

Chapter 2

IoT System Architectures

2.1 Introduction

In this chapter, we study architectures for IoT systems. We will study typical components used for networks, databases, etc.

Figure 2.1 shows the organization of an IoT system:

- The *plant* or *environment* is the physical system with which the IoT system interacts. We will use these two terms interchangeably.
- A set of *devices* form the leaves of the network. A node may include sensors and/or actuators, processors, and memory. Each node has a network interface. A node may or may not run the Internet Protocol.
- *Hubs* provide first-level connectivity between the nodes and the rest of the network. Hubs are typically run IP.
- *Fog processors* perform operations on local sets of nodes and hubs. Keeping some servers nearer the nodes reduces latency. However, fog devices may not have as much compute power as cloud servers. Fog devices also introduce system management issues.
- *Cloud servers* provide computational services for the IoT system. *Databases* store data and computational results. The cloud may provide a variety of services that mediate between nodes and users.

2.2 Protocols Concepts

Several protocols are used for data services in IoT systems.

Communication protocols may not provide sufficient abstraction for many applications. IoT systems need multi-hop, end-to-end communication. They also may exhibit complex relationships between data sources and sinks. Higher-level protocols can provide services that model more closely the needs of IoT systems. Given

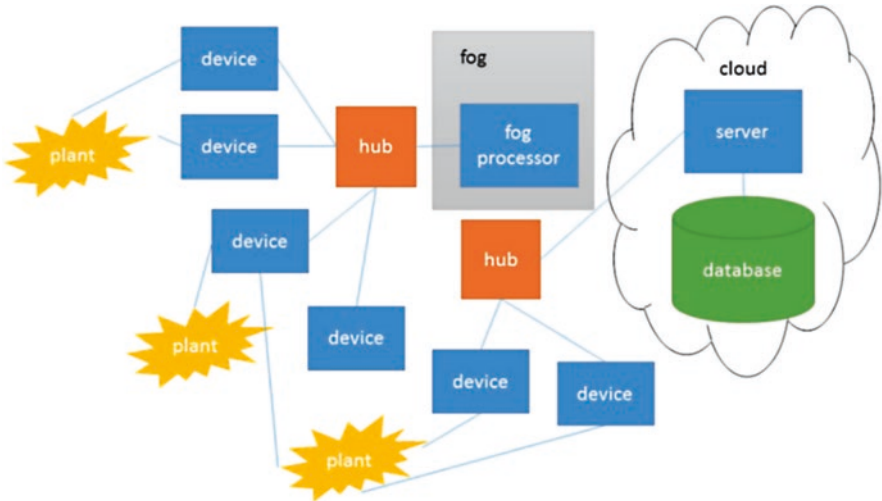


Fig. 2.1 Organization of an IoT system

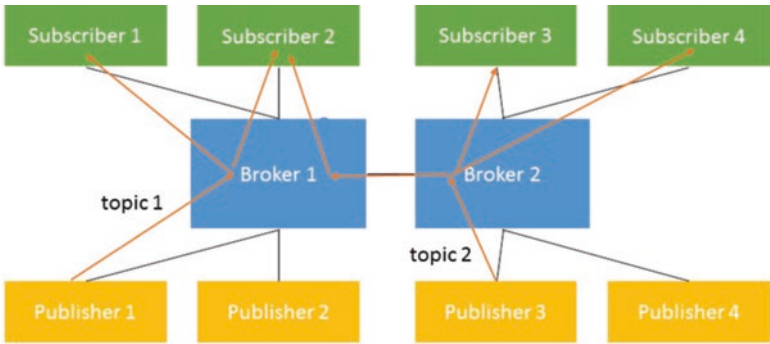


Fig. 2.2 The publish/subscribe model

the heterogeneous and long-lived nature of most IoT systems, standards are often used rather than custom protocols. Several different protocols have been proposed and, to varying degrees, used for IoT systems [Duf13]. The user space has not yet converged on a single standard for IoT communication services.

Given the prevalence of event-oriented models in IoT systems, a protocol should support event-style communication.

The HTTP protocol uses a request/response design pattern. A client issues a request for a hypertext object; the server then replies with the object in response.

