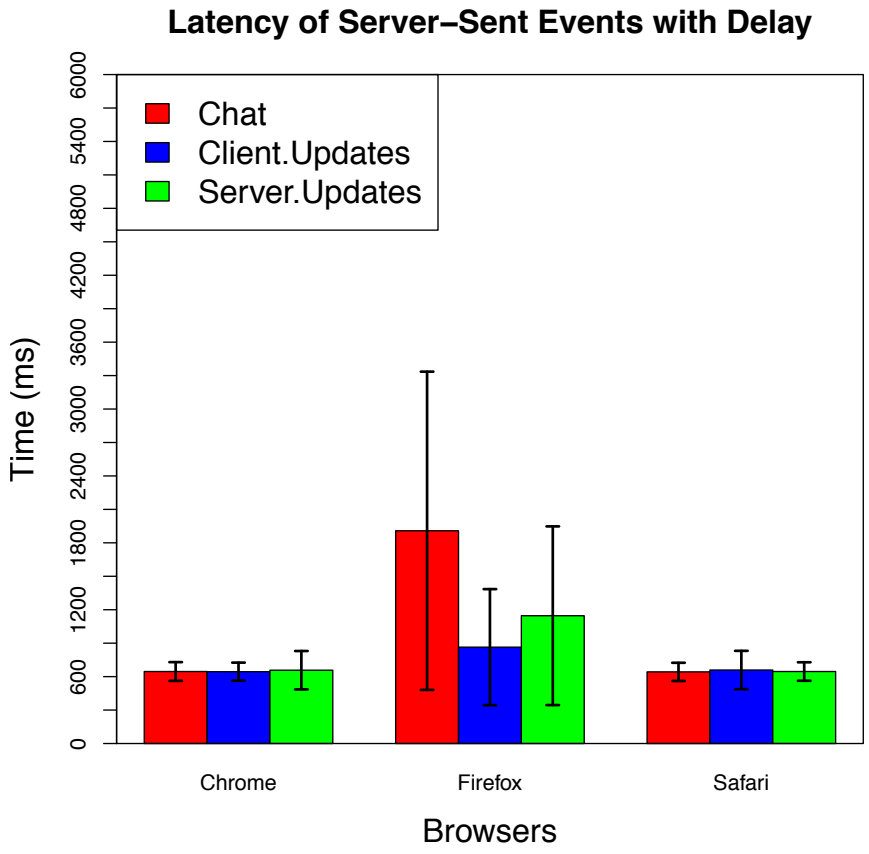


Figure 4.11: Application latency of the Server-Sent Events protocol with the Nagle's algorithm disabled. with the GEO satellite delay (RTT=600ms) emulated delays across all tested browsers.

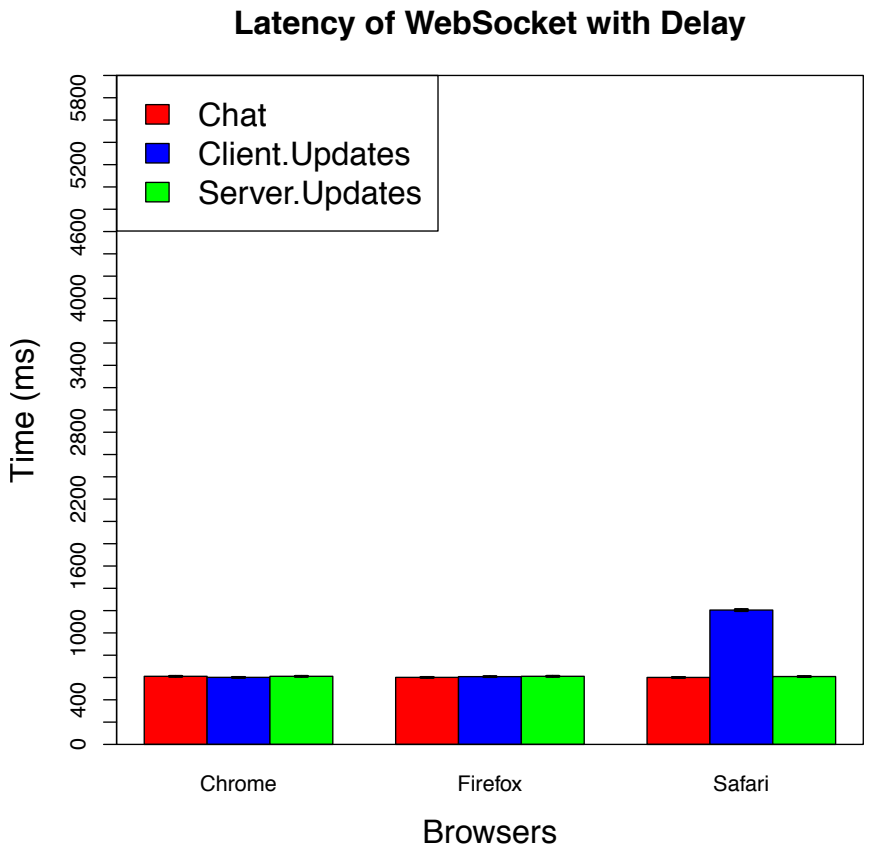


Figure 4.12: Application latency of the WebSocket protocol with the Nagle's algorithm disabled. with the GEO satellite delay (RTT=600ms) emulated delays across all tested browsers.

is not available, Server-Sent Events is a workable replacement if Nagle's Algorithm is disabled. Both protocols achieve much better performance than the legacy methods based on polling, e.g. Ajax Long Polling, especially when most of the data comes from the server. If only Ajax Long Polling is supported, then the best recommendation is to switch to an offline mode and synchronize the data later.

Chapter 5

Privacy Preservation Algorithms and Data Structures

The Internet of Things is forecasted to reach 50 billions of interconnected devices by 2020 [2]: in this network, the need to secure communication between things and humans, with a billion devices IoT, is one of the biggest challenges we will have to cope with, to avoid a massive privacy issue. In this regard, the European Union is investigating how to regulate the future Internet of Things [80].

The state-of-art in IoT protocols does not address the privacy needs of our society [81]: protocols can only provide security for the communication between two parties by leveraging strong cryptographic algorithms. However, the main privacy concern is not related to link attacks, but to normal operations of cloud and mobile systems. These systems must continuosly answer one specific question about every piece of data they receive: who can access it? In order to answer that question, IoT applica-

tions must agree on what data format to use and its meaning. However, that answer need to given in a timely manner to support both the high number of incoming data points and the increasingly-complex user interface patterns. In order to be simple, technology must be fast.

After discussing the possible data formats solutions for the Internet of Things, in this chapter we propose a new approach for fine-grained data access control. Our approach allows users to specify privacy preferences and enforces them when data is accessed. This approach can even be is co-located with the data and executed entirely on the user's mobile device. No external server support is needed, giving the user full control over his/her data at any time without trusting an external party. Our approach is based on RDF and SPARQL, modelling privacy preferences with RDF and checking them with SPARQL queries. This allows us to reuse the full power of RDF/SPARQL support in the existing RDF store on the mobile device without the need to add an additional reasoner or specific parser to process language specific rules. At the same time we retain the expressiveness of access control policies.

Our approach does not assume any special support from the RDF store and can be used on top of any RDF store that offers support for SPARQL. To filter RDF triples we introduce a novel two stage approach that combines (1) an initial efficient query analysis stage that extracts the necessary metadata about the query and the (2) filtering phase that filters the result set without having to access the store for additional metadata (about the query). Our evaluation shows that this improved filtering algorithm results in a 10 times increase in system performance compared to our previous approach.

