Decentralized IoT with Wattsworth

John Donnal, Ryan McDowell, Michael Kutzer

***Abstract*—Low cost sensors, powerful single board computers, and high bandwidth wireless networks have encouraged the proliferation of IoT systems to both measure and control our environment. Unfortunately those eager to realize the benefits of IoT are generally forced to choose between centralized commercial offerings or developing their own system from scratch. The former is cost effective but brittle, while the later is flexible but often impractical. Wattsworth is an open source platform for decentralized IoT providing users with the best of both worlds. Users can leverage a turn-key software stack while still retaining total control of their systems with no dependency on third party infrastructure. This paper presents three case studies showing how Wattsworth solves traditionally challenging IoT scenarios. In particular, these case studies illustrate how to collect data from proprietary sensors, design complex user interfaces, and coordinate diverse geographically distributed devices. Source code, full documentation, and installation media for Wattsworth are available at `https://wattsworth.net`.**

I. Introduction

To date the majority of commercial IoT providers offer centralized solutions often referred to as Platform-as-a-Service (PaaS). In this model, users install brand specific hardware which connects back to the central provider who stores the data on behalf of their users [1]. Users then access their data through a website or mobile phone application. While convenient, PaaS systems expose users to a variety of security and privacy issues, encourage vendor lock-in, and are often difficult to customize beyond the expected use cases. But, most importantly they require constant connectivity to the service provider. If the provider's server is unavailable either due to a network interruption or a permanent termination of service, the client is left with a crippled IoT system. Depending on a third party to handle user data is unnecessary. A preferable model is a decentralized system where nodes connect to each other on demand and each is capable of collecting, processing, and storing data locally [2]. A decentralized system is not in contrast to the cloud. Centralizing data on the cloud can be convenient, but by making it a requirement, and in particular making it a requirement to use the vendor's cloud, it becomes a significant inconvenience. The user should be able to decide if a cloud connection is necessary and if so, where and how their data should be stored.

Wattsworth is a decentralized IoT platform that provides this flexibility. A formal discussion of the system architecture can be found in [3]. This paper focuses on the use of Wattsworth in otherwise challenging IoT scenarios. Any device capable of running Linux can be a Wattsworth node from single board computers (SBC's) to powerful cloud servers. All nodes run the same code. Because of this each node is self sufficient and does not necessarily require a network connection. While such standalone operation is possible, Wattsworth nodes are designed to connect to one another. The particular architecture is up to the user. By forming a mesh, data can move freely between nodes with no single point of failure. By connecting multiple nodes to a single centralized node users can build systems similar to PaaS IoT, but under their own control. Wattsworth is composed of two independent software stacks, Joule and Lumen. Joule runs the data pipeline and Lumen runs the user interface. The data pipeline is the core of Wattsworth. It is composed of one or more modules that work together to collect, process, and present sensor data. Data flows between modules through streams. These streams are transport agnostic meaning two modules may be connected locally on the same machine or remotely across a network with no change to the modules themselves. This provides the flexibility to place modules at the most efficient location on the network in terms of computation and storage resources.

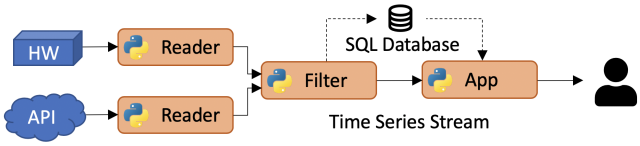

Fig. 1. Wattsworth is a decentralized IoT platform that uses a series interconnected of modules, to collect, process and present sensor data. Reader modules collect data from sensors, Filters process this data into actionable information, and Apps present this information to users through a web interface. The collection of modules is called a data pipeline.

II. Wattsworth Modules

Modules provide the core functionality of Wattsworth. They perform one of three tasks- generating data (Readers), processing data (Filters), or presenting data (Apps). While modules can be designed to implement more than one of these roles, restricting modules to a single task reduces complexity and facilitates code reuse. Figure 1 shows how the three module types combine to form a data pipeline. Reader modules interface with a sensor or an external Application Programming Interface (API) to collect new data for the pipeline. They have no input streams and one output stream. Filters process data collected by Readers. They have one or more input streams and may have one or more output streams. In addition to streams, Filters can also write outputs to a shared relational database. Depending on the type of information, it may be easier to represent as relational data rather than a streaming time series. Finally, Apps present the information produced by filters to end users through a web interface. Apps may have one or more input streams and no output streams. They can also access the relational database. Joule spawns modules and tracks their