A *publish/subscribe* protocol [Twi11] requires less coupling between the client and server as illustrated in Fig. 2.2. The server, known as a publisher, classifies messages into categories. Clients subscribe to the categories of interest to them. Publish/subscribe systems are typically mediated by *brokers* which receive published

messages from publishers and send them to subscribers. Messages may be organized by topic; all message of a given *topic* are distributed by the brokers to the subscribers for that topic. The broker knows the identities of subscribers but the publisher does not. Brokers may interact with each other using a bridge protocol. A *bridge* allows indirect publication of messages, with a message going from the publisher to a first broker, then to a second broker, and finally to subscribers who are not connected to the first broker.

Data Distribution Service (DDS) (<http://portals.omg.org/dds/>) [Obj16] is a publish/subscribe software architecture; several implementations of DDS are in use. A DDS *domain* maintains a logical *global data space*; the data is managed over a set of local stores. Publishers and subscribers are dynamically discovered across the network. Publishers can specify a number of quality of service parameters that are enforced by the brokers.

Real-Time Publish/Subscribe Protocol (RTPS) [Obj14] is a so-called wire protocol that defines a protocol for communication with DDS and other publish/subscribe systems. RTPS provides QoS properties, fault tolerance, and type safety.

Esposito et al. [Esp09] developed an architecture for time-sensitive publish/subscribe systems that would be scalable to Internet-sized systems. They identified three major design goals: predictable latency, guaranteed delivery in the presence of multiple faults, and continued performance under scaling. They identified several types of fault models for publish/subscribe systems: network anomalies (loss, ordering, corruption, delay, congestion, partitioning), link crash, node crash, and churn of nodes unexpectedly joining and leaving the system. Their architecture has three abstraction layers: the network layer consists of domains composed of nodes; the nodes layer consists of clusters, with each cluster's members belonging to the same stub domain; and a coordinators layer. The coordination layer routes messages using a tree-based topology built on top of a distributed hash table. The coordinator is p -redundant to provide fault-tolerant coordination. To provide fault-tolerant overlays, they formulate a model for path diversity that can be computed with limited knowledge of the network connections.

Kang et al. [Kan12] used a semantics-aware communication mechanism to reduce overhead and improve reliability. They use state-space estimators at both the publisher and subscriber to maintain continuity of sensor values in the presence of network variations. Their state estimator is of the form $x_{k+1} = F_{k+1}x_k$. The designer sets a model precision bound δ for each sensor. The bound is used to manage bandwidth requirements. Their system also dynamically adjusts the model precision bound.

Choi et al. [Choi16] combined DDS with the OpenFlow software-defined networking protocol to ensure that DDS can implement the QoS parameters. They added two QoS parameters that could not be easily deduced from the standard DDS parameters: MINIMUM_SEPARATION and an E2E_LATENCY specified by subscribers.

2.3 IoT-Oriented Protocols

We can divide protocols into two major categories: those that are tied to a specific physical layer and those that are not. Generally speaking, protocols that rely on a specific physical layer do not use the Internet Protocol, while protocols that are physical layer agnostic do use IP.

Zigbee [Zig14, Far08] is a mesh network designed for low-power operation. A variety of derivative application standards specialize the protocol for applications such as smart homes and utilities. Zigbee is based on the IEEE 802.15.4 PHY and MAC standards. 802.15.4 operates in three bands: 868 MHz, 915 MHz, and 2.4 GHz. It delivers bit rates from 20 to 250 kbps, depending on the frequency band. The Zigbee NWK layer sits on top of the 802.15.4 MAC layer and provides data and management services. The APL layer includes three sections: the application support sublayer, the Zigbee Device Objects layer, and the application framework.

Zigbee provides two types of network security models: a centralized security network can be started only by a Zigbee coordinator/trust center; distributed security networks do not have a central trust center. Nodes can join either type of network and adapt to the type of network they have joined. Networks are *formed* by either coordinators or routers after scanning to select an available channel. Coordinators form centralized security networks, while routers form distributed security networks. *Network steering* is the name for the process by which a node joins a network. After identifying an open network, the node associates with that network and receives a network key. Clusters define interfaces for features and domains.