This chapter is structured as follows: in Section 5.1 we analyze the pos-

sible choices for an IoT application at the data level, completing the interoperability work started on Chapter 2. Then, Section 5.2 gives a short overview on our target scenario and our assumptions. Based on this, Section 5.3 presents the basic filtering algorithm based on previous work. Then, we introduce our new improved filtering algorithm in Section 5.4. Section 5.5 presents evaluation results. Finally, Section 5.6 gives an overview of related work before we wrap up the chapter with future work and a conclusion in Section 5.7.

5.1 Data Interoperability

Data Interoperability is the other problem we have to solve when developing an IoT application with multiple devices. Moreover, the data we want to send evolves over time and we need a solution that allows to interconnect today's things with tomorrow's. Supporting data interoperability in a communication between two or more parties requires two steps. First, all the parties needs to agree on a specific data format to use. Secondly, all parties needs to agree on the semantics of the data stored in that data format.

The semantics problem is very important, but it is often neglected in IoT applications. The most common example is avoiding to transmit the unit of measure with a sensed sample or a command, e.g. 'advance by 1' is a command that has no meaning and can be understood as 'advance by 1 meter' or 'advance by 1 parsec'.

This section is organized as follows: firstly, we discuss the various option for data formats available in IoT products; then, we discuss the semantics of the data, and how to ensure our application will be interoperable in

Data Format	Encoding	Tipology
XML	textual	Tree-Based, values as strings
EXI	binary	Tree-Based, different data types
N3/Turtle	textual	Graph-Based, values as strings
JSON	textual	Object-Oriented, values as strings
Message Pack	binary	Object-Oriented, different data types
Protocol Buffers	binary	Object-Oriented, different data types
Bysant	binary	Object-Oriented, different data types
BSON	binary	Object-Oriented, different data types

Table 5.1: Data Formats for IoT Applications

the future; thirdly, we propose a recommendation that allows automatic conversion between the different data formats.

5.1.1 A Plethora of Data Formats

A data format dictates how data is represented on the network or disk. A data format can be either textual or binary: textual is usually human-friendly, while binary is harder to debug if an error happens. However, data encoded in a binary format is usually much smaller and requires less battery to be sent.
A data format forces a particular data representation and support some data types. The most common data representations are:

- Graph-based, in which the data is modeled as nodes that can have different relations between each other.

- Tree-based, in which the data is modeled as nodes that might have one or more children.
- Object-Oriented, in which the data is modeled with the common datatypes found in programming languages.

Table 5.1 shows the most common data formats. XML is extremely common, albeit very verbose, and it is usually used in big enterprises; XML usually encodes all data types as strings. EXI [49] is a binary representation of XML that can encode values depending on their type, e.g. Strings, Integer, Float.
Turtle [82] is a W3C recommendation for representing graph data based on the RDF model [83]. RDF is also used as the basis of the Linked Data initiative [84], allowing to reuse different data sets from different parties. The benefit of using RDF inside IoT applications is that we can identify objects globally, e.g. unit of measures.
JSON is the data format language of the web, and it is being applied in lieu of XML everywhere, because it is more developer-friendly as Maps, Lists, and all other basic datatypes can be serialized directly to JSON. MsgPack [85], Bison [86], and Bysant [87] are binarization of JSON, as they have a one-to-one conversion which maps to the same datatypes and they are all schemaless. Protocol Buffers [88] is a schema-driven data format that can be easily used in modern Object-Oriented languages, but it requires a schema compilation step.

5.1.2 Data semantics

To better define the semantics of IoT data, a number of ontologies have been developed on multiple layers of abstraction [89]: (1) sensor-centric

ontologies like the Ontonym sensor ontology [90], the Sensor Data Ontology [91], and OntoSensor [92]; (2) observation-centric ontologies like the Semantic Sensor Network Ontology [93], the Sensei Observation and Measurement Ontology [94], and stimuli-centered ontologies [95], as well as (3) context-centric ontologies like COMANTO [96] and SOUPA [97]. Clearly it is impossible to specify a single ontology that defines the semantics of all possible data items as they are in many cases application (domain) specific. This has lead to the development of rather abstract and complex ontologies that try to fit all possible cases by providing a conceptual framework only, omitting concrete instances like specific sensor models, etc. Such ontologies try to impose an overarching structure onto IoT systems and their data, e.g. specifying abstract metadata classes for stimuli, observations, measurements, sensors and features of interest. In practice however it is not clear how such complex ontologies will actually help developers. It is often very difficult to model even simple things like a temperature reading with an existing ontology since this requires to understand its abstract concepts (will this be a measurement or an observation) and to define concrete aspects like the unit of the reading. To do so, other ontologies must be used, which are in turn very complicated, trying to model all possible units in a structured way. Instead, multiple simple ontologies are needed that restrict themselves to very small areas, specifying them in full detail such that developers can quickly understand and use them.

5.1.3 A common data format

The Object-Oriented typology of data formats is the most widespread across developers, and there are several proposals to specify a binary ver-

sion of it. Moreover, it is possible to represent any tree-like data structures like XML in a Object-Oriented typology. The JSON-LD [98] specification allows to store RDF data on top of JSON. JSON-LD provides also a way of transforming the data from one representation to another [99]. Thus, selecting an Object-Oriented typology allows to convert into the others, without losing data or the semantics associated with the data.

5.2 Access Control for RDF Stores

In this work, we are focusing on how access to personal user data. To clarify our target scenario, consider two friends Alice and Bob who want to exchange personal data with their smartphone devices. Each device by default, denies access unless otherwise instructed by its user. Alice uses her smartphone, contacts Bob's smartphone and asks for his location. Bob receives a notification on his smartphone that Alice has requested to access his location[1]. Bob grants Alice access and this privacy preference is stored in his smartphone. Alice can now retrieve and view Bob's location on her smartphone. Other data is still not accessible. Next time Alice requests to view Bob's location, if the request matches Bob's stored privacy preference, then she is automatically granted (or denied) access. Otherwise, Bob is notified about Alice's new request and decides whether to grant her access or not.

To realise this example we propose an access control system for RDF stores on mobile devices. By storing the data directly on the users' mobile devices, users can have full control over their data without trusting any

[1]We consider location, but modern personal IoT application can track blood pressure, activity, and other extremely sensitive personal data

external server or provider. However, the access control algorithms must be executed on the mobile device, too and thus they must be very efficient to respond in a timely fashion and not waste battery life.
Our approach is not limited to mobile devices, but it can also serve as a basis for a massive privacy system that is powered by cloud computing service providers. We primarily consider mobile devices as they are easier to measure than cloud-sized systems with hundred of thousand of user. However, the algorithms presented in this work are totally mobile-independent.
Our approach models access control policies for RDF data using the Privacy Preference Ontology (PPO). PPO is non-domain specific and can model privacy preferences for any RDF scenario. In this section, we provide an overview of PPO and we explain how we model privacy preferences using it. Subsequently, we describe how the Privacy Preference Manager (PPM) enforces such privacy preferences by filtering out RDF data based on them. The PPM is datastore independent and therefore can be easily customisable to provide fine-grained access control to any datastore.

5.2.1 Privacy Preference Ontology (PPO)

PPO[2] [100,101] is a *light-weight* Attribute-based Access Control (ABAC) vocabulary that allows users to describe fine-grained privacy preferences for restricting or granting access to non-domain specific Linked Data elements, such as Social Semantic Data. Considering that PPO is described in RDF(S), it does not require a specific parser or reasoner but it retains the expressivity of fine-grained access control policies similar to rule-

[2]PPO – `http://vocab.deri.ie/ppo#`

based approaches. Among other use-cases, PPO can be used to restrict part of FOAF[3] profile records to users that have specific attributes. It provides a machine-readable way to define settings such as "Provide my location only to my family" or "Grant read access to my activity only to Alice".

