# ~~Survey on Technologies~~ for Enabling Real-Time Communication in the Web of Things

**Piotr Krawiec, Maciej Sosnowski, Jordi Mongay Batalla, Constandinos X. Mavromoustakis, George Mastorakis and Evangelos Pallis**

**Abstract** The Web of Things (WoT) can be considered as a step towards the Internet of Everything development. The concept of WoT assumes that objects of the Internet of Things (IoT) seamlessly interact with the Web by re-using web protocols wherever possible. One of the most desirable service in the WoT is real-time communication, due to the event-driven character of many IoT applications. This chapter provides an overview of the protocols which are taken into account in order to ensure real-time interaction of WoT objects. We describe two technologies: WebSocket and WebRTC, which are a part of HTML5 specification and are considered as solutions that bring real-time communication capabilities into the WoT. CoAP, a specialized protocol for use in resource constrained devices, is also presented, as well as two solutions that implement publish/subscribe interaction model. Next, we discuss which protocols can have the greatest impact on the WoT development.

P. Krawiec (✉) · M. Sosnowski · J.M. Batalla
National Institute of Telecommunications, Warsaw, Poland
e-mail: P.Krawiec@itl.waw.pl

M. Sosnowski
e-mail: M.Sosnowski3@itl.waw.pl

P. Krawiec · M. Sosnowski · J.M. Batalla
Warsaw University of Technology, Warsaw, Poland

C.X. Mavromoustakis
University of Nicosia, Nicosia, Cyprus
e-mail: mavromoustakis.c@unic.ac.cy

G. Mastorakis · E. Pallis
Technological Educational Institute of Crete, Crete, Greece
e-mail: gmastorakis@staff.teicrete.gr

J.M. Batalla et al. (eds.), *Beyond the Internet of Things*,
Internet of Things, DOI 10.1007/978-3-319-50758-3_13

## 1 Introduction

The next, 5th generation of mobile system (5G) is being designed to meet the requirements of serving billions of low-cost wireless nodes, such as sensors, actuators or wereables. Consequently, it is anticipated that 5G will provide the foundation for widespread deployment of the Internet of Things (IoT) [1, 2]. Since mobile networks systematically transform to all-IP architecture, also 5G IoT will be based in general on reusing well-known IP stack wherever possible [3], ensuring unified, homogeneous communication between all entities at the transport layer [4, 5]. The similar trend has emerged also on the application level [6, 7]. Conjunction IoT objects and devices with well-established World Wide Web technologies (HTTP, HTML, XML, JSON etc.) alters IoT into the Web of Things (WoT) [8, 9].

The driving forces that have brought the leading role of the Web in many spheres of human activity (business, social, entertainment etc.) are web services. They represent the building blocks of web-based distributed applications, which have introduced innovations and added values in traditional business sectors as logistic or banking, as well as stimulated the emergence of new gain-revenue “e-*” markets such as e-commerce or e-entertainment. Web services, in accordance with Service Oriented Architecture (SOA), are offered to clients through unified interface, regardless of which technology was used for their implementation and who controls them. In this way, web services provide seamless communication and unified access to different functional entities within and across domains. They are a convenient tool for integration and control of various, distributed systems, which allows implementation of complex scenarios (for example: web application for advanced energy cost optimization which binds building automation system with virtual energy market place and distribution grid control).

Web of Things concept unifies interaction of IoT objects with web applications by reusing standard web protocols. Data from a huge number of different sensors and devices, interconnected with the Internet, will be presented on the Web and accessible for web services. This gives an opportunity to elevate web applications to a completely new level, by integrating the present and future web services with, derived from IoT devices, information about the physical world. In turn, WoT that brings IoT to the web services landscape, can be considered as a significant contributor to the Internet of Everything (IoE), which includes ubiquitous interactions of machines (i.e. IoT objects) and humans (i.e. final consumers of web services).

Many usage scenarios for IoT require that data, which are linked with detected changes in surrounding environment, should be almost instantaneously available for the use by web services. Consequently, there is a need for a real-time communication, at the application level, to ensure that information is disseminated between IoT objects and the Web without undue delay.

In this chapter we give an overview of existing, commonly used technologies for real-time communication in the Web, and analyze their suitability for Web of Things development. Our survey focuses only on solutions that can be applicable for resource-constrained devices. Furthermore, we discuss the future development trends and forecast for WoT real-time services.

The chapter is structured as follows. In Sect. 2 we briefly introduce a concept and requirements for development of real-time applications in WoT domain. Section 3 presents the main enabling protocols for WoT real-time services. The next Section provides an outlook for future trends and research. Finally, in Sect. 5 we conclude our survey.

## 2 Real-Time Communication in WoT Domain

World Wide Web has come a long way during the last twenty years. The emergence of Web 2.0, the SOA concept and web services resulted in the Web transformation from a set of simple, static web-pages to the complex ecosystem, with powerful web applications influencing many fields of everyday life. Such transformation caused that original communication model based on transferring batched data from server to a client upon request, has become insufficient. Nowadays, many web applications require real-time communication capabilities, to forward a message to a client as soon as an appropriate data appears on the server side, without waiting for explicit request from the client. For example, web applications for supporting decision making process often rely on accurate information provided by different web components that perform real-time analytics (such components can be fed with data obtained from e.g. social media platforms). To cope with this issue, many solutions have been proposed in recent years to enable real-time communication in the Web, such as Ajax polling and long-polling techniques [10], WebSocket [11] or WebRTC [12], just to mention a few.