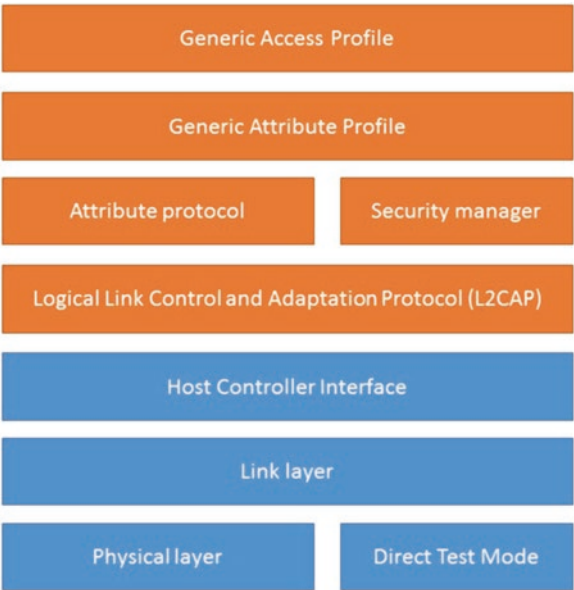
Bluetooth Low Energy (BLE) (<https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/low-energy>) [Hay13] is a part of the Bluetooth standard designed for low-power operation such as devices powered from coin cell batteries. A BLE device can work as a transmitter, receiver, or both. Figure 2.3 illustrates the *Bluetooth Classic* protocol stack.

The link layer provides an advertising service; devices can scan to identify nodes and networks. Devices can act as gateways to the Internet based on network address translation. The BLE protocol is stateful. BLE includes a number of optimizations to reduce power consumption.

LoRa (<http://loro-alliance.org>) [LoR15] is designed for wide-area IoT applications with a base station covering hundreds of square kilometers. It is designed to support a network topology with gateways for end devices, with gateways organized into their own star network. Data rates range from 0.3 to 50 kbps.

MQTT (<http://www.mqtt.org>) [IBM12, Oas14] is an IoT-oriented protocol with publish/subscribe semantics. The protocol is designed for low overhead and is agnostic to the data payload. MQTT provides three levels of quality of service: *at most once* provides best-effort service, *at least once* assures delivery but may incur duplicates, and *exactly once* ensures the message is delivered without duplication.

Fig. 2.3 The Bluetooth stack



MQTT is based on a publish/subscribe model. A message is given a retention attribute when it is published; messages with QoS designations of *at least once* or *exactly once* should set the retention flag. A new subscriber to the topic will receive the last publication on that topic.

When setting up a connection, a client can provide a *will* to the server to specify a message to be published if the client is unexpectedly disconnected.

Messages are classified using topic strings similar to hierarchical file names. The set of topics is organized into a topic tree. Topic names follow the names of the nodes in the topic tree path, with node names separated by “/”. Subscribers can use wildcards in the topic string: ‘+’ denotes a wildcard match at one level of the topic tree; “#” denotes a match at any number of levels of the topic tree.

XMPP (<http://xmpp.org>) is a protocol for streaming XML. It provides security, authentication, and information about network availability, and rosters of clients. XMPP-IoT (<http://xmpp-iot.org>) is a dialect of XMPP designed for IoT applications.

REST [Vaq14, Rod15] is widely used for Web services and has received some use as an IoT service model. REST is a design pattern for stateless HTTP transfers. It exposes directory-structured form resource indicators. REST can be used to transfer XML or JSON data. Clients access resources using GET, PUT, POST, and DELETE methods.

CoAP (<http://coap.technology>) [IET14] is a REST-based Web transfer protocol designed for IoT devices. It can be used with several types of data payloads, including XML and JSON.

Google Cloud Pub/Sub [Goo17A, Goo17B] can be used to provide publish/subscribe service to IoT and other systems. Topics and subscriptions are exposed as

REST collections. The system is divided into a data plane for messages and a control plane for allocation to servers known as *routers*; data plane servers are known as *forwarders*. The routers balance consistency and uniformity of data using a consistent hashing algorithm. A message life cycle includes several steps. When a publisher sends a message, it is written to storage. The subscribers receive the message, and the publisher receives an acknowledgment. Subscribers acknowledge the message to Google Cloud Pub/Sub. The message is deleted from storage once at least one subscriber for each subscription has acknowledged the message. The system monitors itself to detect and mitigate service problems.