As PPO deals with RDF(S)/OWL data, a privacy preference defines: (1) the resource, statement, named graph, dataset or context it must grant or restrict access to; (2) the conditions refining what to grant or restrict (for example defining which instance of a class as subject or object to grant); (3) the access control privileges (including `Create`, `Read`, `Write`, `Update`, `Delete` and `Append`); and (4) an `AccessSpace`, defined by either an agent or a SPARQL query that specifies a graph pattern that must be satisfied by the requesting user.

Example

Figure 5.1 illustrates Bob's privacy preference that restricts his location only to Alice. The location is modelled as an instance of type `SpatialThing`[4] which includes longitude and latitude. Hence the privacy preference is applied to any resource of this type – in our case, Bob's location. In this example Alice is granted the `read` access to Bob's location.

5.2.2 Privacy Preference Manager (PPM)

The PPM [102, 103] is an access control manager that allows users to create privacy preferences for RDF data. The manager also filters the

[3]Friend-of-a-Friend (FOAF) – `http://www.foaf-project.org`

[4]WGS84 – `http://www.w3.org/2003/01/geo/wgs84_pos#`

```
PREFIX ppo:      <http://vocab.deri.ie/ppo#> .
PREFIX wgs84: <http://www.w3.org/2003/01/geo/wgs84_pos#>
<http://bob.com/PrivacyPref#1> a ppo:PrivacyPreference;
ppo:hasCondition [
        ppo:classAsSubject wgs84:SpatialThing];
ppo:assignAccess acl:Read;
ppo:hasAccessSpace [
        ppo:hasAccessAgent <http://alice.com/me>].
[...]
```

Figure 5.1: Bob's privacy preference to grant Alice his location

requested data by returning only a *subset* of the requested data containing only those triples that are granted access as specified by the privacy preferences. The PPM was developed as a Web application – either as a centralised Web application or in a federated Web environment. The privacy preferences are stored separately from the data and can only be accessed by the PPM.

Although the PPM is suited for Web environments, it is not originally designed for operating on mobile devices due to their limited resources – such as processing power, memory resources and battery life. To port the PPM to mobile devices we modified the enforcing algorithm substantially to reduce the number of querying operations needed for filtering. In addition we designed a new filtering algorithm that extends our previous one to further reduce the number of queries. In the subsequent sections we first explain the original filtering algorithm and outline the parts which are resource expensive. We then provide our extended algorithm and

evaluate both of them.

5.3 PPM Access Control Filtering Algorithm (PPF-1)

The PPM access control filtering algorithm (called PPF-1 in this chapter) consists of (1) a matching part which maps the triples in the requested result set to the specific privacy preferences that apply to the triple; and (2) a filtering part that filters the result set by checking which triples a requester is granted access. This algorithm was not published in our previous work and therefore in this section we provide a detailed overview. Initially, PPF-1 expects a list of requested triples together with the named graph they reside in. Moreover, the set of privacy preferences related to the data in the store is also passed to the algorithm. With these, PPF-1 first matches the triples to their corresponding privacy preferences; then, it checks what the requester can access and grants the requester a filtered result set. The following sections describe the different parts of PPF-1 in more detail: Section 5.3.1 describes the matching part and Section 5.3.2 describes the filtering part.

5.3.1 Privacy Preferences and Triples Matching

Algorithm 1 illustrates the matching between triples and privacy preferences. This part iterates through every triple in the result set and for every triple it checks all the privacy preferences to match which ones apply to the triple. The algorithm checks whether each privacy preference applies to: (1) the named graph in which the triple resides; (2) a

Data: *resultSet* and *privacyPreferencesList*

Result: (1) *protectedTriplesList*; (2) *unprotectedTriplesList*; (3) *accessAgentsList*; and (4) *accessPrivilegesList*.

```
List<PrivacyPreference> pList ← privacyPreferencesList;
List<Triple> rs ← resultSet;
Triple t ← new Triple();
PrivacyPreference p ← new PrivacyPreference();
forall the t ∈ rs do
    forall the p ∈ pList do
        if p.Match(t) then
            pURI ← p.getPrivacyPreferenceURI();
            aURI ← p.getAgentURI();
            privilege ← getAccessPrivilege();
            protectedTriplesList.add(t, pURI);
            accessAgentsList.add(aURI, pURI);
            accessPrivilegesList.add(privilege, pURI);
        else
            unprotectedTriplesList.add(t);
        end
    end
end
```

Algorithm 1: Privacy Preferences and Triples Matching

Data: *subject URI* or *object URI* of the triple and *restricted class*
Result: *boolean isInstance* – i.e. whether the subject or object is an instance of the class

```
query ← "SELECT ?o WHERE <subject URI ∨ object URI of
restricted triple> rdf:type ?o";
result ← executeQuery(query);
if (result ≠ restrictedClass) then
    remote ← getEndpoint(subject ∨ object);
    remoteResult ← remote.executeQuery(query);
    if remoteResult ≠ restrictedClass then
        isInstance ← false;
    else
        isInstance ← true;
    end
else
    isInstance ← true;
end
```

Algorithm 2: Class Matching

resource in the triple; and (3) a rectified statement – i.e. the triple's subject, predicate and object.

The algorithm checks whether each privacy preference has a condition that specifies: (1) the resource must be the subject of the triple; (2) the resource must be the object of the triple; (3) the subject of the triple must be an instance of a certain class; (4) the object of the triple must be an instance of a certain class; (5) contains a particular predicate; and (6) contains a particular literal.

For most of these checks, the values in both the requested triples and in the privacy preferences are tested to check whether they are both the same. However, for testing whether a subject or object of the triple are instances of a particular class, the algorithm queries the store each time a privacy preference (for each triple) is tested. This part is explained in Algorithm 2.

Algorithm 2 checks whether the subject or object of a requested triple are instances of a class specified in a privacy preference. This algorithm is called by algorithm 1 that passes the subject or object of the triple and the restricted class specified in the privacy preferences as parameters. The algorithm constructs a query that gets the class type of the subject or object. If the class type matches with the restricted class then the algorithm returns true to Algorithm 1. Otherwise it returns false. If the result of the query does not contain any result (i.e. `result = null`), then the algorithm fetches the endpoint URI of the datastore in which the class types for the subject or object are specified. The endpoint URIs are mapped to the subjects and objects. Once the class type is retrieved, the algorithm returns to Algorithm 1 whether they match (true) or not (false).

```
Data: protectedTriplesList
Result: (1) accessTriplesList(triple, privilege)
        (2) noAccessTriplesList(triple)
Iterator<ProtectedTriple> pIterator =
protectedTriplesList.Iterator();
while pIterator.hasNext() do
    pt ← pIterator.next();
    forall the agent ∈ accessAgentsList do
        if pt.privacyPreferenceURI = agent.privacyPreferenceURI
        then
            if ¬(pt.Triple ∈ accessTriplesList) then
                privilege ← accessPrivilegesList.Privilege;
                accessTriplesList.add(pt.Triple, privilege);
            end
        else
            noAccessTriplesList.add(pt.Triple);
        end
    end
end
```

Algorithm 3: Privacy Preferences Filtering

If any of the `p.Match(t)` conditions in Algorithm 1 are true, then the triple and the privacy preference's URI are added to the `protectedTriplesList`. Moreover the access privileges of each matched privacy preferences are added to the `accessPrivilegesList` together with the privacy preference URI – in order to map the triples to the access privileges by using the privacy preference URI as the lookup identifier. Similarly, the access agent in each matched privacy preference are added to the `accessAgentsList` together with the privacy preference URI. Once all the triples are iterated, the filtering part filters the protected triples as explained below.

