Wattsworth: An Open-Source Platform for Decentralized Sensor Networks

John S. Donnal

***Abstract*—Internet of Things (IoT) sensor networks have the potential to deliver major benefits across a wide variety of industries, but designing the necessary data collection, processing, and presentation infrastructure can make them cost prohibitive. This has led to a Faustian bargain with centralized vendors who offer platform-as-a-service (PaaS) IoT solutions. Customers cede control of their sensors and all data they collect to a third party in exchange for convenient analytics. This type of perpetual subscription is a compelling revenue model that has stifled industry innovation in alternative architectures. This article presents Wattsworth, an open-source, decentralized alternative to PaaS-based IoT. Data collection, processing, and presentation are distributed across the network using a modular architecture. Lightweight "Data Apps" provide customizable user interfaces ensuring the information is relevant and actionable. Case studies of Wattsworth deployments across a variety of hardware platforms from single board computers to cloud-based servers demonstrate its utility in decentralized IoT applications. Full documentation, source code, and installation media are available online at https://wattsworth.net.**

***Index Terms*—Distributed information systems, edge computing, Internet of Things (IoT), open source software, sensor systems.**

I. Introduction

SENSOR networks are proliferating due to advances in microelectromechanical systems (MEMS), wireless networks, and low power embedded systems [1]. These technologies enable real-time telemetry of nearly any physical parameter. While low-cost hardware makes sensor networks possible, designing software to make them useful is still a significant challenge. To be effective, sensor networks must convert raw data into actionable information and present it to the right individuals at the right time. It is tempting to leverage existing centralized technology platforms for this task. Indeed, Google, Facebook, Amazon, and others excel in collecting a large amounts of data about their users and converting it into actionable (and monetizable) information. These platforms require centralization. Users trust Amazon precisely because it is centralized, and users rely on Google and Facebook to provide a centralized index of the Web and their friends, respectively. While there are examples of decentralized social media (Mastodon [2], Diaspora [3]), and e-commerce (Bazaar [4]), there is a little to recommend them in terms of efficiency and ease of use. Out of necessity, these major platforms are centrally controlled and as a result, the data they collect is centralized as well. Critically, centralized platforms enjoy a network effect where additional data increases the value of existing data. Internet of Things (IoT) vendors, eager to capitalize on a similar business model, offer centralized sensor networks often with branded hardware specific to their platform.

However, sensor data is geographically dispersed and high bandwidth both of which discourage centralization. Sensor data is generated at the network edge—edges that are often low bandwidth and unreliable. Requiring data to make a round trip to a cloud server and back to users who are often in close proximity to the sensors themselves creates a brittle system with a single point of failure. A preferable alternative is to distribute the computation across the network, processing the data where it is collected. The research community has long recognized the potential of such distributed computation [5]. Unfortunately, to date, most of the efforts have focused on distribution across well-connected nodes within the data centers with the goal of improving, the application resiliency or speed. For example, tools like Apache Storm and Kafka can be used to coordinate distributed computation but require reliable, high bandwidth connections between nodes [6], [7]. Current research into blockchains and smart contracts holds the promise of creating the decentralized data marketplaces [8], [9] but designing the fundamental infrastructure for collecting and utilizing the decentralized data remains an open problem.

II. IoT Network Architectures

A typical centralized IoT network is shown in Fig. 1. This model is offered by a variety of cloud vendors, including Amazon, IBM, and Microsoft [10]. Brand specific hardware is installed on the customer premise and establishes network connections to a central server run by the provider. Customers access their data through a Web interface hosted by the provider. This architecture is commonly referred to as platform-as-a-service (PaaS) because the customer pays for the platform and not a particular hardware or network configuration [11]. Because the customer is not involved with the hardware architecture the provider can improve efficiency and reduce costs by collocating the customer data in common databases and rely on application logic to ensure customers only have the access to their own data. This creates at least three problems. First, centralization leads to a single point of

Manuscript received June 26, 2019; revised August 6, 2019; accepted August 23, 2019. Date of publication October 11, 2019; date of current version January 10, 2020. This work was supported in part by the Office of Naval Research MVDC Risk Reduction Program under Grant N0001418WX01706, and in part by the Naval Sea Systems Command (NAVSEA) Engineering Directorate (05T).

The author is with the Weapons, Robotics, and Control Engineering, United States Naval Academy, Annapolis, MD 21402 USA (e-mail: donnal@usna.edu).

Digital Object Identifier 10.1109/JIOT.2019.2946853

U.S. Government work not protected by U.S. copyright.

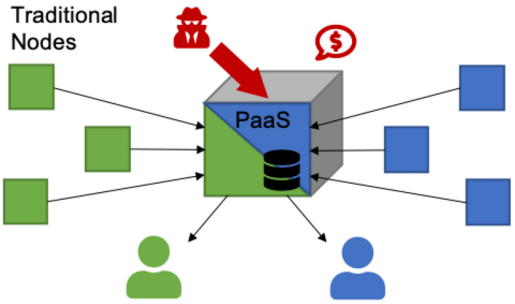

Fig. 1. In typical PaaS IoT deployments, users rely on a central server to manage their data. While initially convenient, this model leads to vendor lock-in and suffers from inherent privacy and security concerns.