It should be noted that considering the Web domain, the term “real-time” does not refer to strict delay constraints, violation of which results in some critical errors that lead to the entire system failure. Such constraints, described as *hard* real-time [13], are characteristic for industrial automation sector where they are usually met by using dedicated infrastructure. In the case of web applications, we have in mind *soft* (a.k.a. *near*) real-time requirements. It means that there is no sharp deadline for data delivery. However, increasing delay in information transfer has a very negative impact on service quality perceived by the user.

Real-time communication becomes even more important when we consider IoT scenarios. IoT concept assumes on-line observation and manipulation of the surrounding environment through different entities (IoT objects) equipped with sensors and actuators. A user of an IoT application expects approximately the same reaction times as he/she experiences in physical world. When the user uses a smartphone to turn on a room light, the bulb should illuminate almost immediately, as it is when he/she uses a physical light switch. Likewise, an output data of sensors should be quickly transferred to IoT applications, because the user will not tolerate stale information about, usually rapidly changing, physical world. In complex scenarios, which require closely synchronized control over multiple distributed actuators, real-time delivery technologies will be an absolute necessity.

The SOA design principle, the backbone of the current web development, is implemented nowadays in accordance with one of two major service-oriented architectures:

- Web Services that use remote procedure calls (RPC) for client-server interactions. This technology is commonly referred as *WS*-* architecture. WS-* relies on SOAP (Simple Object Access Protocol [14]) messages, which carry XML (eXtensible Markup Language [15]) payload and are exchanged between web components through HTTP-based transport mechanism. The primary idea of WS-* service is to manipulate web resources using an arbitrary set of operations. WS-* architecture is considered as a very well suited for enterprise IT systems [16].
- Representation State Transfer (REST) architecture [17]. It assumes that all entities in the Web are abstracted as resources, which are addressable and searchable via Uniform Resource Identifiers (URI). RESTful web services exploit HTTP as an application protocol and manipulate web resources using a uniform set of operations (HTTP's methods: GET, PUT, POST and DELETE). Each end-point is responsible for maintaining its state, what has a positive impact on overall system scalability and robustness.

Although dedicated DPWS (Devices Profile for Web Services) specification [18] was proposed to meet IoT requirements in WS-* domain, the WS-* web services seem to be less favored as the way for implementing Web of Things. The main reason is an extensive overhead (in terms of computational and communication resources) linked with XML-based SOAP messages handling [19]. Stateless, lightweight RESTful services are a preferable choice in the context of distributed IoT web applications [20, 21]. No need for maintaining states at server side is crucial if we consider a huge number of IoT objects that can be involved in many IoT scenarios. On the other hand, in REST concept each request carries all the information necessary for a receiver to understand the request and recognize the sender state. Therefore, resource identifiers and state descriptions should be carefully designed, taking into account limitations in processing power and memory capacity of IoT devices.

However, the REST supports point-to-point communication only, which is carried out between client and server. It is definitely not sufficient for IoT purposes. Using IoT web application the user, with a single click, should turn on multiple light sources in a room. On the other hand, a light source should comply with control messages obtained from several entities: users (via user interface), light sensors, presence detectors etc. Therefore, traditional REST architecture considered in the WoT context needs to be enhanced with another models of interactions: one-to-many and many-to-many. Such models can be implemented based on *publish/subscribe* communication pattern. It assumes that interacting entities act as a message publisher (i.e. data sender) or/and a subscriber (i.e. data receiver). The

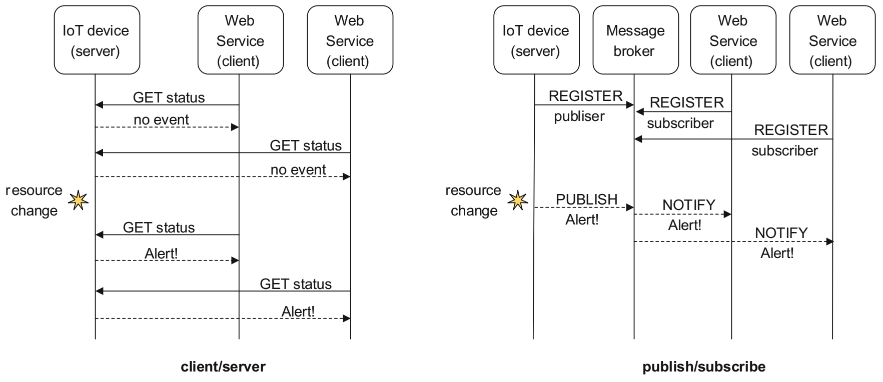

**Fig. 1** Client/server (polling) versus publish/subscribe (push) interaction models

publisher sends a message without receiving direct requests from interested entities and even without explicit indication of message receivers. In turn, the subscribers are provided only with messages that they had earlier expressed interest in.

In comparison with the client/server model, publish/subscribe systems ensure high efficiency and scalability since IoT objects can push messages to announce any changes in their states just after detecting them, and there is no need to establish a separate connection with each involved receiver (as it is in the case of traditional polling—see Fig. 1). Publish/subscribe approach can be realized in two ways: (1) as a fully distributed system bases on multicast or peer-to-peer communication, or (2) with the support of an intermediate point, a *message broker*, which implements "store and forward" functionality. The broker is responsible for receiving messages from publishers and distributing them among registered subscribers.