5.3.2 Privacy Preferences Filtering

Algorithm 3 filters the triples to send back only the triples which the agent has access to. The algorithm checks that for each triple in the `protectedTriplesList`, the agent has been granted access by matching the privacy preference URI bound to the triple with the URI bound to the agent. If these match, then the triple is added to the `accessTriplesList`. If the privacy preference URI does not match to any of the URIs bound to the agent, then the triple is added to the `noAccessTriplesList`. Once completed, the filtering algorithm sends back the `accessTriplesList` that represents the filtered result set.

5.4 Extended Access Control Filtering Algorithm (PPF-2)

PPF-1 has a major performance bottleneck in the privacy preference matching phase: for each restricted triple and for every privacy preference PPF-1 executes a query on the RDF store to test whether the subject or object is of a particular class type. For instance if there are 100 requested triples and 100 privacy preferences that test different types of classes, then PPF-1 will initiate 10,000 queries – assuming that each privacy preference tests only one class type. This may result in a large overhead since executing a query can be expensive – specifically on mobile devices with restricted resources. To increase efficiency, the number of necessary store accesses for identifying the class of a resource must be reduced without losing PPF-1's fine-grained control over data access.

In this section we introduce an extended filtering algorithm (called PPF-2) that fulfils these requirements. The main idea of PPF-2 is to identify the class of a resource by analysing both the requested query and the ontologies used by the data. To reduce the effort of analysing the used ontologies, we perform an ahead-of-time indexing phase for the ontologies at the system start time. This index is later used to identify the given classes. With this ahead-of-time indexing in place, the actual filtering process becomes a two stage algorithm, as follows:

1. analysis of the query to derive the resources' classes (Stage 1);
2. filtering of the triples (Stage 2), using the knowledge derived in Stage 1.

In the following we describe how we realise Step 1. Stage 2 is similar to

the filtering done in PPF-1 and thus not explained again.

5.4.1 Knowledge Extraction from the Ontology and Query

Our solution is based on a query analysis step that allows to identify the classes of each resource based on the attributes that are used in the query. The query analyser parses the SPARQL query and for each resource it extracts *inbound* and *outbound* properties. Inbound properties are extracted from the triples in which the resource is the object. Outbound properties are extracted from the triples in which the resource is the subject. Based on these properties it is possible to identify the classes of a resource by looking at the ontologies data. Our approach uses a closed-world assumption, i.e. we assume that the filtering algorithm knows every ontology on which a privacy preference can be defined. This assumption is valid because: if an ontology is unknown when the privacy preference is defined, then the PPM can retrieve it before any actual query is run. The RDF Schema [5] standard defines two type of relationship for properties: `rdfs:domain` and `rdfs:range`. The first is used to state that any resource that has a given property is an instance of a class, while the second is used to state that the values of a property are instances of a class. Thus, both of them can be used to derive the actual class(es) of a resource.

[5]RDF Schema – `http://www.w3.org/TR/rdf-schema/`

```
PREFIX rdf: http://www.w3.org/2000/01/rdf-schema
SELECT ?class ?property
WHERE {
  {
    ?property <rdf:#domain> ?class
  }
  UNION
  {
        ?property <rdf:#domain> ?parent .
        ?class <rdf:#subClassOf>+ ?parent
  }
};
```

Figure 5.2: The SPARQL 1.1 query to build the index on the domain relationship.

5.4.2 Defining an Index to derive Classes from Properties

As mentioned before, it is possible to identify the class of a resource by looking at the query and leveraging the ontology. Similarly to accessing the store, querying the ontologies is a slow process. This can be improved by indexing the ontologies (once) *before* any actual query is run. Thus, it is possible to make the identification of a resource's class a memory-only operation.

Figure 5.2 shows a query that – when executed on a RDF store containing all the ontologies – extracts all the given properties of a specific class.

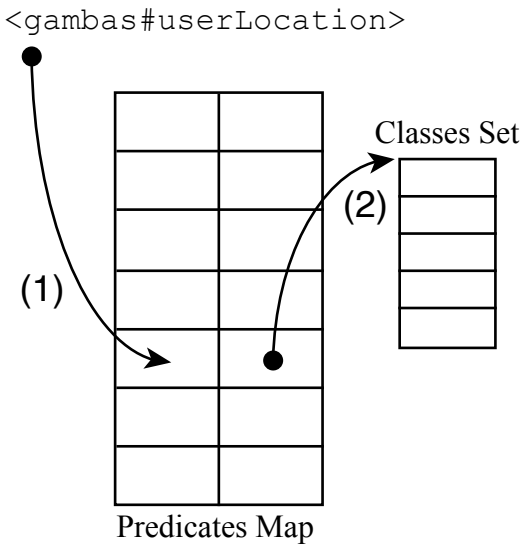


Figure 5.3: The index data structure used by the class derivation algorithm. The map is accessed with the predicate (1) and then the set is processed (2).

```
PREFIX gambas: http://www.gambas-ict.eu/ont/
PREFIX wgs84: http://www.w3.org/2003/01/geo/wgs84_pos#
SELECT ?lat ?long ?noise
WHERE {?user <gambas:userLocation> ?location .
  ?location <wgs84:lat> ?lat .
  ?location <wgs84:long> ?long .
  ?location <gambas:noiseLevel> ?noise}
```

Figure 5.4: A SPARQL query example where the resources' class can be uniquely determined by the query analysis step.

Moreover, it uses the "new" path syntax introduced in SPARQL 1.1 to gather all the properties of its super classes. A similar query is then used to extrapolate the classes from the `rdfs:range` relationship. With this information two indexes are built, one for using the `rdfs:domain` and one for using the `rdfs:range` relationships. To guarantee $O(logN)$ time cost (with N the number of different predicates) to access fast access to the information in an index, we use a combination of Red-Black tree-based map and set implementations.

Figure 5.3 shows an example of how the `rdfs:domain` index is used. Given a resource linked through a predicate `userLocation`; we use the predicate as a key into the predicates map. The accompanying value in the map points to a set of classes, which we add to a result set. This procedure is then repeated for all predicates of the given resource. Then, all the resulting sets are intersected The resulting intersected set contains all the classes that the resource can be an instance of. This process is repeated for each index and the results are intersected.

Example

Figure 5.4 shows a SPARQL query usable to extract the location (given as latitude and longitude) of a given user and the noise level at this location. The `?user` is modelled as a `gambas:User`, a subclass of `foaf:Agent`. The `?location` is a `gambas:Place`, a subclass of `dol:Location` [6], which has an attached `wgs84:lat` (latitude) and `wgs84:long` (longitude). In order to derive the classes of the variables in the query of Figure 5.4, the algorithm proceeds as follows for the `?user` resource:

1. extract the `<gambas:userLocation>` property;
2. access the index on `rdfs:domain` using the property as key;
3. access the linked classes set, which contains only the `gambas:User` class.

A similar approach can be applied to the `?location` resource. In the following section we will show a comparison of the performances of this modification versus the base case.