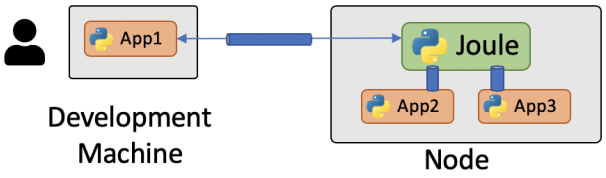

Fig. 2. During development, a module can run as a standalone executable on the developer's local machine while still connecting to the data pipeline on an active Wattsworth node. This allows developers to use more powerful software tools than are typically available on an IoT node.

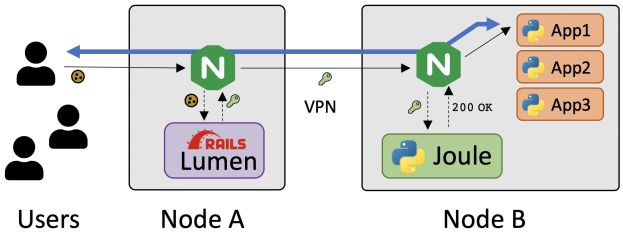

Fig. 3. In production, Wattsworth connects users to Apps through a series of reverse proxies. These proxies provide user authentication and transport layer security allowing developers to focus solely on the application logic rather than the supporting infrastructure typically required for production web servers.

execution, collecting logs and restarting any modules that fail. Because modules run as separate processes, restarting one module does not affect the rest of the pipeline. See [4] for more detailed information on module architecture and pipeline management. For continuity in the figures that follow modules are shown in orange, Joule instances in green and Lumen instances in purple. While every Wattsworth node is capable of running both Joule and Lumen it is not a requirement (see section IV).

A. Modules in Development

Writing software for IoT is particularly difficult because the target hardware often has limited computational resources and no traditional input/output devices such as a keyboard or screen. Because of this, Wattsworth provides a standalone execution environment for running a module on a development machine (eg a laptop or desktop) with its inputs and outputs connected to a live node called the target. This allows developers to use integrated development environments (IDE's) and visual debugging tools to inspect the execution of their module while still having it run with the same streams as it does in production. The particular structure of the standalone execution environment depends on the module type. Readers have no inputs and one output. When run in standalone mode they may either connect their output to the target or write it to stdout making it visible on the terminal. Using stdout facilitates debugging, and also makes reader modules useful outside of the Wattsworth environment because no target node is required. Standalone execution of Filters is more complex because they require multiple input and output streams as well as access to the target's relational database. Apps use the same execution environment as Filters and also run a local web server to host the user interface.

B. Modules in Production

Once a module is ready for production it can be deployed to one or more nodes by transferring the code and adding an associated configuration file to the Module Configuration directory (`/etc/joule/module_configs` by default). These configuration files are parsed on startup to create the data pipeline. Stream configuration files may be used to customize the data retention policies and specify the units, display types, and other aspects of stream elements. Complete details on these configuration files is available in the Wattsworth documentation and discussed in [4]. Streams connecting local modules are carried by inter-process communication (IPC) pipes and streams connecting remote modules are carried by secured HTTPS connections similar to the standalone environment discussed previously. Allowing remote users to connect to Apps requires a more complex communication architecture shown in fig. 3. Wattsworth is composed of two separate software stacks, Joule and Lumen. Joule controls module execution and Lumen provides a web interface to view stream data and interact with Apps. This diagram shows the communication path for a client using Lumen on Node A to access an App running on Node B. When a client interacts with an App, or any other aspect of Lumen, they provide a unique cookie as an authentication token. Requests destined for an App rather than Lumen itself are intercepted by Nginx which is an open source reverse proxy server [5]. Nginx sends the request header including the cookie to the local Lumen server running on node A. If the cookie is authentic, Lumen returns Nginx an API key for the associated Joule instance which may be local or running on a different node in the network. In this case the request is proxied to Node B where it is again intercepted by Nginx. Like on Node A, Nginx sends the header which includes both the unique App ID, and appended API key to the Joule daemon for verification. If the API key is valid, Joule responds with the UNIX socket path for the App. Nginx then finally delivers the request to the target App after modifying the headers and URL to make it appear as if the request has come directly from a local user. This makes the transition from development to production transparent. Apps operate identically in development and production. An additional benefit of this proxy structure is that it removes the need for apps to handle user authentication and transport layer security. This allows App designers to focus entirely on the business logic of presenting information to end users. The rest of this paper discusses how modules can be used to solve a variety of challenging IoT scenarios.