## 3 Protocols for Real-Time Interactions with WoT Objects

The current Web takes advantage of different approaches to achieve real-time communication. However, some of them are too costly for resource-constrained IoT devices. For example, Ajax long-polling technique requires maintaining connections between IoT objects and remote peers at HTTP level, what is highly resource-consuming.

In this chapter we present an overview of protocols for real-time interactions proposed for WoT domain. The survey covers both the approaches that base on the current web technologies (WebSocket, WebRTC) and are compatible with REST principles that exploit client/server communication (CoAP), as well as the solutions that are in line with publish/subscribe paradigm (MQTT, AMQP).

## *3.1 WebSocket*

WebSocket protocol was specified by IETF [11] in 2011 and afterwards incorporated by World Wide Web Consortium (W3C) into the latest HTML standard denoted as HTML5 [22]. It allows a client to establish a full-duplex connection with a remote web server by using single TCP socket.

The connection establishment is initiated by HTTP handshake procedure, during which the client provides to the server an HTTP Upgrade header to trigger protocol switching. Next, the WebSocket connection is created over already established TCP session that uses standard HTTP or HTTPS port number (80 or 443, respectively). In this way, WebSocket fits well to the environment where end-to-end communication is broken for non-HTTP traffic due to existence of firewalls, NATs or proxies.

Established connection is keeping open as long as needed, allowing asynchronous data exchange between a client and a server (WebSocket standard defines short Ping and Pong messages to implement keep-alive mechanism). Using this connection, an IoT object (a server) can push a message to, for example, a web application (a client) whenever it obtains new data to send. Comparing to previously proposed real-time web techniques, such as Ajax polling and long-polling, WebSocket has much better server performance because maintaining an open low-level (i.e. TCP) connection consumes a small amount of server resources. In contrast, asynchronous communication provided by Ajax technologies requires opening and handling several HTTP connections, which are being closed due to timeouts or when HTTP response was sent (see Fig. 2), what has negative impact on server performance. Moreover, Websocket leads to latency reduction [23], since every re-establishment of HTTP connection, as it occurs in long-polling, requires performing additional TCP three-way handshake. Another advantage of WebSocket

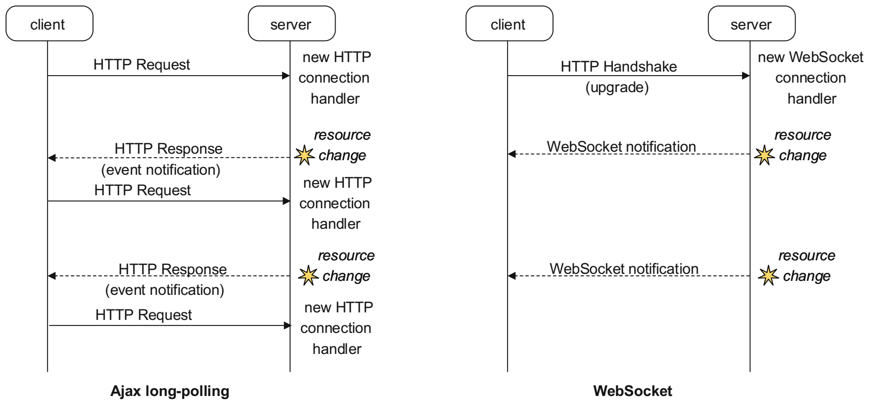

**Fig. 2** Sequence diagrams for Ajax long-polling and WebSocket real-time communication technologies

is a very low protocol overhead [18]. Data transferred through WebSocket channel are organized into frames with a header size of 2 bytes only.

Presented features make the WebSocket considered as a good candidate to ensure real-time communication in WoT. In [24] authors propose to use this protocol to connect WSN (Wireless Sensor Network) gateway with sensor cloud service in IoT industrial production system, whereas solution presented in [25] demonstrates using WebSocket to set up communication between smartphone and system controller deployed on Raspberry Pi platform, in home automation scenario.

WebSocket standard specifies two types of application payload: UTF-8 text and binary data. However, during HTTP handshake a client can indicate the use of a specific subprotocol, i.e. an application-level protocol layered over WebSocket connection. One of the registered subprotocol is WAMP (Web Application Messaging Protocol) [26], which offers publish/subscribe communication pattern using URIs as identifiers and JSON for message serialization.

## *3.2 WebRTC*

HTML5 recommendation specifies also WebRTC [12], a technology which provides real-time communication between web clients (in contrast to WebSocket, which operates on client-server connections). The primary goal of WebRTC is to enable inter-personal multimedia services using standard, built-in functionalities of web browsers. In this way, WebRTC introduces a real-time voice and video communication straight into web services, eliminating the need to install dedicated software or plug-ins.

Although WebRTC allows web applications to send and receive streaming data directly to/from remote peers, without relying it through an intermediate server, it still needs out-of-band signaling to initialize peer-to-peer communication. This functionality is typically provided through a dedicated web server (see Fig. 3). WebRTC specification assumes Session Description Protocol (SDP) for connection parameters negotiation. On the other hand, the specification does not define concrete technologies to be used for exchanging SDP messages between peers. Nevertheless, applied signaling protocol should be layered over HTTP (or WebSocket) to easily pass through firewalls and proxies as a standard web traffic. The analogous problem applies to data plane. To cope with this issue, WebRTC exploits RTP/Secure RTP (Real-time Transport Protocol [27]) for media transport and it requires encapsulation of RTP packets into ICE (Interactive Connectivity Establishment) protocol [28], which provides a set of techniques for traversing NATs and firewalls.