5.5 Evaluation

In order to evaluate the performance gain achieved by our extended filtering algorithm, we conducted a number of experiments on a Google Nexus 7 device running Android 4.2.2. Our system is implemented in Java. We compared two configurations with a PPM running on top of an RDF On the Go data store [104]. In the first configuration the PPM

[6] DOLCE – `http://ontologydesignpatterns.org/wiki/Ontology:DOLCE\%2BDnS\_Ultralite`

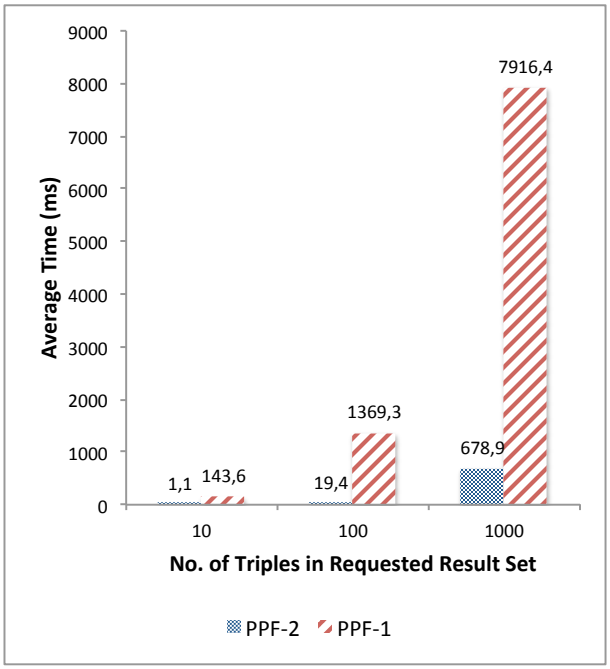


Figure 5.5: Performance with varying size of result set.

is using our previous filtering algorithm PPF-1. In the second one, the PPM is using our new filtering algorithm PPF-2.

5.5.1 Evaluation Setup and Architecture

The evaluation dataset was composed of 15000 triples, containing data about seven real-world user profiles. Using this dataset we executed a sample query on a user's topic interests and filtered the intermediate results with both algorithms (PPF-1 and PPF-2). Since we are mainly interested in the overhead induced by access control instead of query execution, we measured the execution time for filtering, omitting the time needed to execute the sample query on the dataset. The latter time depends only on the underlying RDF store and thus is the same for

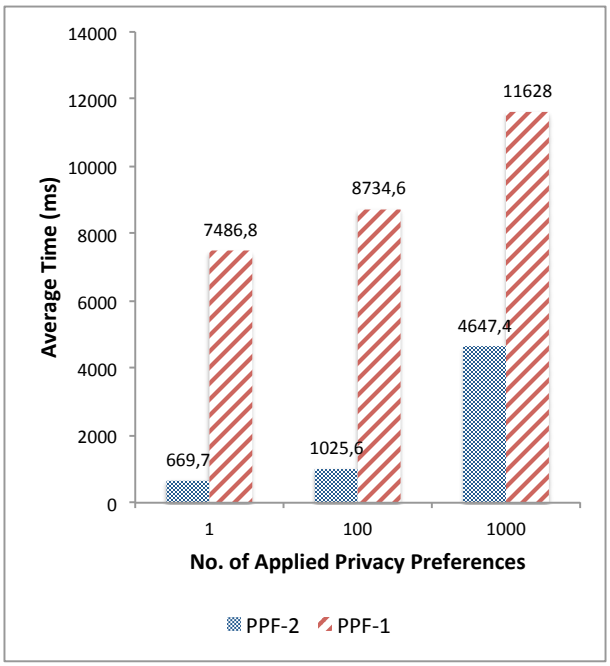


Figure 5.6: Performance with varying number of privacy preferences

both filtering algorithms. To characterise the filtering performance in scenarios with different complexity, we varied both the number of triples in the intermediate result and the number of checked privacy preferences. Each experiment was repeated ten times. We started measuring after an initial preheating phase consisting of ten filtering runs. This reduced the variance introduced by the Android Just-in-Time optimiser. Moreover, each experiment was executed independently in a separate Android App, with no other running App and with all synchronisation services disabled – further reducing variances.

5.5.2 Query types and datasets

Figure 5.5 shows the execution time for filtering an intermediate result set of varying size (10, 100, and 1000 triples) using a single privacy preference. As can be seen, PPF-2 clearly outperforms PPF-1 by at least a factor of 10, confirming the effectiveness of the predefined index technique (see Section 5.4). Even for an intermediate result set of 1000 triples (representing the result of a query matching a comparatively large number of the 15000 triples in the RDF store), PPF-2 requires only approximately 0.7s to check access and filter the result set. In comparison, PPF-1 requires nearly 8s, making it unsuitable for many scenarios, e.g. interactive systems. The time required for filtering a mid size intermediate result set of 100 triples is around 0.02s for PPF-2 (compared to approximately 1.4s for PPF-1). Filtering a small intermediate result set of only 10 triples is nearly not measurable with both algorithms.

Figure 5.6 shows the execution time for filtering an intermediate result set of fixed size (1000 triples) using a varying number of privacy preferences (1, 100, and 1000 preferences). Again, PPF-2 clearly outperforms PPF-1

for all measurement points, reducing the absolute time for filtering triples with 100 privacy preferences to around 1s, down from 8.7s. Interestingly, the results for filtering with one privacy preference are quite similar (0.7s for PPF-2, down from 7.5s) due to fixed (i.e. size-independent) execution efforts. For 1000 privacy preferences, PPF-2 can still outperform PPF-1 by a factor of approximately 2.5 but both algorithms may still be too slow to be used in time critical scenarios (with PPF-1 requiring around 11.6s and PPF-2 around 4.6s).

Note that the presented results are only valid for situations in which the original query contains knowledge that can be used for filtering optimisation. This may not always be the case. Therefore we also conducted experiments with an unbound query that requested all triples in the RDF store. This query contains no knowledge for PPF-2. In this case PPF-2 is reduced to PPF-1. It must access the store for each triple check and thus cannot perform better than PPF-1. This is confirmed by our measurements, since the results for PPF-1 and PPF-2 are the same in this case.

5.6 Related Work

Access control and privacy for RDF data is not a new topic. In this section we discuss related approaches and explain how our work differs from earlier work.

Access control privileges for RDF data can be modelled using the Web Access Control (WAC) vocabulary[7]. However, this vocabulary is designed to specify access control to entire RDF documents rather than to spe-

[7]WAC — `http://www.w3.org/ns/auth/acl`

cific data contained within the RDF document. *Privacy policies* can be modelled using the Platform for Privacy Preferences (P3P)[8]. It specifies a protocol that enables Web sites to share their privacy policies with Web users expressed in XML. P3P does not ensure that Web sites act according to their publicised policies and it does not enable end users to define their own privacy preferences. The authors in [105] propose a *privacy preference formal model* consisting of relationships between subjects and objects in Social Semantic Web applications. However, the proposed formal model does not provide fine-grain access control for RDF data. Similarly, the authors in [106] also propose an access control model for semantic networks. However, they do not cater for RDF data in mobile devices. `RelBac` [107] is a relational access control model that provides a formal model based on relationships amongst communities and resources. It is also not intended for RDF data stored in mobile devices.

The authors in [108] propose an *access control framework* for Social Networks by specifying privacy rules using the Semantic Web Rule Language (SWRL)[9]. However, this work does not support processing SWRL rules on mobile devices and requires a specific parser to process the SWRL syntax.

The authors in [109] compare 12 rule-based languages for enforcing access control. Most of them require defining a large amount of rules for defining access control policies. Moreover, these require specific reasoners and parsers; apart from a system to enforce them. Our system however is based on an RDF(S) vocabulary thus processable by RDF parsers without installing a specific parser. It is also light-weight and requires

[8]P3P — `http://www.w3.org/TR/P3P/`

[9]SWRL — `http://www.w3.org/Submission/SWRL/`

minimum amount of defining access control policies but keeping similar expressivity as rule-based approaches.