Amazon Web Services (AWS) IoT [Bar15] is a managed cloud service for IoT devices, which are termed *things*. A *thing shadow* is a cloud model of a thing. A rule engine transforms messages based on rules and routes the results to AWS services. The *message broker* is based on MQTT. A *Thing Registry* assigns unique identity to things.

Microsoft Azure (<https://azure.microsoft.com/en-us/services/iot-hub/>) provides IoT-oriented services. Its Service Fabric is a middleware communication system that supports microservices running on a cluster. A microservice may be either stateless or stateful. It also provides a container model for applications; a *container* provides an isolated environment but relies on the operating system, in contrast to a virtual machine which runs underneath the operating system. It provides databases using both structured and unstructured approaches. It also provides APIs for artificial intelligence services.

2.4 Databases

Databases are used for both short-term and long-term storage. Applications may rely on databases to retrieve data over a time window for analysis. Some use cases may require archival storage of values.

Unstructured databases, known as *noSQL*, are used in many IoT systems. A noSQL database does not have a schema. Simple noSQL databases represent data as key-value pairs, but other representations are possible. The lack of a schema allows quick deployment but may cause maintenance problems.

The Amazon Simple Storage Service (Amazon S3) (<https://aws.amazon.com/s3/>) is an object store with a Web service interface. Data can be pushed to other, lower-cost storage services for long-term, infrequent use. Notifications can be issued when objects operated upon.

Google Cloud Storage (<https://cloud.google.com/storage>) is an object store for unstructured data. It provides three different service models at different latency/latency/price points. Cloud SQL can be used to perform database operations. Streaming transfers are supported using HTTP chunked transfer encoding.

Time-series data possesses structure that may require special handling to provide proper database performance. Time series are sometimes stored as blobs in relational databases to allow specialized algorithms.

Dynamic time warping (DTW) [Rat04, Rak12] is widely used to search over time-series data. DTW was originally used to compare waveforms for speech processing. Correlation provides a direct comparison of two waveforms. By warping one waveform, non-exact matches can be found. Dynamic programming can be used to find the minimum warp match between two-time series; a limit on maximum warping is typically applied to avoid obviously bad matches. Very efficient algorithms have been developed to provide high-speed search. Among other techniques, these algorithms abandon a warp computation early when partial results exceed a given bound. Fast DTW algorithms have been used to search very large databases.

2.5 Time Bases

Many IoT systems require a notion of global time. Several algorithms, starting with Lamport's algorithm [Lam78], have been developed for the synchronization of clocks in a distributed system.

The Network Time Protocol (RFC1305) is used on the Internet for distributed time synchronization.

2.6 Security

Security is a system property; the system can be only as secure as its weakest component. Security features are provided by components at several layers in the IoT stack: devices, physical networks, and middleware. A unified view of IoT system security architectures has not yet emerged.

Some, but not all processors for low-power operation, provide security features such as encryption accelerators and root of trust. The National Security Agency has developed families of lightweight block ciphers [Sch13]: SIMON targets hardware implementations, and SPECK is intended for software implementations. Gulcan et al. [Gul14] developed a low-power implementation of SIMON.

Several networks provide security features. Bluetooth Low Energy provides a Simple Secure Pairing protocol to protect against passive eavesdropping. It also provides address randomization. As discussed above, Zigbee provides two network security models: centralized and distributed. LoRa provides unique network keys, unique application keys, and device-specific keys.

MQTT does not specifically require encryption, but it can be used with several different security standards. *MQTT and the NIST Framework for Improving Critical Infrastructure Cybersecurity* [Oas14B] describe the relationship between MQTT and the NIST Cybersecurity Framework.

We will study IoT system security in more detail in Chap. 6.