Apart from *RTCPeerConnection* API (Application Programming Interface) for creating audio and video connections, WebRTC provides also *RTCDataChannel* API, which enables peer-to-peer data sharing between web clients. The WebRTC data channel is constructed using Stream Control Transmission Protocol (SCTP [29]) as a generic transport service, which is encapsulated in DTLS (Datagram

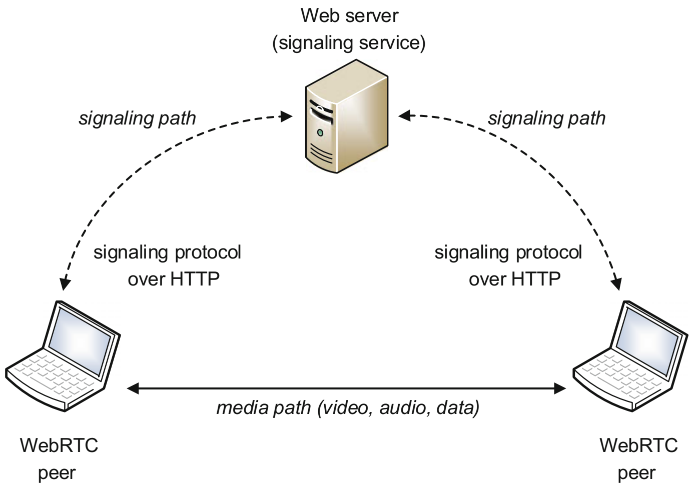

**Fig. 3** WebRTC communication model

Transport Layer Security [30]), and next layered over ICE [31]. The data channel is considered as a solution which can merge WebRTC with the IoT, since it will enable to enrich inter-personal real-time communication with information about surrounding environment available at IoT domain. In this way, WebRTC can be thought as a tool for easy integration of voice and video services with IoT objects within one application. It can apply in various IoT scenarios. For example, during WebRTC video teleconsultation a physician can have real-time access to sensor data gathered by patient's personal e-health wearables. There are already available platforms such as open-source Streembit [32] or commercial Skedans [33], which exploit WebRTC to provide real-time collaboration between humans and smart devices (e.g. drones).

## *3.3 CoAP*

Constrained Application Protocol (CoAP) [34] is a solution specifically designed to operate in a resource-constrained IoT environment. It is based on the REST concept and exploits client/server communication model to manipulate the resources using well-known GET, PUT, POST and DELETE requests of the HTTP. Likewise, CoAP response codes are in accordance with the HTTP specification. In this way, CoAP is characterized by easy integration of IoT devices with existing web services.

Furthermore, CoAP is a very lightweight. It is binary protocol and uses fixed header with a length of 4 bytes only, which may be followed by additional options header. Comparison presented in [35] shows that average transaction size in bytes is almost 10 times smaller in CoAP than in HTTP. Additionally, CoAP operates over UDP to avoid bandwidth and energy-consuming TCP mechanisms, such as three-way handshake and automatic packet retransmissions. Since UDP is a stateless protocol, CoAP introduces two abstract sub-layers. The upper sub-layer is responsible for requests/responses handling, whereas the lower controls data transmission through UDP using four types of messages:

- Confirmable (CON)—the message must be acknowledged;
- Non-confirmable (NON)—acknowledgement of the message is not required;
- Acknowledgement (ACK)—it acknowledges the CON message;
- Reset (RST)—it is used to indicate that received request cannot be properly processed.

CON messages are designed for establishing reliable communication based on retransmissions and timeout mechanisms (see Fig. 4). Real-time data transfer, for which retransmissions are undesired due to excessive delay, can be realized using NON messages. Moreover, NON messages are used for implementing multicast connections [36].

Besides client/server interaction model, CoAP offers also a publish/subscribe functionality [37], which is triggered by setting the *observe* option in the GET request. The server that receives request with this option, registers the sender as the observer of the requested resource. Then, whenever the state of the resource changes, the server sends a notification to all clients from the list of registered observers. To remove a subscription, a client uses a RST message as a response to notification received from a server.

There are many CoAP implementation available nowadays. Some of them are designed for non-constrained devices, such as jCoAP [38] intendent for Java-based mobile devices (e.g. smartphones with Android operating system). Other implementations are highly optimized and can be used as a build-in modules in embedded operating systems. The most popular are *Erbium* [39] used in Contiki and *libcoap* [40], which has been ported to TinyOS.

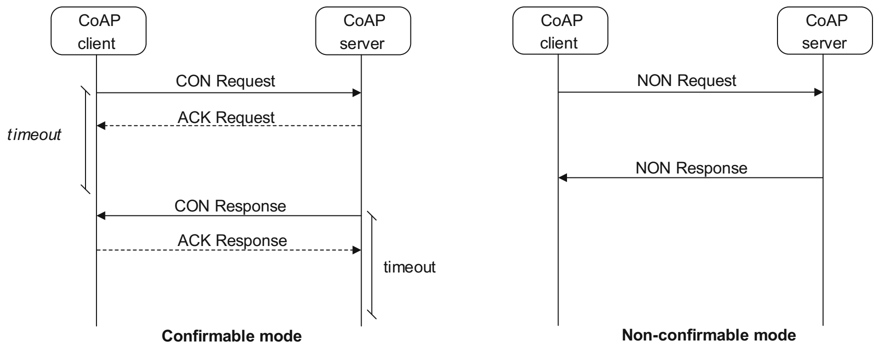

**Fig. 4** Message exchange diagram for CoAP confirmable and non-confirmable modes

## *3.4 MQTT*

Message Queue Telemetry Transport (MQTT) [41] is an open, asynchronous publish/subscribe protocol that operates over TCP connections. It is simple and lightweight (2 bytes header), since it was directly designed for resource-constrained devices and low-bandwidth, high-latency or unreliable networks. Hence, it has been proposed for various IoT applications, such as home automation system [42], personal e-health system [43] or to control Smart Lights [44].