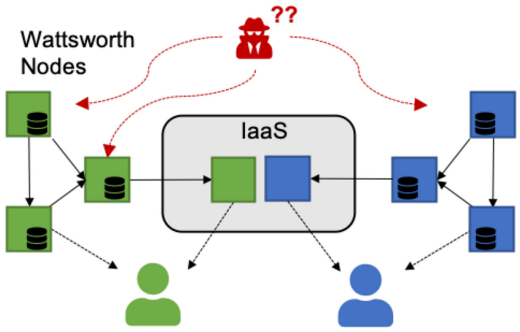

Fig. 2. Wattsworth is a decentralized alternative to PaaS-based IoT. Data and processing are distributed throughout the network on collaborative nodes.

failure. If the provider chooses to shut down the service or a customer terminates their subscription they lose access to their data and the sensor hardware itself may not be portable to another vendor. Second, it creates a high profile target for attackers. Any single customer's data may not be worth the cost to compromise a well-protected server, but aggregating the data of hundreds or thousands of customers increases the value of a breach often changing the calculus of an adversary. Numerous breaches to the widely trusted systems testify to the difficulty of securing the aggregated data sets [12]. Finally, it creates a conflict of interest in data confidentiality. The provider charges a subscription proportional to the value of the data as seen by the customer. Due to the network effect of centralization, however, the provider sees additional value beyond that available to the customer and has an incentive to realize this value through sales to third parties or in house analytics (customer acquisition, product development, etc.). These problems are not unique to a particular brand or company, they are intrinsic to the PaaS model itself.

A. Wattsworth Platform

The solution to this problem is to remove the need for third-party control. Wattsworth is a combination of two software projects, Joule and Lumen, that together provide a decentralized alternative to PaaS-based IoT. Joule collects data and Lumen presents it. Each node in the network can run one or both software stacks depending on the data pipeline and desired user access patterns (see Section V).

Joule, first presented in [13] as a stream processing engine, has been significantly expanded to support the modular data visualizations called Data Apps, an application programming interface (API) for creating dynamic data pipelines, and a new decentralized security model that ensures the confidentiality of data in transit. Lumen presents Data Apps to users though a dynamic Web platform that also provides tools for the decentralized data visualization and node management.

Fig. 2 illustrates the same sensor network run with Wattsworth as opposed to a PaaS provider. Each node stores the data locally and can present data directly to the users. Nodes can collaborate to share data, and if the processing or storage requirements exceed the capabilities of edge compute nodes, centralized nodes running exactly the same software stack can be added as required. Instead of a centrally managed platform, the cloud provider now hosts multiple virtual private servers (VPSs) that are wholly managed by the customer. This setup, called infrastructure as a service (IaaS), is significantly less susceptible to vendor lock-in because a VPS is a commodity that can be readily purchased from any cloud provider [14].

The goal of this article is to provide the sensor network community with a practical alternative to centrally hosted IoT models. To this end, Wattsworth is licensed as open-source software and installation media is available for common hardware platforms. This article discusses Wattsworth at the system level as illustrated in Fig. 3. Complete operational details, including full user-level documentation, are available online at `https://wattsworth.net`.

III. Joule

Joule is a decentralized platform for collecting, processing, and storing the sensor data. Modular blocks are connected by streams to form data pipelines ranging in complexity from a single Linux process to multiple networked nodes. Fig. 4 shows a logical pipeline on the top and its implementation across three different nodes below. A module can receive data from inside a composite module (input to 2), from another module on the same node (input to 3), or from another module on a different node (inputs to 5) with no code modification.

Pipelines are statically configured using INI format text files discussed in [13]. In addition to static pipelines a new API allows modules to request stream inputs and outputs at runtime. This enables dynamic data pipelines based on user input or network events.

A. Local Storage

Sensor data accumulate quickly. For example, three axis accelerometers used to detect faults in industrial machinery are typically sampled at 5 KHz or higher in order to capture diagnostic harmonics [15]. Assuming a 12-bit analog to digital converter (ADC) this results in well over 1 GB of data per sensor per day. The storage costs of this data are trivial with hard drives costing $0.03 per gigabit or less [16], but the network bandwidth to centralize this data is significant to the point of being impractical for many applications. The result is a hesitancy to use higher bandwidth sensors in IoT applications. Wattsworth eliminates the problem by storing all the data locally on the node where it is collected. As shown in Fig. 3, the open-source time-series database, TimescaleDB [17] operates in parallel with the data pipeline. Based on user configuration, some, all, or none of the streams

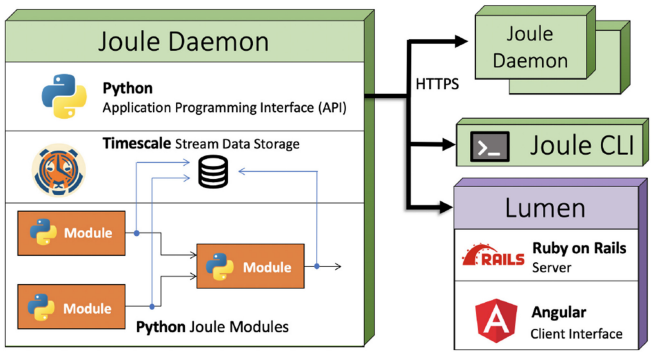

Fig. 3. Wattsworth is a combination of two software projects, Joule and Lumen. Joule manages data collection and Lumen manages data presentation. A secure API connects Joule nodes to each other, to Lumen, and to users through a CLI.