Fig. 4. The Ambient Weather WS-1900 Osprey weather station [7]. This system sends sensor data to `api.ambientweather.net` but it can be converted to send data to a user-controlled Wattsworth node instead. This removes the dependence on a third party for data collection and storage.

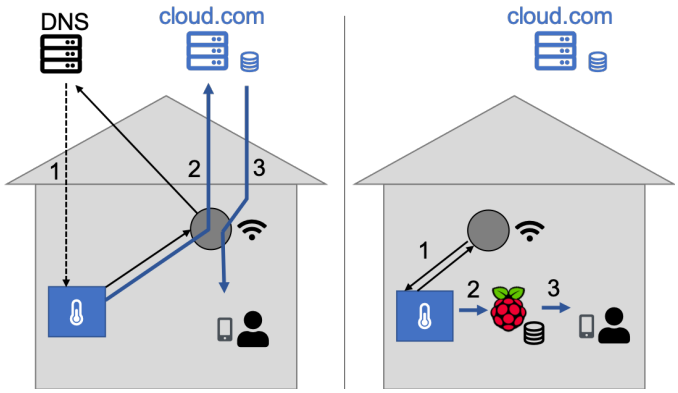

Fig. 5. The communication architecture for an IoT device like the Osprey (fig. 4) is shown on the left. By intercepting DNS requests (1) the data traffic (2) can be redirected to a local Wattsworth node as shown on the right. In both architectures, users interact with the data using a web application (3).

III. Using Proprietary Sensors

Despite the advantages of decentralized IoT, it is difficult to find devices that are not tied to a specific vendor's ecosystem [6]. This is particularly true in the cost sensitive commercial sector where vendors may artificially deflate the cost of hardware expecting to recoup this loss with data analytics and subscription fees. This business model does not encourage interoperability between platforms. Fortunately it is often possible to convert proprietary, vendor specific hardware, to a decentralized system like Wattsworth. This section describes the techniques required to perform such a conversion.

The Ambient Weather WS-1900 Osprey weather station (fig. 4) is typical of proprietary IoT hardware. Its communication architecture is shown on the left side of fig. 5. The system periodically transmits sensor data to a server run by the vendor (path 2) where it is stored and accessible by the user through a website (path 3). The right side fig. 5 shows the data path once the unit has been converted to communicate with a Wattsworth node instead. The details for a particular IoT device may vary but this general procedure will work unless the device has a hard coded certificate for the control server or a pre-shared encryption key both of which are uncommon.

1) Monitor Communication: The first step is to monitor the communication between the device and the control server. This can be done using a network analyzer like Wireshark. In order to intercept the network traffic the analyzer must either run upstream of the wireless access point or on a client with an adapter that can monitor wireless traffic in promiscuous mode. The Osprey weather station does not use encryption, but for devices that use secure sockets layer (SSL) or transport layer security (TLS) to encrypt their traffic, a Man-In-the-Middle (MitM) proxy like Fiddler [8] can be used recover the plain text data. Once the traffic is intercepted, it needs to be analyzed to determine the name of the control server and the end point on that server that is used by the IoT device. For example, the Osprey sends JavaScript Object Notation (JSON) encoded data to `api.ambientweather.net/endpoint`.

2) Redirect Traffic: To find this server, the Osprey like any other networked device, relies on the Domain Name System (DNS). DNS is a protocol that allows devices to convert domain names into an IP address by querying one or more resolvers. Path (1) on the left side of fig. 5 shows the typical DNS resolution path. The Osprey queries the DNS resolver on the local network, who forwards the query to its own resolver, usually an Internet Service Provider (ISP), who may forward it yet further until an authoritative resolver is found which responds with the requested IP address. However, the local DNS resolver can be reconfigured to respond to queries for `api.ambientweather.net` with the Wattsworth node's IP address rather than recursively calling its own resolver for the true address. This causes the Osprey to incorrectly route its data to the Wattsworth node instead of the cloud as shown in the right side of fig. 5. The final step is to capture this traffic with a Reader Module on the node.