In [110] the authors propose a system whereby users can set access control to RDF documents. Our approach provides more fine-grained access control to the data rather than to the whole RDF document.

The authors in [111] present a role-based *access control model for RDF stores* called RAP that binds role permissions to RDF store actions, such as inserting a triple. This model does not support fine-grained access control for data stored in mobile devices. The authors in [112] also present an access control framework for RDF stores that consists of a pre-policy evaluation and query rewriting. The authors use Protune [113] for expressing the policies which requires a specific framework to process these policies.

Finally, the authors in [114] propose an access control vocabulary that is similar to our PPO and a manager similar to our PPM. However, their model applies only to named graphs, unlike our model which we apply to statements, resources and classes. Although they provide support for mobile devices, the access control policies are sent to a central server and processed on this server. Our approach supports access control filtering directly on mobile devices.

5.7 Conclusion and Future Work

Access to personal data must be controlled tightly and efficiently. In this chapter we presented our approach for fine-grained access control for RDF data on mobile devices. It allows users to fully control access to their data directly on their mobile devices, increasing their trust in

the system. This will increase their willingness to share such data with others in a privacy preserving manner and independently of any external provider. However, this system can be extended to work with a trusted external provider, as all algorithms and data structure presented do not make any mobile assumptions.

As we have shown, Linked Data technology like RDF and SPARQL can be used – even on mobile devices – to realise access control for RDF data. By using RDF to model our privacy preferences (with the same expressivity as rule-based approaches) and a SPARQL engine to check them, no special rule language and reasoner components are necessary. Instead, the store managing the user data can be used to realise the access control on this data. Our experiments show that to be efficient such a system should combine multiple techniques, e.g. pre-indexing, query analysis as well as result filtering. This way we could improve performance by a factor of ten in many cases.

We presented two filtering algorithms that can be used to enforce privacy preferences: PPF-1 and PPF-2. We analysed PPF-1 and showed that it is ill-suited for mobile applications, due to the serious overhead introduced by the filtering process. The same overhead can cause similar problems with high-throughput cloud applications. In order to reduce the overhead, we proposed PPF-2, which uses a novel approach for extracting knowledge from the requestor's query. We evaluated both algorithms and showed that our optimisation improves the filtering process by a factor of ten.

Our work can be extended in several directions. Firstly, an evaluation of the impact of the proposed index on a combination of different types of privacy preferences is needed. Secondly, access space queries remain

problematic, as they need to be tested on the store. It should be possible to address this in a similar manner as PPF-2 by analysing and building indexes for access space queries prior to executing the filtering algorithm.

Conclusion

At the beginning of this 3-years program on the Internet of Things, this new field appeared extremely fragmented and not coherent: different approaches were presented and they seemed in complete contradiction, as they were solving very different needs. In fact, most of the development in the IoT field happens in non-compatible silos, but we believed that a common interaction model should exists. We thought that, if we had found that model, we could have been able to solve the interoperability problem between silos, thus building *the* Internet of Things.

The *Ponte* project, as presented in Chapter 2, implements such a model. We identified a set of primitives for IoT application interactions: thus we can solve the interoperability problem, and bridge between the various solution in the IoT field.

Ponte enables companies and engineers to design IoT applications by cherry picking the best technologies, without sticking a single vendor and solution family. Thus, we evaluated various protocols and solutions in various condition of delay and error on the link, and we assessed the overall application latency in Chapters 3 and 4.

Ponte enables the creation of a distributed hub for *the* Internet of Things: who can access that data? The right and availability of privacy is in state of great flux. In fact, we might assume that everything we do online is

public, as it can be logged and wiretapped easily. Moreover, *the* Internet of Things might allow unprecedented availability of data about ourselves and our environment. Thanks to the Big Data movement, unprecedented correlation and prevision will be possible. While technology progresses in that direction, we believe that every person must be in control on how his data is processed, stored, and accessed.

The algorithms and data structures presented in Chapter 5 allows everyone to specify who and how the data about themselves can be accessed. However, these approaches have been tried before and failed due to the cost of data filtering at high scale. In order to make it viable, we devised new algorithms to allow faster processing, up to 10 times the state of art. Thus, privacy data filtering is now possible with a low overhead.

This work can be extended in several directions. Firstly, *Ponte* should be extended to support more IoT protocols, e.g. ZigBee. Secondly, *Ponte* needs to be extended to have native privacy support to allow people to specify how their data can be processed and accessed. Thirdly, the amount of data that the IoT will produce need to be stored: a new data storage system needs to be evaluated and customized to support the variety of data and applications of the bright and connected future that the IoT will give us.

Acknowledgements

In 1988, when my father put me in front of that clunky keyboard, nobody thought that I would have spent a great part of my life staring at a screen, pushing random keys on now-stilish keyboards. Even in difficult times, my parents always encouraged and supported me to develop my passion. Thanks.

Ten years ago, when I fell in love with Anna, I did not know that she would become the most important person of my life. Anna is my secret sauce. In the last ten years, she made me a better person and she still fixes all my mess. I love you, Anna.

Raffaella and Enzo have always helped me and offered their invaluable advices. Thanks for considering me family.

I would like to thank the monster gang, Francesco, Jonathan and Nicolò, to the endless hours put into *not* studying when we were undergrads. Thanks monsters.

I would like to thank my tutors, Giovanni and Alessandro, to introduce me to this amazing field and to offer me the possibility to develop my carrer among their team. Thanks Francesco and Giulio for the mutual support in this almost-ending process!
I owe much of what I have achieved to my friends at Mavigex: Isabella, Daniele, Claudio, Alessio, Rosalba, and Max. Thanks, there is much of all of you in this work. Unfortunately in the writing of this dissertation no balls where thrown, and no designer went mad. I miss you.
Thanks, Bologna Vista Radio Club. We should be on air again.
Living a semester abroad change your perspective of life. I would like to thank Gregor Schiele and his wife, Sabrina, for making my time in Ireland special. Gregor, I still miss our discussions about life and technology.
Finally, I had the greatest possible company when writing and revising this thesis: my little puppy dog, Puffetta (Smurfette).

Grazie.

Bibliography

[1] M. Weiser, "The computer for the 21st century," *Scientific American*, Feb. 1991. [Online]. Available: http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html

[2] "More than 50 billion connected devices," Ericsson, February 2011. [Online]. Available: http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf

[3] "It's the Beginning of a New Era: The Digital Industrial Economy," October 2013. [Online]. Available: https://www.gartner.com/newsroom/id/2602817

[4] L. Tan and N. Wang, "Future internet: The internet of things," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, vol. 5. IEEE, 2010, pp. V5–376.

[5] M. Collina, G. Schiele, A. Vanelli Coralli, and G. E. Corazza, "The ponte project: Platform architecture, primitives, and data formats for interoperability in the internet of things," *Submitted to IEEE Internet of Things Journal*, 2014.

[6] "Ponte Eclipse Project," December 2013. [Online]. Available: http://eclipse.org/ponte

[7] M. Collina, G. E. Corazza, and A. Vanelli-Coralli, "Introducing the qest broker: Scaling the iot by bridging mqtt and rest," in *Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on*. IEEE, 2012, pp. 36–41.

[8] M. Collina, A. Vanelli-Coralli, C. Caini, G. Corazza, and R. Secchi, "Latency analysis of real-time web protocols over a satellite link."