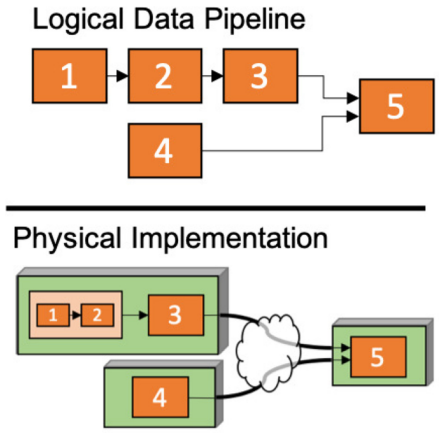

Fig. 4. Joule modules are connected by data streams to form pipelines (top). Modules are agnostic to the physical implementation of the pipeline which may span multiple processes and even machines (bottom).

are persisted and those that are persisted have rolling retention policies (e.g., "keep 1 week of data") so that local storage is not exhausted.

B. Data Pipes

Pipes are fundamental to data processing in Joule. Batch processing unbounded time series like realtime sensor data is difficult because the entire data set is never available. Even for historic data, resource-constrained systems like single-board computers do not have sufficient RAM to work with large data sets. Instead, data must be processed with streaming algorithms that work with sequential chunks of data. Joule pipes enable efficient chunked data processing. These pipes are conceptually similar to POSIX pipes commonly used in Linux [18]. Like POSIX pipes, Joule pipes decouple producers and consumers using a read/write abstraction and an internal buffer to compensate for different data access patterns with the assumption that the consumer can read at the producer's average data rate. Unlike POSIX pipes, read does not implicitly flush the buffer. This allows the consumer to retain a segment of data from the previous read. Padding chunks with previous data simplifies the logic for acausal filters and other algorithms that suffer from edge effects at chunk boundaries.

Joule pipes also provide transport layer abstraction making it easier to write topology independent streaming algorithms. Modules are agnostic to the sources of their inputs and the destinations of their outputs as well as their transport mechanism. This encourages code reuse and allows pipelines (such as the one shown in Fig. 4) to match the distribution of resources in the network.

Finally, Joule pipes include mechanics for handling the missing data. Data gaps can be produced intentionally such as stopping a sensor to save power, or unintentionally by a hardware failure or network partition. Joule pipes use the concept of data intervals to indicate such gaps. Consumer reads are aligned to interval boundaries. That is, all data returned by a read will be within a single interval. Ignoring the interval boundaries does not require any special handling, but consumers that need to track data gaps (for example to reset internal state) can check the pipe's `end_of_interval` attribute after each read. This flag will be True if the last read terminated at an interval boundary and False otherwise.

C. Communication

Joule nodes are connected through an API. The API allows modules and any other process to dynamically modify the pipeline by creating new streams or subscribing to streams produced by other modules. This allows for ad hoc data processing models, for example filtering a time range of data based on user request.

A command-line interface (CLI) wrapper around the API allows users to interact with Joule nodes from the terminal. This interface is self-documenting and provides complete access to the streams and modules of any connected node. This type of remote access and dynamic introspection requires a robust security model which is discussed below.

D. Security

By default, a Joule node can only be controlled locally and only by the administrator (root) or a user with root privileges. Static text files in the `/etc/joule` directory specify module and stream configurations. These files are owned by root and readable only by Joule. Without further configuration the API is inaccessible, and the node cannot be modified at runtime or accessed remotely.

API access for a local user can be enabled with the CLI command:

```
$> sudo joule admin authorize
```

The left side of Fig. 5 illustrates its execution. This command bypasses the API and directly reads `/etc/joule/main.conf` meaning it requires root privileges to execute as indicated by the use of `sudo`. Using the credentials from this file (1), it creates a master record for the current user in the database (2). Users are authenticated by 256-bit keys represented as ASCII strings. A valid key must be passed in the header of every API request. In order to simplify CLI usage keys are stored by hostname in a JSON text file in the user's home directory (3). When multiple nodes are available, users can then specify the target of a particular command with the `-n` flag.

```
$> joule -n bar stream list
-- connecting to node [bar] --
├── Weather
│   ├── Indoor
│   │   ├── Temperature
│   │   └── Humidity
│   └── Outdoor
│       ├── Temperature
│       └── Humidity
└── Power
    ├── Voltage
    └── Current
```

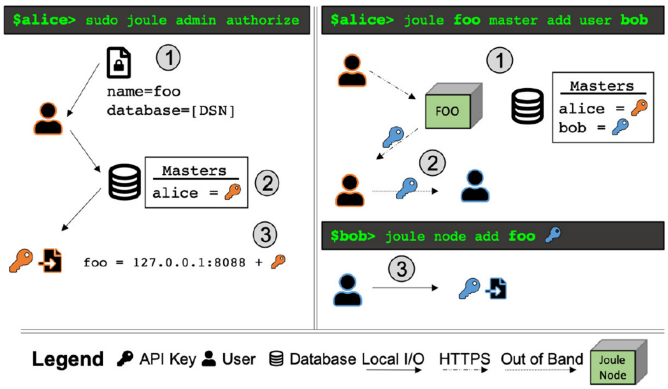

Fig. 5. Managing access to Joule nodes. The `admin authorize` command grants API access to local administrators (left side). Once authorized, users can grant access to others using the `master add` command (right side). The operation of these commands is discussed in Section III-D.

Omitting this flag uses the default node which can be configured by the user.