References

- [Bar15] Barr, J. (2015, October 8). AWS IoT: Cloud services for connected devices. *AWS Blog*. <https://aws.amazon.com/blogs/aws/aws-iot-cloud-services-for-connected-devices/>
- [Choi16] Choi, H.-Y., King, A. L., & Lee, I. (2016). Making DDS really real-time with OpenFlow. *2016 international conference on embedded software (EMSOFT)* (pp. 1–10). Pittsburgh, PA.
- [Duf13] Duffy, P. (2013, April 30) Beyond MQTT: A Cisco view on IoT protocols. *Cisco Blogs*. <https://blogs.cisco.com/digital/beyond-mqtt-a-cisco-view-on-iot-protocols>
- [Esp09] Esposito, C., Cotroneo, D., & Gokhale, A.. 2009. Reliable publish/subscribe middleware for time-sensitive internet-scale applications. *Proceedings of the third ACM international conference on distributed event-based systems (DEBS'09)*. ACM, New York, Article 16, 12 pages.
- [Far08] Farahani, S. (2008). *Zigbee wireless networks and transceivers*. Amsterdam: Newnes.
- [Goo17A] Google. (2017, April 19). What is Google Cloud Pub/Sub? <https://cloud.google.com/pubsub/docs/overview>
- [Goo17B] Google. (2017, April 3). Google Cloud Pub/Sub: A Google-scale messaging service. <https://cloud.google.com/pubsub/architecture>
- [Gul14] Gulcan, E., Aysu, A., & Schaumont, P. (2015). A flexible and compact hardware architecture for the SIMON block cipher. In T. Eisenbarth & E. Öztürk (Eds.), *Lightweight cryptography for security and privacy. LightSec 2014, Lecture Notes in Computer Science* (Vol. 8898, pp. 34–50). Cham: Springer.
- [Hay13] Heydon, R. (2013). *Bluetooth low energy: The developer's handbook*. Prentice Hall: Upper Saddle River, NJ.
- [IBM12] IBM International Technical Support Organization (2012, September). *Building smarter planet solutions with MQTT and IBM WebSphere MQ telemetry*, Redbooks.
- [IET14] Internet Engineering Task Force (2014, June). *The constrained application protocol (CoAP)*, RFC 7252, Shelby, Z., Hartke, K., & Bormann, C.
- [Kan12] Kang, W., Kapitanova, K., & Son, S. H. (2012). RDDS: A real-time data distribution service for cyber-physical systems. *IEEE Transactions on Industrial Informatics*, 8(2), 393–405.
- [Lam78] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), 558–565.
- [LoR15] LoRa Alliance (2015, November). *LoRaWAN: What is it? A technical overview of LoRa and LoRaWAN*.
- [Oas14] Oasis. (2014, 29). *MQTT version 3.1.1*. Oasis standard.
- [Oas14B] Oasis (2014, May 28). *MQTT and the NISTG cybersecurity framework version 1.0*. Committee note 01.
- [Obj14] Object Management Group. (2014). *The real-time publish-subscribe protocol (RTPS) DDS interoperability wire protocol specification*, Version 2.2.
- [Obj16] Object Management Group. (2016). What is DDS? <http://portals.omg.org/dds/what-is-dds-3/>, accessed May 4, 2017.
- [Sch13] Schneier, B. SIMON and SPECK: New NSA encryption algorithms. *Schneier on Security*. https://www.schneier.com/blog/archives/2013/07/simon_and_speck.html, retrieved May 8, 2017.
- [Vaq14] Vaqqas, M. (2014, September 23) RESTful web services: A tutorial. *Dr. Dobb's*. <http://www.drdoobs.com/web-development/restful-web-services-a-tutorial/240169069>
- [Rak12] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., & Keogh, E.. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. *Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining (KDD'12)* (pp. 262–270). ACM, New York.
- [Rat04] Ratanamahatana, C. A., & Keogh, E. (2004, August 22–25). Everything you know about dynamic time warping is wrong. *Third workshop on mining temporal and sequential data*, in

- conjunction with the tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD-2004)*. Seattle, WA.
- [Rod15] Rodriguez, Alex. (2008, November 6). RESTful web services: The basics. IBM developerWorks, updated February 9, 2015. <https://www.ibm.com/developerworks/library/ws-restful/index.html>
- [Twi11] Twin Oaks Computing, Inc. (2011). What can DDS do for you?
- [Zig14] Zigbee Alliance (2014, December 2). ZigBee 3.0: The open, global standard for the Internet of Things. <http://www.zigbee.org/zigbee-for-developers/zigbee/>

Internet-of-Things (IoT) Systems

Architectures, Algorithms, Methodologies

Serpanos, D.; Wolf, M.C.

2018, XII, 95 p. 24 illus., 13 illus. in color., Hardcover

ISBN: 978-3-319-69714-7