[9] R. H. Weber, "Internet of things–new security and privacy challenges," *Computer Law & Security Review*, vol. 26, no. 1, pp. 23–30, 2010.

[10] O. Sacco, M. Collina, G. Schiele, G. E. Corazza, J. G. Breslin, and M. Hauswirth, "Fine-grained access control for rdf data on mobile devices," in *Web Information Systems Engineering–WISE 2013*. Springer, 2013, pp. 478–487.

[11] H. Zimmermann, "Osi reference model–the iso model of architecture for open systems interconnection," *Communications, IEEE Transactions on*, vol. 28, no. 4, pp. 425–432, 1980.

[12] S. Bluetooth, "Bluetooth core specification v4.0," 2010.

[13] B. P. Crow, I. Widjaja, L. Kim, and P. T. Sakai, "Ieee 802.11 wireless local area networks," *Communications Magazine, IEEE*, vol. 35, no. 9, pp. 116–126, 1997.

[14] "IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer

(PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pp. 0_1–305, 2006. [Online]. Available: http://dx.doi.org/10.1109/ieeestd.2006.232110

[15] I. Podnar, M. Hauswirth, and M. Jazayeri, "Mobile push: Delivering content to mobile users," in *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*. IEEE, 2002, pp. 563–568.

[16] B. Latré, P. De Mil, I. Moerman, N. Van Dierdonck, B. Dhoedt, and P. Demeester, "Maximum throughput and minimum delay in ieee 802.15.4," in *Mobile Ad-hoc and Sensor Networks*. Springer, 2005, pp. 866–876.

[17] N. Sastry and D. Wagner, "Security considerations for ieee 802.15.4 networks," in *Proceedings of the 3rd ACM workshop on Wireless security*. ACM, 2004, pp. 32–42.

[18] S. Tozlu, "Feasibility of wi-fi enabled sensors for internet of things," in *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*. IEEE, 2011, pp. 291–296.

[19] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012.

[20] S. Kamath, "Measuring bluetooth low energy power consumption," *Texas instruments application note AN092, Dallas*, 2010.

[21] K. Mikhaylov, N. Plevritakis, and J. Tervonen, "Performance analysis and comparison of bluetooth low energy with ieee 802.15. 4 and simpliciti," *Journal of Sensor and Actuator Networks*, vol. 2, no. 3, pp. 589–613, 2013.

[22] J. Nieminen, B. Patil, T. Savolainen, M. Isomaki, Z. Shelby, and C. Gomez, "Transmission of ipv6 packets over bluetooth low energy draft-ietf-6lo-btle-00."

[23] H. Wang, M. Xi, J. Liu, and C. Chen, "Transmitting ipv6 packets over bluetooth low energy based on bluez," in *Advanced Communication Technology (ICACT), 2013 15th International Conference on.* IEEE, 2013, pp. 72–77.

[24] J. Postel, "Rfc 791: Internet protocol," 1981.

[25] ——, "Rfc 793: Transmission control protocol," 1981.

[26] "Free Pool of IPv4 Address Space Depleted." February 2011. [Online]. Available: http://www.nro.net/news/ipv4-free-pool-depleted

[27] J. Hui and P. Thubert, "Compression format for ipv6 datagrams over ieee 802.15.4-based networks," 2011.

[28] J. Nieminen, "Transmission of ipv6 packets over bluetooth low energy," 2013. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-6lo-btle/

[29] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[30] J. Postel, "Rfc 768: User datagram protocol," 1980.

[31] S. R. Das, E. M. Belding-Royer, and C. E. Perkins, "Ad hoc on-demand distance vector (aodv) routing," 2003.

[32] "MQTT," April 2012. [Online]. Available: http://mqtt.org

[33] Z. Shelby, K. Hartke, and C. Bormann, "Constrained application protocol (coap)," 2013.

[34] K. Hartke, "Observing resources in coap," 2013.

[35] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," 1999.

[36] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Trans. Internet Technol.*, pp. 115–150, May 2002.

[37] L. Masinter, T. Berners-Lee, and R. T. Fielding, "Uniform resource identifier (uri): Generic syntax," 2005.

[38] "The WebSocket API," World Wide Web Consortium (W3C), April 2012. [Online]. Available: http://dev.w3.org/html5/websockets/

[39] I. Hickson, "Server-sent events," World Wide Web Consortium (W3C), 2012.

[40] "OASIS -Advancing open standard for the information society consortium." January 2014. [Online]. Available: https://www.oasis-open.org/

[41] "OASIS MQTT Technical Committee," January 2014. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt

[42] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s—a publish/subscribe protocol for wireless sensor networks," in *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on.* IEEE, 2008, pp. 791–798.

[43] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[44] M. Kovatsch, S. Mayer, and B. Ostermaier, "Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on.* IEEE, 2012, pp. 751–756.

[45] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *Proc. Internet of Things (IOT)*, Tokyo, Japan, December 2010, pp. 1–8.

[46] M. Blackstock and R. Lea, "Toward interoperability in a web of things," in *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication.* ACM, 2013, pp. 1565–1574.

[47] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri, "Building automation and smart cities: An integration approach based on a

service-oriented architecture," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on.* IEEE, 2013, pp. 1361–1367.

[48] A. P. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, "Web services for the internet of things through coap and exi," in *Communications Workshops (ICC), 2011 IEEE International Conference on.* IEEE, 2011, pp. 1–6.

[49] J. Schneider and T. Kamiya, "Efficient xml interchange (exi) format 1.0," *W3C Working Draft*, vol. 19, 2008.

[50] E. Dijk, "Sleepy devices using coap," 2013.

[51] "Node.js," Joyent, Inc, January 2014. [Online]. Available: http://nodejs.org

[52] G. Erich, H. Richard, J. Ralph, and V. John, "Design patterns: elements of reusable object-oriented software," *Reading: Addison Wesley Publishing Company*, 1995.

[53] A. Rudd and M. Collina, "MQTT.js," October 2013. [Online]. Available: http://github.com/adamvr/MQTT.js

[54] M. Collina, "node-coap," October 2013. [Online]. Available: http://github.com/mcollina/node-coap

[55] S. Ghemawat and J. Dean, "LevelDB," October 2013. [Online]. Available: https://code.google.com/p/leveldb/

[56] "MongoDB," October 2013. [Online]. Available: http://mongodb.org

[57] S. Sanfilippo, "Redis," April 2012. [Online]. Available: http://redis.io

[58] E. Fredkin, "Trie memory," *Communications of the ACM*, vol. 3, no. 9, pp. 490–499, 1960.

[59] "RabbitMQ," October 2013. [Online]. Available: http://www.rabbitmq.com/

[60] "Ponte Source Code," October 2013. [Online]. Available: http://github.com/mcollina/ponte

[61] N. Sharma, "Push technology–long polling." [Online]. Available: http://www.ijcsmr.org/vol2issue5/paper398.pdf

[62] "Mosquitto," October 2013. [Online]. Available: http://mosquitto.org

[63] E. G. Davis, A. Calveras, and I. Demirkol, "Improving packet delivery performance of publish/subscribe protocols in wireless sensor networks," *Sensors*, vol. 13, no. 1, pp. 648–680, 2013.

[64] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino, and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," in *Communications and Vehicular Technology in the Benelux (SCVT), 2013 IEEE 20th Symposium on*. IEEE, 2013, pp. 1–6.

[65] M. Carbone and L. Rizzo, "Dummynet revisited," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 2, pp. 12–20, 2010.

[66] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, "A first look at cellular machine-to-machine traffic: large scale measurement and characterization," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1. ACM, 2012, pp. 65–76.