The API is encrypted with transport layer security (TLS). By default Joule creates a self-signed 2048-bit RSA key pair on initial startup. While this is sufficient for data confidentiality it leaves the network vulnerable to man-in-the-middle attacks. Therefore, it is preferable to sign the keys with a certificate authority. The certificate authority does not need to be globally trusted and many open-source tools make it easy to set up a public key infrastructure (PKI) within an organization. All traffic over the API is encrypted with perfect forward secrecy meaning even if an adversary obtains key material they are unable to decrypt any previously recorded traffic.

Once authorized themselves, users can grant others API access to a node with the command:

```
$> joule master add user <name>
```

The right side of Fig. 5 illustrates its execution. This command sends a POST request to the `/user` endpoint authenticated with the requesting user's API key (1). The node responds with a new key which is displayed on the terminal (2). The requesting user then transfers this key out of band, for example by e-mail, to the new user who can add it to their JSON configuration file for future use (3).

Finally, and most importantly nodes can be connected to each other with the command:

```
$> joule master add joule <hostname>
```

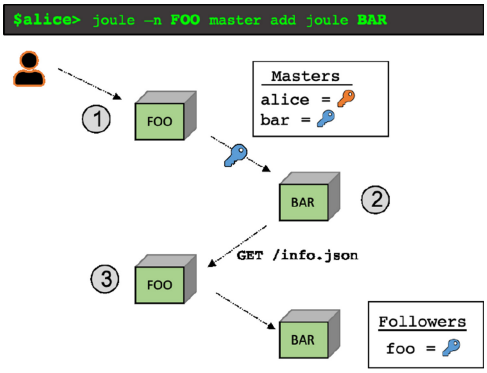

Fig. 6. Connecting Joule nodes. An authorized user can connect nodes in a master–follower relationship. This allows data pipelines to extend from the follower to the master (in this case FOO to BAR). The relationship can be made reciprocal by repeating the command with the nodes reversed assuming the user is a master of BAR as well as FOO.

Fig. 6 illustrates the execution sequence for two nodes, FOO and BAR. An authorized user sends a POST request authenticated with their API key to the `/user` endpoint on FOO (1). FOO then generates a new key which it sends with a POST request to the `/follower` endpoint on BAR (2). The `/follower` endpoint is by necessity unauthenticated. BAR confirms that the key is valid and that FOO is accessible by sending a GET request to FOO's `/info` endpoint (3). This step is not required for security but is helpful to detect NAT and firewall configurations that prevent bidirectional traffic between nodes. Once BAR confirms that it has access to FOO the process is complete, and the user is notified about the new relationship. Data pipelines can now be extended from FOO to BAR. The relationship can be made reciprocal by running the command again with the nodes reversed.

IV. Lumen

Lumen is a user interface for Joule. Joule processes data into actionable information and Lumen present it to the end users through the modular interfaces called Data Apps. Lumen also provides stream configuration tools and a visualization interface that can simultaneously plot data from multiple nodes on a single interactive display.

Unlike centralized sensor networks which rely on a single visualization server (or cluster of servers) running in a data center, Lumen can run directly on edge nodes in parallel with Joule. This allows users to access data from any point in the network. It also improves reliability. If the network is unavailable a standalone node is self-sufficient, or in the less extreme event of a network, partition users can still access data from locally reachable nodes.

Lumen is a single-page Web application (SPA) that emulates the look and feel of a native desktop application in a Web browser. Using a browser-based interface provides two major benefits. First, it allows cross-platform remote access. Second, it off loads computation to the user's machine allowing even the most resource-constrained nodes to provide low latency and responsive user interface.

Even though Lumen and Joule can run in parallel on the same node, they do not communicate by default. An authorized

user must explicitly grant a Lumen instance access to a Joule node with the command:

```
$> joule master add lumen <hostname>
```

The execution sequence is similar to the Joule–Joule pairing shown in Fig. 6 but with an additional authentication step that associates the Joule node to a particular Lumen user who becomes the node's "owner." This is required because in Lumen, unlike Joule, authentication and authorization are not identical. In Joule, any authenticated request (via API key) is also authorized. In Lumen, authorization to view a particular node must be explicitly granted by its owner.

A. Data Stream Visualization

Lumen provides a dynamic time series visualization tool that allows users to view large data sets using the minimal network bandwidth (Fig. 7). Similar to online maps, the full data set is never sent to the client browser, instead Lumen requests down-sampled data from the Joule node with a total size less than or equal to the number of pixels in the browser window. As the user zooms in on a smaller time range the resolution of the data is increased proportionally until the range is small enough that the raw data matches the user's resolution and downsampling is not necessary. Joule nodes maintain a cache of downsampled data to improve query speed. By default, Joule streams are iteratively decimated by a factor of four leading to a total storage cost roughly twice that of the raw data alone [19]. Lumen can retrieve data from multiple Joule nodes simultaneously which means a single plot may contain data from many different physical locations. This makes Lumen feel similar to centralized platforms and facilitates data exploration.

B. Data Apps

Viewing raw time series data can reveal important trends and is a fundamental first step in developing data processing algorithms, but simple line graphs are unsatisfactory to the majority of end users. One of the primary difficulties in deploying effective sensor networks is providing actionable information to a wide variety of stake holders. For example, in an industrial setting equipment operators, maintenance technicians, and plant managers will all likely request different types of information and levels of detail. Not only will they disagree on data presentation, their requests may change over time as well. This is a particularly challenging problem in the centralized systems because they tend to be tightly coupled. Even a small change on a centrally hosted dashboard usually requires to write and deploy privileges on a large portion of the codebase which is a permission generally held by few if any of the end users. Centralized interfaces also require continuous network connectivity. Data Apps are designed to avoid these problems.