3) Process Sensor Data: While the data is now flowing to the Wattsworth node it will be ignored by the operating system because `/endpoint` is not associated with a server process. To retrieve this data a reader module running on this node hosts a local web server. A reverse proxy like Nginx can be used to direct `/endpoint` to this local server and the reader module will then receives all of the Osprey's traffic. Once the reader unpacks the data and writes it to its output stream, the data can be be presented to the user through an App (path 3) providing the same user experience as the vendor's website. Of course to support remote data access at least one of the nodes in the Wattsworth mesh must be reachable outside of the local network (not shown in this figure). This is typically solved by running a node on a virtual private server (VPS) with a cloud provider like Azure or AWS and using a virtual private network (VPN) to connect back to nodes like this one which sit behind network address translation (NAT) boundaries. This type of architecture is discussed in section IV.

IV. Modular User Interfaces

In addition to the challenge of simply collecting data, effective IoT systems must present this data to users in a way that is relevant and actionable. Designing interfaces is time

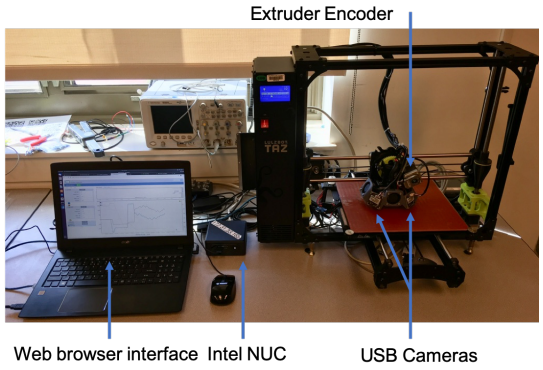

Fig. 6. A research platform for designing optical diagnostics for additive manufacturing (AM). Two USB cameras record the print head and five encoders track the motion of each stepper motor. Wattsworth is used to provide a visual interface to control printer operation and manage data collection.

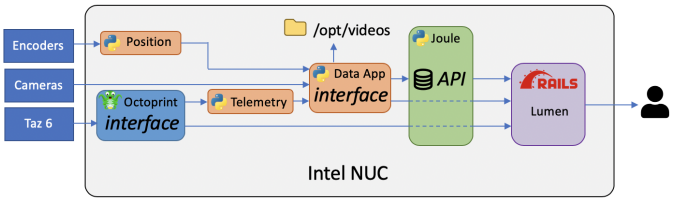

Fig. 7. The AM Research Platform data pipeline. Postion and Telemetry are reader modules that collect sensor data. Printer Control is an App that controls the USB cameras and labels data collected by the Readers. The Printer Control App and Octoprint are available through Lumen, the Wattsworth webserver.

consuming and can be particularly frustrating as users request different and perhaps conflicting features. Wattsworth Apps simplify interface development. By using Apps and Lumen's built-in visualization tools, developers can build custom interfaces from multiple components with a minimal amount of code. The following case study illustrates the benefits of this modular approach.

Figure 6 shows a research platform for developing new additive manufacturing (AM) diagnostics. A TAZ6 printer [9] is retrofitted with video cameras and high precision encoders that monitor each stepper motor. The system is operated by users who do not have an intimate knowledge of Linux or the command line so a visual UI was required. Designing an interface to control printer operation as well as manage the data collection is a complex task that would typically require significant software development. However, using Wattsworth, a modular UI combining a minimal amount of custom code with existing open source software was designed in only a few days.

The system is managed by an Intel NUC running both the Joule and Lumen components of Wattsworth as shown in fig. 7. Octoprint, an open source software application, is used to control the TAZ6. Octoprint hosts both a web based user interface and an application programming interface (API). The API allows other programs to receive telemetry data such as bed and extruder temperature. The Joule pipeline uses two Reader modules, the first collects data from the stepper

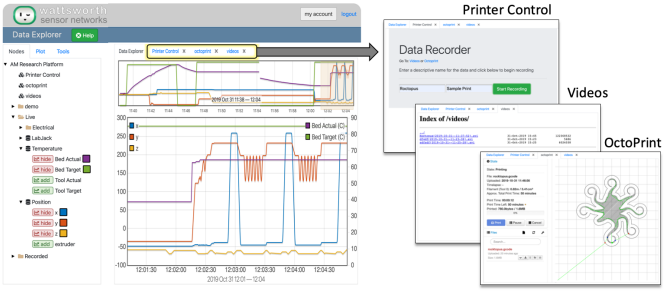

Fig. 8. The AM Research Platform user interface is powered by Lumen. The Data Explorer, currently selected, is a built-in interactive plotting tool. The Printer Control App, Octoprint, and a list of recorded videos are displayed in the other tabs as shown on the right.

motor encoders and the second collects telemetry data from Octoprint. These two streams are connected to an App module that controls the video cameras. The App provides a simple interface that allows a user to enter a name and description for an experiment and then begin collecting video and sensor data. Videos are stored in the local file system and sensor data is stored in the Joule timeseries database. Octoprint and the App module are combined by Lumen into the composite interface shown in fig. 8.