[67] J. Hant, D. Lanzinger, and D. Sklar, "Assessing the performance of packet retransmission schemes over satellite links," in *Aerospace Conference, 2006 IEEE*. IEEE, 2006, pp. 13–pp.

[68] E. Balandina, Y. Koucheryavy, and A. Gurtov, "Computing the retransmission timeout in coap," in *Internet of Things, Smart Spaces, and Next Generation Networking*. Springer, 2013, pp. 352–362.

[69] J. J. Garrett *et al.*, "Ajax: A new approach to web applications," 2005.

[70] D. Crockford, "The application/json media type for javascript object notation (json)," 2006.

[71] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," RFC 3135, June, Tech. Rep., 2001.

[72] J. Resig *et al.*, "jquery: The write less, do more, javascript library," 2014.

[73] G. E. Krasner, S. T. Pope *et al.*, "A description of the model-view-controller user interface paradigm in the smalltalk-80 system," *Journal of object oriented programming*, vol. 1, no. 3, pp. 26–49, 1988.

[74] J. Ashkenas *et al.*, "Backbone.js," 2014.

[75] M. Belshe and R. Peon, "Spdy protocol," 2012.

[76] A. Cardaci, L. Caviglione, A. Gotta, and N. Tonellotto, "Performance evaluation of spdy over high latency satellite channels," in *Personal Satellite Services*. Springer, 2013, pp. 123–134.

[77] "Apache http server project," Apache Software Foundation, January 2014. [Online]. Available: http://httpd.apache.org

[78] "Nginx," Nginx Inc., January 2014. [Online]. Available: http://nginx.org

[79] "Oracle vm virtualbox," Oracle, January 2014. [Online]. Available: https://www.virtualbox.org

[80] "EU investigating IoT regulations," April 2012. [Online]. Available: http://bit.ly/HyYSb2

[81] D. Giusto, A. Iera, G. Morabito, L. Atzori, C. M. Medaglia, and A. Serbanati, "An overview of privacy and security issues in the internet of things," in *The Internet of Things*. Springer New York, 2010, pp. 389–395.

[82] D. Beckett and T. Berners-Lee, "Turtle - terse rdf triple language," *W3C Candidate Recommendation*, 2013.

[83] F. Manola, E. Miller, and B. McBride, "Rdf primer," *W3C recommendation*, vol. 10, pp. 1–107, 2004.

[84] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 3, pp. 1–22, 2009.

[85] "MsgPack," October 2013. [Online]. Available: http://msgpack.org

[86] "BSON," October 2013. [Online]. Available: http://bsonspec.org

[87] "Bysant," October 2013. [Online]. Available: http://wiki.eclipse.org/images/6/6a/M3DABysantSerializer.pdf

[88] "Protocol Buffers," October 2013. [Online]. Available: https://developers.google.com/protocol-buffers

[89] "W3c ssn incubator group review of sensor and observation ontologies." [Online]. Available: http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report#Review_of_Sensor_and_Observation_ontologies

[90] G. Stevenson, S. Knox, S. Dobson, and P. Nixon, "Ontonym: a collection of upper ontologies for developing pervasive systems," in *Context, Information and Ontologies (CIAO'09), 1st Workshop on*, 2009, pp. 1–8.

[91] M. Eid, R. Liscano, and A. E. Saddik, "A universal ontology for sensor networks data," in *Computational Intelligence for Measurement Systems and Applications (CIMSA 2007), IEEE International Conference on*, 2007, pp. 59–62.

[92] C. Goodwin and D. J. Russomanno, "An ontology-based sensor network prototype environment," in *Information Processing in Sensor Networks, Fifth International Conference on*, 2006.

[93] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog

et al., "The ssn ontology of the w3c semantic sensor network incubator group," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 17, pp. 25–32, 2012.

[94] W. Wei and P. Barnaghi, "Semantic annotation and reasoning for sensor data," in *Smart sensing and context (EuroSSC'09), 4th European conference on*, 2009, pp. 66–76.

[95] C. Stasch, K. Janowicz, A. Bröring, I. Reis, and W. Kuhn, "A stimulus-centric algebraic approach to sensors and observations," in *GeoSensor Networks (GSN'09), 3rd International Conference on*, 2009, pp. 169–179.

[96] M. Strimpakou, I. Roussaki, and M. E. Anagnostou, "A context ontology for pervasive service provision," in *Advanced Information Networking and Applications (AINA 2006), 20th International Conference on*, 2006.

[97] H. Chen, F. Perich, T. Finin, and A. Joshi, "Soupa: Standard ontology for ubiquitous and pervasive applications," in *Mobile and Ubiquitous Systems: Networking and Services, International Conference on*, 2004.

[98] M. Sporny, G. Kellogg, and M. Lanthaler, "JSON-LD 1.0-A JSON-based Serialization for Linked Data," *W3C Working Draft*, 2013.

[99] M. Lanthaler and C. Gütl, "On using json-ld to create evolvable restful services," in *Proceedings of the Third International Workshop on RESTful Design*. ACM, 2012, pp. 25–32.

[100] O. Sacco and J. G. Breslin, "PPO & PPM 2.0: Extending the privacy preference framework to provide finer-grained access control for the web of data," in *I-SEMANTICS '12*, 2012. [Online]. Available: http://doi.acm.org/10.1145/2362499.2362511

[101] O. Sacco and A. Passant, "A Privacy Preference Ontology (PPO) for Linked Data," in *Linked Data on the Web Workshop*, ser. LDOW'11, 2011.

[102] ——, "A Privacy Preference Manager for the Social Semantic Web," in *SPIM Workshop*, 2011.

[103] O. Sacco, A. Passant, and S. Decker, "An Access Control Framework for the Web of Data," in *IEEE TrustCom-11*, 2011.

[104] D. Le-Phuoc, J. X. Parreira, V. Reynolds, and M. Hauswirth, "RDF On the Go: An RDF Storage and Query Processor for Mobile Devices," in *Posters and Demos of the ISWC 2010*, 2010.

[105] P. Kärger and W. Siberski, "Guarding a Walled Garden Semantic Privacy Preferences for the Social Web," *The Semantic Web: Research and Applications*, 2010.

[106] T. Ryutov, T. Kichkaylo, and R. Neches, "Access control policies for semantic networks," in *POLICY*, 2009.

[107] F. Giunchiglia, R. Zhang, and B. Crispo, "Ontology Driven Community Access Control," *Trust and Privacy on the Social and Semantic Web, SPOT'09*, 2009.

[108] B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham, "A Semantic Web Based Framework for Social Network Access Control," ser. SACMAT'09, 2009.

[109] O. D. De Coi J.L., "A review of trust management, security and privacy policy languages." in *International Conference on Security and Cryptography*, ser. SECRYPT'08, 2008.

[110] J. Hollenbach and J. Presbrey, "Using RDF Metadata to Enable Access Control on the Social Semantic Web," in *CK'09*, 2009.

[111] P. Reddivari, "Policy based access control for a rdf store," in *In Proceedings of the Policy Management for the Web Workshop, A WWW 2005 Workshop*, 2005.

[112] F. Abel, J. L. De Coi, N. Henze, A. W. Koesling, D. Krause, and D. Olmedilla, "Enabling advanced and context-dependent access control in rdf stores," in *ISWC'07/ASWC'07*, 2007.

[113] P. Bonatti and D. Olmedilla, "Driving and monitoring provisional trust negotiation with metapolicies," in *POLICY*, 2005.

[114] L. Costabello, S. Villata, F. Gandon *et al.*, "Context-aware access control for rdf graph stores," in *ECAI*, 2012.