The communication between MQTT publisher and subscriber is provided by intermediating broker. Publishers publish messages to specified topics, and a broker distributes them among the registered subscribers of given topic. The topics are composed by slash-separated strings; each string is considered as a level, and subscribers can use wildcards to express interest in a group of topics (+—match all strings on one level; #—match all strings below a level). MQTT broker may hold last published message on each topic (*retained message*) and next send it to a newly connected subscriber, so the subscriber obtains last good value immediately after registration.

The protocol attempts to ensure reliability and assurance in message delivery. It provides three QoS (Quality of Service) levels of guarantees for delivering a message:

- Level 0—a message delivered at most once, no confirmation required;
- Level 1—a message delivered at least once, confirmation required;
- Level 2—a message delivered exactly once, with a handshake procedure.

Higher QoS level needs more communication between a broker and MQTT client so there is a trade-off between QoS and protocol overhead. MQTT defines also the Last Will and Testament (LWT) feature for stressing publisher abnormal disconnection. When a publisher disconnects ungracefully, LWT message (defined by the publisher during registration) is send to its subscribers to inform them about device or network failure. LWT message is ignored when a publisher ends communication with a broker in a proper manner (i.e. by sending DISCONNECT message).

## *3.5 AMQP*

Advanced Message Queueing Protocol (AMQP) [45, 46] is an open, TCP/IP-based publish/subscribe protocol that can be used for brokering of messages between different processes or systems regardless of their internal designs. AMQP broker handles messages by *exchanges* which route the messages to none, one or many queues according to specified rules (called *bindings*) depending on a usage scenario. When publishing a message, publishers may specify various meta-data to be used by an exchange in queue selecting process, but typically a queue is bound to an exchange by usage of a subscription key.

Each AMQP message contains the routing key, which is composed as a chain of strings separated by dots—each string is considered as a level. The subscription keys used by queues are similar to routing keys, but may contain two wildcards, as it is in MQTT. A message goes to a queue when both keys match.

The protocol was designed for business application, where reliability is a must. Therefore, subscribers must acknowledge acceptance of each message what ensures its delivery to an application (TCP protocol guarantees data delivery only at the transport layer). Queues may provide temporary storage when the destination is busy or not connected. Since the message brokering task is divided into exchanges and message queues, it gives more flexibility for developers in designing applications. A message can go through a chain of AMQP brokers and can be processed or modified in many places, as well as be processed in parallel. Moreover, AMQP is a binary protocol so its overhead is quite low, with fixed-size header of 8 bytes. The authors of [47] exploited AMQP in their *Stack4Things* platform designed for managing IoT devices in Smart City scenario.

## 4 Discussion and Future Trends Analysis

The power of the WoT concept is the seamless integration of IoT objects with the whole web ecosystem. While different vendor-specific solutions cover a large part of the current market (survey presented in [48] shows that almost a half of respondents deployed custom written IoT protocol in their networks), we think that in the near future systems that base on standard web technologies will become predominant. Simplicity is the key to widespread implementation and use of IoT services, while web-based IoT solutions take advantage of the open HTTP protocol and therefore they can be easily integrated with web services without the need for significant investments in a new infrastructure. Furthermore, the possibility of using the existing knowledge, in terms of skilled engineers and deployment best practices, and also well-known tools, could result in another reduction of WoT implementation costs.

In the context of WoT, a real-time communication is of particular importance. Unfortunately, such communication is very difficult to achieve by IoT devices with traditional web mechanisms as HTTP request/response or Ajax polling.

The release of HTML5 specification can cope with this issue, and especially the WebSocket protocol. It enables bidirectional real-time interaction between a client and a server through always-on TCP connection. Compared with traditional HTTP techniques, it decreases the workload of a server and introduces significantly lower communication overhead. Another advantage is a full compatibility with the Web, since it uses HTTP request during initial phase of connection establishment. It also provides built-in security by taking advantage of TLS (Transport Layer Security [49]) cryptographic protocol. WebSocket may become a major candidate in each scenario, where involved IoT devices have enough resources to cope with HTTP/TCP/IP stack implementation (it is worth noting that there has been developed implementations of a web server, for which ROM and RAM memory requirements do not exceed a dozen kilobytes [50]). What is more, WebSocket does not need full HTTP stack implementation. Only a minimal set of functions is required, with Upgrade option, just to handle the protocol switching procedure.

One more powerful technology introduced by HTML5 is the WebRTC. Although WebRTC and IoT intersection is broadly discussed, and several platforms which merge WebRTC with IoT devices have already been presented, in our opinion it is hard to expect that this protocol will play a significant role in the WoT development. WebRTC requires implementation of full HTTP stack, jointly with additional signaling protocols, thus it imposes too large computational and memory burden to many IoT devices (the authors in [51] indicate that the current WebRTC runtime is quite CPU intensive even in traditional web browsers).