A Data App is a Joule module that provides a Web interface. Any module can become an app by mapping URL endpoints to functions that return HTML. This design pattern matches other popular Python Web frameworks. Lumen integrates Data Apps from multiple Joule nodes providing a composite interface as shown in Fig. 8. Because Data Apps are entirely hosted by a module, the code base is constrained to the relevant

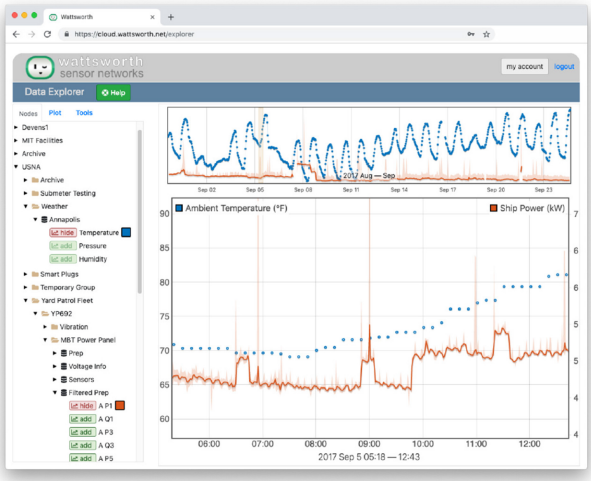

Fig. 7. Lumen's interactive plotting tool can display data streams from multiple Joule nodes simultaneously. This allows users to easily explore decentralized data using minimal network bandwidth.

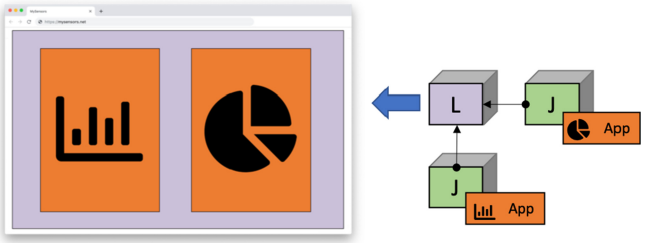

Fig. 8. Data Apps are user interfaces hosted by Joule modules. Apps are proxied through Joule's API to Lumen which integrates them along with native content to form a composite interface. The result is visually indistinguishable from a centrally hosted Web page.

presentation logic and any errors only affect the app itself. This reduces development complexity and risks in making it possible for end users to design and modify apps themselves. Additionally, apps eliminate the dependency on a single centralized server because they can be presented by any Lumen instance associated with the node. Indeed, assuming both Joule and Lumen are running on the same node, users can access local apps directly without any need for network connectivity.

When run as a standalone process, Data Apps host their Web interface on a local TCP port similar to other Web frameworks. This facilitates debugging and code development. Once an app is ready for production the module is simply added to the Joule pipeline. Modules automatically detect the Joule execution environment and serve their interface over a Unix socket that is directly connected to the Joule daemon. Joule then acts as a proxy allowing access to the app through its API. This provides apps with both data confidentiality and authentication. Because the Joule API uses TLS, the app is automatically secured by HTTPS and authenticated by API key. Lumen instances associated with the Joule host provide another layer of proxy support allowing end users to interact with the module directly from within the Lumen interface.

C. Security

Without careful consideration, composing a webpage from multiple sources lead to cross-site scripting (XSS)

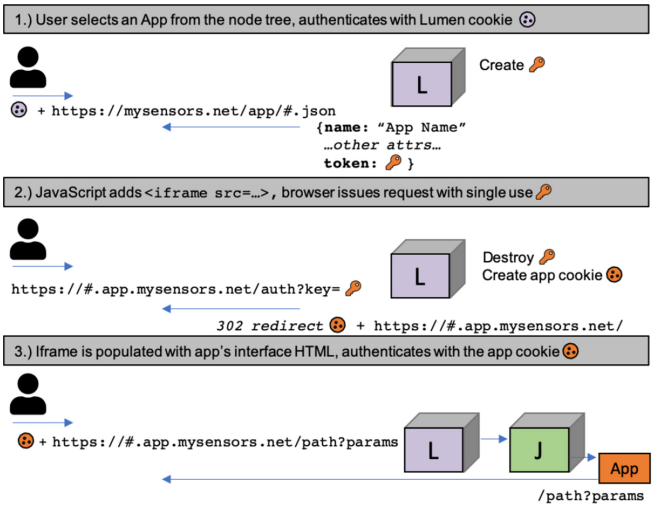

Fig. 9. Security infrastructure for composite interfaces in Lumen. Apps are hosted on dynamic subdomains in order to segregate authentication credentials.

vulnerabilities [20]. Module authors have complete control over a Data App including the ability to add JavaScript that executes in the user's browser. Simply combining multiple Data Apps along with other Lumen content on a shared page allows malicious scripts in one App to extract information from other Apps and even worse issue commands to the Lumen server itself without the user's knowledge or consent. The fundamental problem is the browser security model which sandboxes content by domain name allowing all scripts on a page to act with the same level of authorization.

To understand the solution to this problem, a brief overview of the browser authentication model is helpful. Single-page applications like Lumen uses background HTTP requests to provide dynamic content without reloading the page. This technique is called AJAX. In order to prevent the user from having to authorize each of these HTTP transactions with a password, AJAX requests are authenticated with cookies. Cookies are cryptographic tokens stored by the browser that are automatically attached to AJAX requests and cannot be viewed or modified by JavaScript running on the page. The Web server issues the cookie to the browser when a user logs into the site and removes it they log out. The security problem arises because any AJAX request on the page whether from a Data App or from the Lumen frontend code itself is tagged with the same cookie. This means the Lumen server cannot differentiate one source of requests from another.