V. Building Distributed Systems

This final case study discusses a more complex system architecture that highlights Wattsworth's ability to coordinate data collection across diverse and geographically distributed devices. The test bench shown in fig. 9 consists of three separate nodes: a Raspberry Pi Zero, a Raspberry Pi 4, and an Amazon Web Services (AWS) virtual private server (VPS). These nodes are connected as shown in the block diagram in fig. 10. Directional arrows indicate master-follower relationships. Arrows point from followers to masters. Either Lumen or Joule may be a master of another Joule instance. Connecting Joule instances allows a pipeline to extend from the follower to the master. Connecting Joule to a Lumen instance makes its data streams and Apps available to users through the web interface. Relationships are authenticated with unique API keys and all network traffic between instances is encrypted with Transport Layer Security (TLS).

In this setup the Pi Zero runs a Reader module that collects GPS data. This data is stored locally on the Zero but is accessible by the Pi 4 because of the master-follower relationship. A reader module on the Pi 4 uses a data acquisition device (DAQ) to monitor two potentiometers. The red and green wires indicate their physical rotation. An App on the Pi 4 subscribes to both the *remote* GPS data and to the *local* potentiometer data to create the interface shown in fig. 11. Apps may be nested inside the Lumen interface as shown in fig. 8 or displayed in a separate window. In this case the 2.8" screen on the Pi 4 is displaying the App in a full screen window. This gives the appearance of a single-purpose embedded system but without requiring any platform specific code. Finally, a Wattsworth node running on the AWS

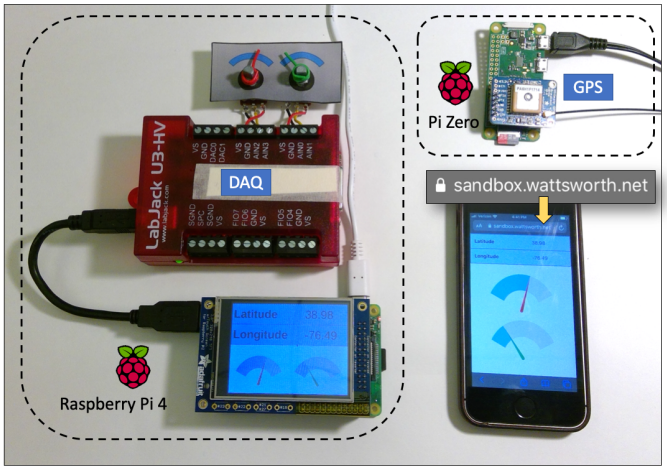

Fig. 9. Distributed IoT test bench. The Pi 4 combines GPS data from the Pi Zero with two potentiometers (red and green dials) to create a simple user interface. This interface is displayed on a 2.8" TFT screen and is also available at `https://sandbox.wattsworth.net`.

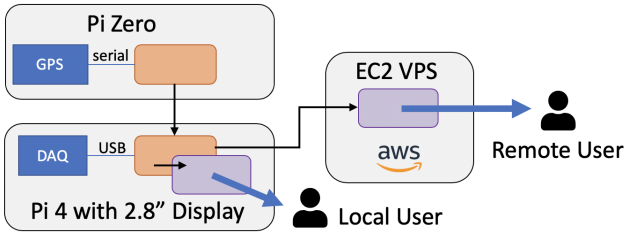

Fig. 10. The Wattsworth architecture for the test bench in fig. 9. The Pi Zero only runs Joule while the Pi 4 runs both Joule and Lumen. An App on the Pi 4 hosts an interface that is available through the local Lumen and through a remote Lumen running on an EC2 instance in AWS.

cloud allows remote users to view the same interface on a mobile phone. Cloud nodes like this one are often required for remote access due to Network Address Translation (NAT) boundaries which are common in commercial and residential networks. It is important to note that the App itself contains no logic for controlling 2.8" displays, network security, or user authentication. All of this is handled transparently by Wattsworth allowing the App designer to focus solely on the design of the interface itself.