Another key point is that WebRTC has been designed for direct client-to-client communication. In IoT there is no clear separation between a client and a server, since most devices exploit both functionalities (for example, a sensor acts as a client and periodically sends gathered data to a server and, at the same time, it is a server that listens on commands from a control web application). In that case in IoT applications we can dispose of client-to-client restriction and achieve the WebRTC functionality simply by using standard client-server solutions (for example, WebSocket).

Many IoT scenarios are strongly oriented on event-driven flowchart and they need implementation of the publish/subscribe pattern that enables asynchronous communication. This pattern provides a native support for the sporadic connectivity since it allows the IoT device to publish a notification regardless of the connection status of recipients. HTTP with its client-pull communication does not fully match the above requirements, therefore specialized protocols are considered in this area. MQTT arouses considerable interest as an easy and lightweight solution (other considered publish/subscribe protocol, AMQP, has a more reach feature set, but it introduces higher overhead compared to MQTT).

It is also possible to implement publish/subscribe paradigm using WebSocket and its subprotocol WAMP. Such approach significantly simplifies implementation and management of a web application since all modules base on HTML5 framework. WAMP is Web-native, therefore it can interact with web services without any tunneling or bridging. However, due to higher overhead introduced by the WAMP-based approach, in our opinion the MQTT will be the first choice for scenarios that cover devices with more limited resources.

If we consider highly resource-constrained IoT objects, which periodically move into sleep mode for saving energy, the TCP-based solutions as presented above, are inappropriate. Each transition into the sleep mode means termination of TCP connection, and next expensive connection reestablishment. In that case integration with the Web can be performed with the CoAP protocol, which is dedicated for devices with limited computational and communication resources, and exploits connectionless UDP. As presented in [48], CoAP is not widely adopted yet, however in the future it is anticipated as one of the major players in IoT.

Nevertheless, from the Web point of view, CoAP (similar as MQTT) is still a custom protocol, which requires proxies to cooperate with HTTP-based web services, or dedicated servers for hosting CoAP-enabled web applications. However, using proxies entails some drawbacks. Proxy constitutes a single point of failure—when it collapses, all IoT objects from a CoAP domain have broken connectivity with the Web, regardless of which web service (i.e. HTTP server) they use. Required translation between CoAP and HTTP messages also brings in an extra latency, what might be problematic in case of real-time communications. Last but not least, using proxies in IoT environment, which is characterized by high mobility and variability, arises scalability issues. When the number of IoT objects will increase in a given CoAP domain, the objects might have a problem with access to web services due to proxy overload (even though the web services are hosted on various, not overloaded HTTP servers).

Future research directions related with WoT will focus on two main tracks: to determine how the current Web technologies should be adapted to best fit IoT requirements, and to explore new solutions that are suitable for resource-constrained IoT world and still integrate well with the Web [52]. For example, the solution for overcoming the interoperability problem may come out from a new version of HTTP protocol called HTTP/2 [53], which was approved by IETF in 2015. HTTP/2 preserves compatibility with the transaction semantics of its predecessor—it uses the same methods, headers and status codes. Consequently, it is fully compatible with the Web, bringing seamlessly web services to IoT objects without the need of using and maintaining additional devices such as proxies or gateways. Nevertheless, HTTP/2 exploits completely different mechanisms for transfer HTTP messages between endpoints. It provides efficient header compression [54] in order to reduce the protocol overhead. It also introduces a framing layer between HTTP and TCP planes, which is used for multiplexing several HTTP requests into one TCP connection.

Although HTTP/2 was designed taking into account requirements from the current Web only, its properties cause that it can be well tailored to the needs of Internet of Things. Furthermore, to cope with unnecessary delays and overhead introduced by TCP mechanisms, the replacement of TCP by QUIC protocol [55] may be investigated. QUIC is an experimental transport protocol developed by Google. It works over a connectionless UDP protocol, therefore it is characterized by reduced connection set up latency. For this reason, the further research can focus on adaptation of HTTP/2 and QUIC protocols stack to efficiently work with constrained IoT devices, what allows web applications to fully integrate with IoT objects and services offered by them. Moreover, further research activity can be related with other aspects of real-time communication, for example real-time search engines for efficient resource discovery.

## 5 Summary

Web of Things domain is much more heterogeneous compared to the traditional Web. Currently it is on its early stage of development, and we can distinguish several various solutions which are used, or are anticipated to use in the near future, for the same purposes.

In this chapter we provided a brief overview of the protocols that can be used to ensure (near) real-time communication between WoT entities and analyzed their suitability to meet the WoT real-time requirements. In our opinion, the most promising is WebSocket, which provides full interoperability with existing HTTP infrastructure. One can expect that the embedded WebSocket servers will be common in future IoT resource-constrained devices, causing the users to be able to control and manage the devices simply by using regular web browsers.

Nevertheless, since there is no one protocol that can satisfy all needs, we anticipate that WebSocket-based solutions will be supported by implementation of publish/subscribe interaction model with MQTT protocol, especially when considering devices with small computational and communication capabilities. Likewise, networks which nodes are characterized by very tight resource constraints, can rely on the CoAP protocol to avoid expensive TCP connections. The latter approaches require additional mechanisms (gateways or proxies) to ensure interoperability with HTTP-based web services. For this reason, one of the future research trends will focus on solutions that match the demands of devices with very limited resources, while ensuring seamless integration with the Web.

**Acknowledgments** This work was undertaken under the Pollux II IDSECOM project supported by the National Research Fund Luxembourg and the National Centre for Research and Development in Poland.