Cookies are managed per-domain so the solution is to host each Data App on a different domain. For example, a Lumen instance at:

```
https://mysensors.net
```

serves each app on a unique subdomain with the form:

```
https://<app_id>.app.mysensors.net
```

These subdomains are rendered in sandboxed iframes that are visually indistinguishable from native page content (see example in Fig. 11).

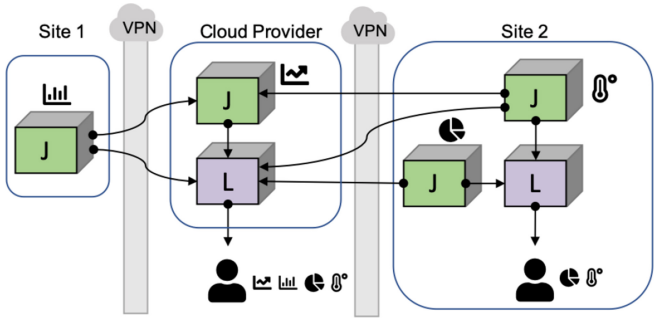

Fig. 10. In multinode deployments, Joule and Lumen typically run on separate systems. This reduces the computation requirement on edge nodes and simplifies user management. Icons by Joule nodes indicate Data Apps and icons by users indicate the Lumen instances through which they are available.

While segregating apps into subdomains removes the XSS vulnerability it introduces an authentication problem because each app effectively becomes a standalone website. To avoid the requiring users to log into each app separately, Lumen performs the authentication handoff illustrated in Fig. 9. This provides each App a unique subdomain cookie without leaking the user's Lumen username/password credentials. When a user opens an App, the Lumen frontend issues a GET request to retrieve a temporary authorization token for the App (1), once this token has been retrieved an iframe is added to the page pointed to the `/auth` endpoint of the app's subdomain with the token as a query parameter (2), e.g.,

```
<iframe src="https://31.app.
            mysensors.net/auth?token=XXXXX">
```

The `/auth` endpoint is intercepted by Lumen which checks the token's validity and generates a new cookie specific to the app. It returns this cookie to the browser and redirects the iframe to the `/` endpoint of the subdomain which is the index page of the data app itself (3). Future AJAX requests generated by the Data App include this new cookie for authentication.

V. Implementation

Wattsworth is designed for flexibility. A data pipeline can span multiple geographically dispersed nodes or run entirely on a single board computer. This section introduces typical deployment configurations for both ends of the spectrum. These should be viewed as suggestions rather than requirements—users are encouraged to create Joule and Lumen configurations that suit their particular data environment.

A. Multinode Systems

A typical configuration for a multisite installation is shown in Fig. 10. Directed links indicate connections from followers to masters as discussed in Section III-D. Joule nodes at each site host locally relevant Data Apps-based off their own data streams while a cloud node synthesizes each of these streams into a more complex App that provides an overall view of the system. Users access these Apps through Lumen gateways. While nodes are capable of running both

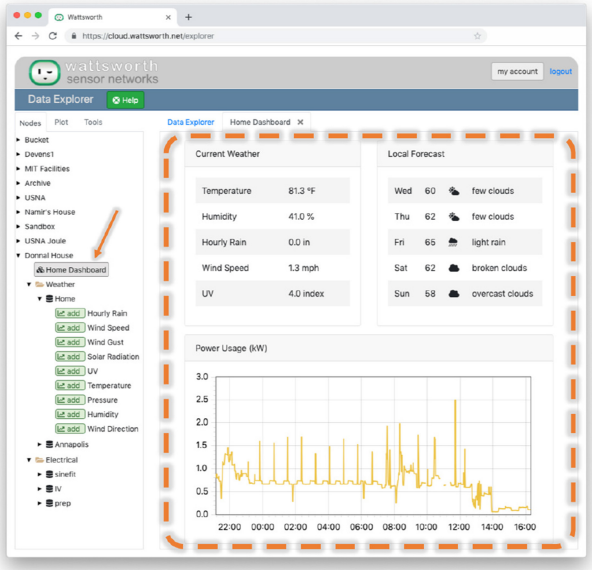

Fig. 11. Data Apps are launched from the node navigation panel (arrow) and displayed alongside native page content (boxed). This particular app displays weather and power usage from sensors in a single family home. The app is hosted by a Joule module running on a Raspberry Pi inside the house, and the Lumen instance is running on a VPS in Amazon's AWS cloud.

Joule and Lumen simultaneously, using relatively few Lumen instances reduces computation overhead and simplifies user management. Disabling Lumen on the resource-constrained edge nodes conserves computation for data processing rather than Web hosting. Additionally, since user accounts are local to each Lumen node, consolidating access to a few gateway nodes reduces the complexity of user management. Of course in the event of a network outage, Lumen can be re-enabled on any node allowing access to its local Data Apps.