VI. Future Work

While Wattsworth is fully operational as described, there are many areas for future work. Designing resilient pipelines that adapt to network partitions or module failures is one area of particular interest. Work by [10]–[12] on dynamic resource allocation in distributed computing environments could be adapted to provide automatic load balancing and dynamic module placement based on network conditions. Another area we hope to explore is alternative inter-module communication strategies. The stream abstraction is well suited to continuous time series data like that produced by sensors, but does not

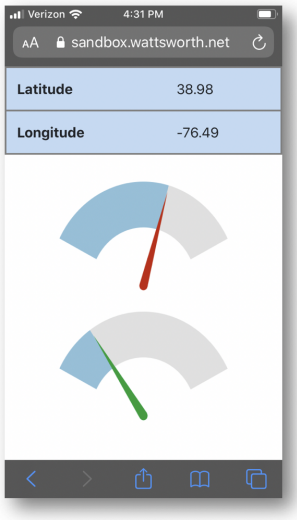

Fig. 11. Viewing the Raspberry Pi 4 Data App on an iPhone SE. Note the padlock icon which indicates the data is encrypted between the phone and the Lumen server at `sandbox.wattsworth.net`. Lumen proxies each request back to Joule on the Pi 4 which then proxies it to the App itself.

work as well for asynchronous information like user interaction or control signals. Alternative topologies like message passing [13], [14] and remote procedure calls [15] could enable exciting new applications in distributed control.

VII. Conclusion

Wattsworth is an open source platform for decentralized IoT systems. By providing code, documentation, and examples under a widely permissive license we hope to encourage research in decentralized IoT and to provide a viable alternative to the currently predominant Platform-as-a-Service IoT model. Critiques and contributions to the code base are encouraged.

Acknowledgment

This work was funded by the Office of Naval Research under Grant Number N0001419WX00587 and the Naval Sea Systems Command (NAVSEA) Engineering Directorate (05T).

References

[1] S. Yangui, P. Ravindran, O. Bibani, R. H. Glitho, N. Ben Hadj-Alouane, M. J. Morrow, and P. A. Polakos, "A platform as-a-service for hybrid cloud/fog environments," in *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2016, pp. 1–7.

[2] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Dec 2016.

[3] J. Donnal, "Wattsworth: An open source platform for decentralized sensor networks," *IEEE Internet of Things Journal*, pp. 1–1, 2019.

[4] ——, "Joule: A real-time framework for decentralized sensor networks," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3615–3623, Oct 2018.

[5] C. Nedelcu and M. Fjordvald, *Nginx HTTP Server*. Packt.

[6] B. Ahlgren, M. Hidell, and E. C. . Ngai, "Internet of things for smart cities: Interoperability and open data," *IEEE Internet Computing*, vol. 20, no. 6, pp. 52–56, Nov 2016.

[7] "Ws-1900 osprey solar powered weather station," *Ambient Weather https://www.ambientweather.com/amws1900.html*, accessed: 2019-11-1.

[8] E. Lawrence, *Debugging with Fiddler*, Austin, Texas, 5 2015.

[9] "Lulzbot taz 6," *https://www.lulzbot.com/store/printers/lulzbot-taz-6*, accessed: 2019-11-2.

[10] H. Zhang, Y. Xiao, S. Bu, D. Niyato, F. R. Yu, and Z. Han, "Computing resource allocation in three-tier iot fog networks: A joint optimization approach combining stackelberg game and matching," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1204–1215, Oct 2017.

[11] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for iot systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, Aug 2018.

[12] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu, "Resource allocation strategy in fog computing based on priced timed petri nets," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1216–1228, Oct 2017.

[13] L. Duan, C. Sun, Y. Zhang, W. Ni, and J. Chen, "A comprehensive security framework for publish/subscribe-based iot services communication," *IEEE Access*, vol. 7, pp. 25 989–26 001, 2019.

[14] L. Roffia, F. Morandi, J. Kiljander, A. D'Elia, F. Vergari, F. Viola, L. Bononi, and T. Salmon Cinotti, "A semantic publish-subscribe architecture for the internet of things," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1274–1296, Dec 2016.

[15] A. Sterz, L. Baumgärtner, R. Mogk, M. Mezini, and B. Freisleben, "Dtn-rpc: Remote procedure calls for disruption-tolerant networking," in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, June 2017, pp. 1–9.