## References

1. Chih-Lin I, Mikko A. Uusitalo, and Klaus Moessner, "The 5G Huddle (From the Guest Editors)", IEEE Vehicular Technology Magazine, Volume 10, Isssue 1, pp. 28–31, March 2015
2. J. Mongay Batalla, M. Gajewski, W. Latoszek, P. Krawiec, C X. Mavromoustakis, G. Mastorakis, "ID-based service-oriented communications for unified access to IoT", Computers and Electrical Engineering Journal, Vol. 52, Elsevier, pp. 98–113, May 2016
3. M. Gajewski, P. Krawiec, "Identification and Access to Objects and Services in the IoT Environment", Chapter on C.X. Mavromoustakis et al. (eds.), Internet of Things (IoT) in 5G Mobile Technologies, Modeling and Optimization in Science and Technologies vol. 8, Springer International Publishing, Switzerland, 2016
4. J. Mongay Batalla, G. Mastorakis, C. Mavromoustakis and J. Żurek, "On cohabitating networking technologies with common wireless access for Home Automation Systems purposes". IEEE Wireless Communications, October 2016
5. J. Mongay Batalla and P. Krawiec, "Conception of ID layer performance at the network level for Internet of Things". Springer Journal Personal and Ubiquitous Computing, Vol. 18, Issue 2, pp. 465–480, 2014

6. K. Karolewicz, A. Bęben, J. Mongay Batalla, G. Mastorakis and C. Mavromoustakis, "On efficient data storage service for IoT". International Journal of Network Management, Wiley, May 2016, pp. 1–14, doi:10.1002/nem.1932
7. J. Mongay Batalla, K. Sienkiewicz, W. Latoszek, P. Krawiec, C. X. Mavromoustakis i G. Mastorakis, "Validation of virtualization platforms for I-IoT purposes". The Journal of Supercomputing, Springer US, 2016, doi:10.1007/s11227-016-1844-2
8. D. Guinard, V. Trifa and E. Wilde, "A resource oriented architecture for the Web of Things," Internet of Things (IOT), 2010, Tokyo, 2010, pp. 1–8. doi:10.1109/IOT.2010.5678452
9. D. Guinard, V. Trifa, F. Mattern and E. Wilde, "From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices". Chapter in: Architecting the In-ternet of Things, Uckelmann, Dieter, Harrison, Mark, Michahelles, Florian (Eds.), Springer-Verlag Berlin Heidelberg, 2011, pp. 97–129
10. E. Bozdag, A. Mesbah, and A. van Deursen. "A Comparison of Push and Pull Techniques for AJAX". In Proceedings of the 2007 9th IEEE International Workshop on Web Site Evolution (WSE'07). IEEE Computer Society, Washington, DC, USA, 2007, pp. 15–22
11. I. Fette, A. Melnikov, "The WebSocket Protocol", Internet Engineering Task Force (IETF), RFC 6455, December 2011
12. H. Alvestrand, "Overview: Real Time Protocols for Browser-based Applications; draft-ietf-rtcweb-overview-15", Internet Engineering Task Force (IETF) Internet-Draft, January, 2016
13. G. C. Buttazzo, "Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications", Real-Time Systems Series. Santa Clara, CA: Springer-Verlag TELOS, 2004
14. Nilo Mitra, Yves Lafon (eds.), "SOAP Version 1.2 Part 0: Primer (Second Edition)", W3C Recommendation, The World Wide Web Consortium, April 2007. Available online: https://www.w3.org/TR/soap12-part0/ Accessed 15 Jul 2016
15. Tim Bray et al. (eds.), "Extensible Markup Language (XML) 1.0", W3C Recommendation, The World Wide Web Consortium, November 2008. Available online: https://www.w3.org/TR/xml/ Accessed 15 Jul 2016
16. C. Pautasso and E. Wilde. "Why is the web loosely coupled?: a multi-faceted metric for service design". In Proc. of the 18th international conference on World Wide Web (WWW'09), pages 911–920, Madrid, Spain, April 2009. ACM
17. R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," ACM Transactions on Internet Technology, vol. 2, May 2002
18. OASIS Specification: Devices Profile for Web Services (DPWS) Version 1.1, OASIS, July 2009. Available online: http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01 Accessed 15 Jul 2016
19. Z. Shelby, "Embedded web services," IEEE Wireless Communications 17(6), pp. 52–57, January 2011
20. D. Yazar and A. Dunkels. "Effcient application integration in IP-based sensor networks". In Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings, page 4348, Berkeley, CA, USA, November 2009
21. D. Guinard, I. Ion, S. Mayer, "In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective", Chapter in: Mobile and Ubiquitous Systems: Computing, Networking, and Services, A. Puiatti and T. Gu (eds.), Springer Berlin Heidelberg, 2012, pp. 326–337
22. Ian Hickson et al. (eds.), "HTML5. A vocabulary and associated APIs for HTML and XHTML", W3C Recommendation, The World Wide Web Consortium, October 2014. Available online: https://www.w3.org/TR/html5/ Accessed 15 Jul 2016
23. Lubbers, P., and Greco, F. "Html5 web sockets: A quantum leap in scalability for the web". SOA World Magazine (2010)
24. Peng Hu, "A System Architecture for Software-Defined Industrial Internet of Things", In Proc. of the Ubiquitous Wireless Broadband (ICUWB), 2015 IEEE International Conference on, IEEE, 2015, pp. 1–5