As an example, Fig. 11 shows Lumen presenting a home dashboard Data App that displays local weather information and electricity usage. The visualizations are produced by the data collected from a backyard weather station, a high-resolution power meter, and a free forecast API. The data pipeline consists of five modules running on a Raspberry Pi 3 inside the home. Remote access is provided by a Lumen gateway running on a VPS on Amazon's cloud. Cloud-based Lumen instances such as this one, do not require significant resources. This particular Lumen instance controls access to over 5TB of data distributed across the nodes listed in the left-hand panel of Fig. 11 and is easily hosted on a free-tier VPS (t2.micro) in Amazon's AWS cloud. Migrating to another provider or hosting it locally is simply a matter of configuration.

B. Standalone Systems

Wattsworth nodes can also function as standalone systems. In scenarios where network connectivity is unreliable or not available a standalone node can run both Joule and Lumen in parallel. That is, a single node can both collect data and host a user interface to view and interact with the data without any network dependency. This is particularly useful for extended data collection in austere environments. With the low cost of persistent storage, high bandwidth data sets are not a problem.

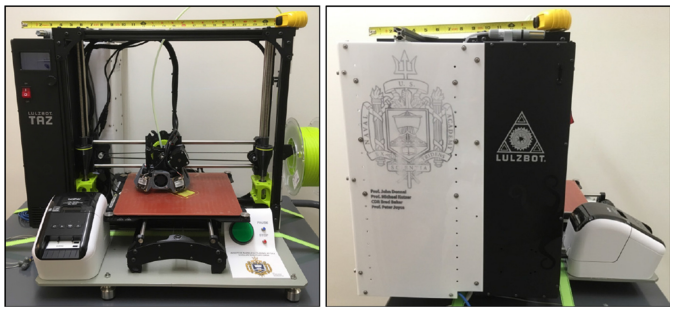

Fig. 12. AM at sea (AMAS) platform. A commercial 3-D printer is retrofitted with multiple sensors and a simplified user interface to evaluate the use of AM technology on naval platforms. The standalone system is controlled by a single embedded computer running both Joule and Lumen.

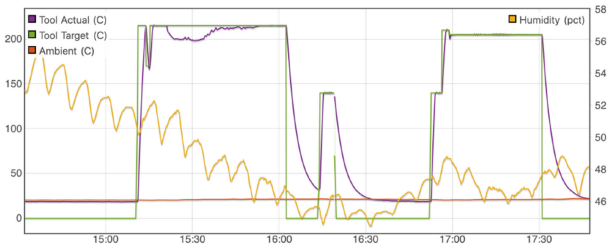

Fig. 13. Lumen plot of telemetry and sensor data during a series of underway prints aboard a Yard Patrol vessel-based out of USNA. The high humidity and poorly controlled tool temperature (particularly around 15:30) can lead to equipment failure and reduce print quality.

Local operators can confirm that the sensors are functioning properly by checking the data collection in Lumen. The Live Update mode, designed particularly for this use case, continually updates the display with the most recent data. When collection is complete, data can be transferred out of band to an aggregating node for processing in a pipeline or exported to another analysis tool like MATLAB or Excel. Of course without any network connectivity, the node must have a display (e.g., be a laptop or desktop) for users to interact with Lumen.

As an example, the 3-D printer in Fig. 12 uses a suite of sensors managed by Wattsworth to evaluate the performance of the additive manufacturing (AM) technology on Navy ships. When underway, printers such as this one suffers reliability issues caused by a range of potential factors, including induced vibration from nearby machinery, loosely controlled temperature and humidity, poor power quality, and high sea state. This platform is a tool for quantifying the impact of these factors in an effort to develop the mitigation techniques and provide guidance to equipment operators. The printer is instrumented with two three-axis accelerometers that measure vibration of both the ship and print bed at 1 KHz during prints and 100 Hz otherwise. An environmental sensor measures ambient temperature and humidity. Two cameras monitor the extruder head recording XGA video (1024×768) at 15 fps during print operation. Finally, telemetry from the printer itself provides both commanded and actual extruder and bed temperature at 1-Hz intervals. All of this data is collected by Joule modules and stored locally on a 1-TB drive.

A local instance of Lumen is available on a standalone WiFi network hosted by the node allowing the ship crew to monitor the data collection using a laptop or mobile device. Fig. 13

shows a Lumen plot of environmental conditions during a series of prints while underway. Writing a custom software stack to operate the sensors and reliably store the large volume of data produced would be both time consuming and error-prone. By using Wattsworth, the system was quickly prototyped using a minimum amount of custom code and installed on a 116 foot Yard Patrol vessel at the U.S. Naval Academy where it currently collects data during cruises in the Chesapeake Bay and the Atlantic Ocean. Additionally, due to the modular architecture, all of the code developed for this system can be used without modification by anyone else interested in collecting data from similar sensors.

VI. Conclusion

Low-cost sensors, high-speed wireless networks, and powerful embedded platforms enable continuous telemetry of the physical environment at an unprecedented scale. However, turning the deluge of data into actionable information remains a significant challenge. Centralized IoT models limit the utility of sensor networks and introduce significant privacy and security concerns. Wattsworth is an open-source decentralized IoT framework that allows users to adapt data processing to match their information needs and hardware resources improving both the reliability and utility of sensor networks.

A. Future Work

Joule currently supports programmatic control of data pipelines through its API. Research into dynamic resource allocation in distributed computing environments by [21]–[23] could be used to implement automatic load balancing and module placement in a multinode network. Additionally, frameworks similar to [24] and [25] could provide bi-directional message passing between modules. This would enable closed-loop feedback useful for autonomous control. All source code is provided under an open license to encourage this and other research into effective decentralized IoT architectures.