25. A. Hasibuan, M. Mustadi, I. E. Y. Syamsuddin and I. M. A. Rosidi, "Design and implementation of modular home automation based on wireless network, REST API, and WebSocket," 2015 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Nusa Dua, 2015, pp. 362–367
26. T. Oberstein, A. Goedde, "The Web Application Messaging Protocol; draft-oberstet-hybi-tavendo-wamp-02", Internet Engineering Task Force (IETF) Internet-Draft, October 2015
27. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", Internet Engineering Task Force (IETF), RFC 3550, July 2003
28. J. Rosenberg. "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols". Internet Engineering Task Force (IETF) RFC 5245, April 2010
29. R. Stewart (ed.), "Stream Control Transmission Protocol", Internet Engineering Task Force (IETF), RFC 4960, September 2007
30. E. Rescorla, N. Modadugu, "Datagram Transport Layer Security Version 1.2", Internet Engineering Task Force (IETF), RFC 6347, January 2012
31. H. Alvestrand, "Transports for WebRTC; draft-ietf-rtcweb-transports-06", Internet Engineering Task Force (IETF) Internet-Draft, August, 2014
32. Streembit: Decentralized, peer-to-peer, secure communication system for humans and machines. Project webpage: http://streembit.github.io/ Accessed 15 Jul 2016
33. Skedans Systems. Webpage: https://skedans.com Accessed 15 Jul 2016
34. Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", Internet Engineering Task Force (IETF), RFC 7252, June 2014
35. W. Colitti, K. Steenhaut, and N. De Caro, "Integrating Wireless Sensor Networks with the Web," in Extending the Internet to Low power and Lossy Networks (IP+SN 2011), 2011
36. A. Rahman, E. Dijk (eds.), "Group Communication for the Constrained Application Protocol (CoAP)", Internet Engineering Task Force (IETF), RFC 7390, October 2014
37. Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", Internet Engineering Task Force (IETF), RFC 7641, September 2015
38. B. Konieczek, M. Rethfeldt, F. Golatowski and D. Timmermann, "Real-Time Communication for the Internet of Things Using jCoAP," 2015 IEEE 18th International Symposium on Real-Time Distributed Computing, Auckland, 2015, pp. 134–141
39. M. Kovatsch, S. Duquennoy, and A. Dunkels. "A Low-Power CoAP for Contiki". In Proc. of IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), Valencia, Spain, October 2011
40. K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, C. Görg, "Implementation of CoAP and its Application in Transport Logistics", In Proc. of the Workshop on Extending the Internet to Low power and Lossy Networks, Chicago, IL, USA, April 2011
41. OASIS Specification: MQTT Version 3.1.1 Plus Errata 01, OASIS, July 2009. Available online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html Accessed 15 Jul 2016
42. Y. Upadhyay, A. Borole and D. Dileepan, "MQTT based secured home automation system," 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, Madhya Pradesh, India, 2016, pp. 1–4
43. Y. F. Gomes, D. F. S. Santos, H. O. Almeida and A. Perkusich, "Integrating MQTT and ISO/IEEE 11073 for health information sharing in the Internet of Things," 2015 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, 2015, pp. 200–201
44. N. K. Walia, P. Kalra and D. Mehrotra, "An IOT by information retrieval approach: Smart lights controlled using WiFi," 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence), Noida, 2016, pp. 708–712
45. S. Vinoski, "Advanced Message Queuing Protocol," in IEEE Internet Computing, vol. 10, no. 6, pp. 87–89, Nov.-Dec. 2006. doi:10.1109/MIC.2006.116

46. OASIS Specification: Advanced Message Queuing Protocol AMQP Version 1.0, OASIS, 2012. Available online: http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html Accessed 15 Jul 2016
47. G. Merlino, D. Bruneo, S. Distefano, F. Longo and A. Puliafito, “Stack4Things: Integrating IoT with OpenStack in a Smart City context,” Smart Computing Workshops (SMARTCOMP Workshops), 2014 International Conference on, Hong Kong, 2014, pp. 21–28
48. M. Thoma, T. Braun, C. Magerkurth and A. F. Antonescu, “Managing things and services with semantics: A survey,” 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, 2014, pp. 1–5
49. E. Rescorla “HTTP Over TLS” Internet Engineering Task Force (IETF), RFC 2818, May 2000
50. Deze Zeng, Song Guo, and Zixue Cheng, “The Web of Things: A Survey”, Journal of Communications, vol. 6, no. 6, September 2011
51. T. Sandholm, B. Magnusson and B. A. Johnsson, “An On-Demand WebRTC and IoT Device Tunneling Service for Hospitals,” Future Internet of Things and Cloud (FiCloud), 2014 International Conference on, Barcelona, 2014, pp. 53–60
52. J. Heuer, J. Hund and O. Pfaff, “Toward the Web of Things: Applying Web Technologies to the Physical World,” in Computer, vol. 48, no. 5, pp. 34–42, May 2015
53. M. Belshe, R. Peon, M. Thomson, (ed.) “Hypertext Transfer Protocol Version 2 (HTTP/2)” Internet Engineering Task Force (IETF), RFC 7540, May 2015
54. Peon, R. and H. Ruellan, “HPACK: Header Compression for HTTP/2”, Internet Engineering Task Force (IETF), RFC 7541, May 2015
55. J. Iyengar, I. Swett, “QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2; draft-tsvwg-quic-protocol-00”, Internet Engineering Task Force (IETF) Internet-Draft, June, 2015