Acknowledgment

The author would like to thank Prof. M. Kutzer for the mechanical design of the Additive Manufacturing at Sea platform, and the U.S. Naval Academy Waterfront Readiness Department for supporting sea trials on the Yard Patrol fleet. It is approved for public release with unlimited distribution (DCN 43-5756-19).

References

[1] B. Ó. hAnnaidh *et al.*, "Devices and sensors applicable to 5G system implementations," in *IEEE MTT-S Int. Microw. Symp. Dig. Workshop Series 5G Hardw. Syst. Technol. (IMWS-5G)*, Dublin, Ireland, Aug. 2018, pp. 1–3.

[2] B. Nystedt, *Tired of Twitter? Join Me on Mastodon*, Wired, San Francisco, CA, USA, Jul. 2018. [Online]. Available: https://www.wired.com/story/join-mastodon-twitter-alternative/

[3] A. Bielenberg, L. Helm, A. Gentilucci, D. Stefanescu, and H. Zhang, "The growth of diaspora—A decentralized online social network in the wild," in *Proc. IEEE INFOCOM Workshops*, Orlando, FL, USA, Mar. 2012, pp. 13–18.

[4] A. Greenberg, *The Fed-Proof Online Market Openbazaar Is Going Anonymous*, Wired, San Francisco, CA, USA, Jun. 2017. [Online]. Available: https://www.wired.com/2017/03/fed-proof-online-market-openbazaar-going-anonymous/

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

[6] J. S. V. D. Veen, B. V. D. Waaij, E. Lazovik, W. Wijbrandi, and R. J. Meijer, "Dynamically scaling apache storm for the analysis of streaming data," in *Proc. IEEE 1st Int. Conf. Big Data Comput. Service Appl.*, Redwood City, CA, USA, Mar./Apr. 2015, pp. 154–161.

[7] M. Sewak and S. Singh, "IoT and distributed machine learning powered optimal state recommender solution," in *Proc. Int. Conf. Internet Things Appl. (IOTA)*, Pune, India, Jan. 2016, pp. 101–106.

[8] A. Suliman, Z. Husain, M. Abououf, M. Alblooshi, and K. Salah, "Monetization of IoT data using smart contracts," *IET Netw.*, vol. 8, no. 1, pp. 32–37, 2019.

[9] G. S. Ramachandran, R. Radhakrishnan, and B. Krishnamachari, "Towards a decentralized data marketplace for smart cities," in *Proc. IEEE Int. Smart Cities Conf. (ISC2)*, Kansas City, MO, USA, Sep. 2018, pp. 1–8.

[10] P. Miller and M. Pelino, *The Forester Wave^{TM}: Industrial IoT Software Platforms, Q3 2018*, Forrester, Cambridge, MA, USA, Aug. 2018.

[11] S. Yangui *et al.*, "A platform as-a-service for hybrid cloud/fog environments," in *Proc. IEEE Int. Symp. Local Metropolitan Area Netw. (LANMAN)*, Rome, Italy, Jun. 2016, pp. 1–7.

[12] C. Mele, *Data Breaches Keep Happening. So Why Don't You Do Something*, New York Times, New York, NY, USA, Aug. 2018.

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

[14] B. Suleiman, "Elasticity economics of cloud-based applications," in *Proc. IEEE 9th Int. Conf. Services Comput.*, Honolulu, HI, USA, Jun. 2012, pp. 694–695.

[15] C. Murphy, *Choosing the Most Suitable MEMS Accelerometer for Your Application*, Analog Devices, Norwood, MA, USA, Nov. 2017.

[16] A. Klein, *Hard Drive Cost Per Gigabyte*, BackBlaze, San Mateo, CA, USA, Jul 2017. [Online]. Available: https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/

[17] *Timescaledb*. Accessed: Jun. 5, 2019. [Online]. Available: https://www.timescale.com/

[18] pipe(7). Sep. 2017. *Linux Programmer's Manual (Man-Page), Release 5.01*. [Online]. Available: http://man7.org/linux/man-pages/man7/pipe.7.html

[19] J. Paris, J. S. Donnal, and S. B. Leeb, "NilmDB: The non-intrusive load monitor database," *IEEE Trans. Smart Grid*, vol. 5, no. 5, pp. 2459–2467, Sep. 2014.

[20] L. K. Shar and H. B. K. Tan, "Defending against cross-site scripting attacks," *Computer*, vol. 45, no. 3, pp. 55–62, Mar. 2012.

[21] 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 Things J.*, vol. 4, no. 5, pp. 1204–1215, Oct. 2017.

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

[23] 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 Things J.*, vol. 4, no. 5, pp. 1216–1228, Oct 2017.

[24] L. Duan, C.-A. Sun, Y. Zhang, W. Ni, and J. Chen, "A comprehensive security framework for publish/subscribe-based IoT services communication," *IEEE Access*, vol. 7, pp. 25989–26001, 2019.

[25] L. Roffia *et al.*, "A semantic publish-subscribe architecture for the Internet of Things," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1274–1296, Dec. 2016.

John S. Donnal received the B.S. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2007, and the Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 2016.

He then served in the U.S. Army as a Platoon Leader with Fort Bragg, NC, USA, and deployed to Camp Liberty, Baghdad, Iraq, from 2009 to 2010. In fall 2016, he joined the Weapons, Robotics, and Controls Engineering Department, United States Naval Academy, Annapolis, MD, USA, where he teaches embedded system design. His current research interests include nonintrusive load monitoring, predictive maintenance, and decentralized cloud infrastructure.