Devify: Decentralized Internet of Things Software Framework for a Peer-to-Peer and Interoperable IoT Device

Jollen Chen
Devify Open Source Project
Devify, Inc.
jollen@flowchain.io

ABSTRACT

This paper addresses the issue of current Internet of Things (IoT) development—the decentralized IoT model—in a manner of a peer-to-peer network and interoperable IoT devices. This paper proposes a new IoT software architecture, the Devify software framework, to address the peer-to-peer IoT network and the interoperable IoT device development. Besides, the work also shows through experiments that an IoT application server can simply use the flow-based programming (FBP) paradigm to define the application as a data exchange network. Therefore, the software architecture also provides such FBP runtime environment for writing IoT application servers.

Keywords

Internet of Things, Interoperability, Peer-to-Peer, Web of Things, Decentralized, Flow-Based Programming

1. INTRODUCTION

In recent years, the development of Internet of Things (IoT) applications has become increasingly complex. Thus, many studies have attempted to address this problem by providing the ability to stream IoT data to the IoT platforms over the web to simplify the creation of IoT applications [16]. However, current existing IoT platforms use the centralized model that they act as brokers or hubs to control the exchanged data between IoT devices. Therefore, many studies argue that IoT should use the decentralized model to ensure secure data exchange and data privacy. Thus, this paper proposes a decentralized IoT software framework to provide the ability of secure data exchange between IoT devices autonomously without any centralized server.

In short, the purpose of a new design for the IoT software architecture is that we need a JavaScript programming framework to support such full range hardware devices. Besides, the Devify software framework implements the convergences of emerging IoT trends: (i) the peer-to-peer networking for IoT devices, (ii) a full-stack JavaScript software framework, (iii) the device interoperability via REST-style RPC operations, and (iv) the use of the Web of Things (WoT) and JSON-LD ontology.

The rest of this paper is structured as follows: we describe the motivation of this work in Section 2, and subsequently study the related work in Section 3. In Section 4, we introduce the proposed software framework and review what technologies are adopted in the software framework. In Section 5, we review the design and implementation details of the peer-to-peer networks as well as the way to address the problem of churn in peer-to-peer networks. In Section 6, we propose the use of the flow-based programming (FBP) paradigm to create the IoT application servers. In Section 7, we show two Devify application examples. We conclude in Section 8.

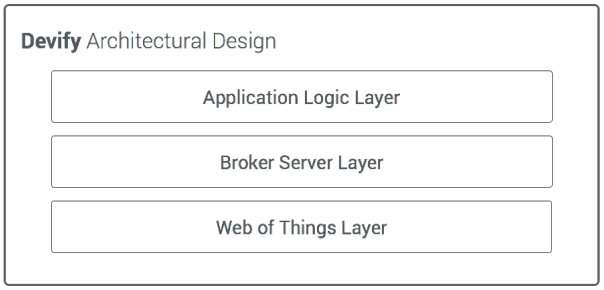

Figure 1: The Devify Architectural Design

2. MOTIVATION

The motivation of this paper is to develop a generic and comprehensive software framework for building various types of trust IoT networks in a decentralized manner. Also, current IoT network comprises a variety range of hardware devices, such as cloud servers, mobile devices, and resource-constrained devices (the heterogeneous IoT hardware devices); thus, the software framework must be able to execute on all these hardware devices. Since the JavaScript has become a significant technology in the popularity of IoT hardware devices, this paper employs a 100% JavaScript implementation for the software framework to support such IoT hardware.

Furthermore, the success of the decentralized IoT should attribute the device interoperability and a peer-to-peer network; thus, this paper will also address such technical challenges. In this work, we attempt to build a generic software framework for future developments of decentralized IoT applications. Among the potential decentralized IoT applications, the nature of the distributed ledger technology (DLT) has a large opportunity to toward a more secure and trusted IoT network. Therefore, we have already developed Flowchain [11], the blockchain for the IoT, to practically prove the concept of this work.

3. RELATED WORK AND ANALYSIS

FBP initially published by IBM in 1978 [12] has obtained a momentum by extensively projects, such as Node-RED [5] and the NoFlo open source project [4]. Also, several IoT platforms have emerged to ease the development of IoT application by using an FBP paradigm. The WoTKit [9] and Node-RED systems provide a platform that can be employed in IoT application. The work [13] is another system to apply the FBP paradigm to develop IoT applications.

Notwithstanding, most of these systems aim to provide FBP development tools that they are not specific to facilitate the microcontrollers and embedded systems. Also, when building the flow-based IoT applications, the IoT device requires an FBP runtime to perform the execution of the application. Therefore, the MicroFlo [6] runtime has proposed an implementation for microcontrollers and embedded systems in which broader used for IoT devices. In addition to an FBP programming framework, this paper provides a light-weight FBP runtime for a verity range of IoT hardware devices, from cloud servers to microcontrollers.

The use of broker services in a peer-to-peer network has subsequently been proposed to handle event-based communications for the IoT[20]. Therefore, this paper also uses such broker architecture pattern to manage sensory events in the peer-to-peer IoT networks. In addition to managing events, this paper also proposes *Virtual Nodes* built upon the broker services to manage connected wireless sensors. We will address this issue in Section 4.2.

4. THE DEVIFY SOFTWARE FRAMEWORK

4.1 Overview

The WoT ontology provides a standards-based model to represent a physical device as an IoT application server which runs on an IoT device [14]. Therefore, the Devify framework adopts the Web of Things ontology as the underlying layer along with a peer-to-peer IoT network system to provide such standards-based model to represent the physical IoT device as a physical object within the application server. Figure 1 shows the first layer (the WoT layer) attempts to utilize the WoT ontology and aims to provide application protocols over the web, such as CoAP, and WebSocket. In the Devify framework, CoAP and WebSocket are primitive protocols bindings for a node. Devify uses CoAP for the application server intended to run on resource-constrained devices, and WebSocket provides the ability to facilitate real-time data transfer.

The broker server is the second layer that implements the peer-to-peer networking, REST-style RPC operations, and a distributed hash table (DHT). Moreover, Devify selects the Chord protocol and algorithm, the technology originally published in 2001 by MIT [17], for providing such DHTs. The broker server layer also aims to simplify the creation of WoT application servers; thus, to begin developing customized IoT application servers, several project boilerplates can be downloaded through the Devify open source project.

Besides, the WoT layer can make distribution possible by providing a service contract without exposing server-side implementation details [8]. Thus, the broker server layer can efficiently encapsulate the peer-to-peer and RPC technical details on the device side, and provide the ability of devices interoperability by RPC operations over the peer-to-peer network. Typically, the broker software architectural pattern can be utilized to structure distributed software systems with decoupled components that interact by such RPC operations [10]. Thus, the broker server layer implements the broker architectural pattern to hide such technical details.

Figure 2 shows that the Devify framework uses the Node.js and V8 engines for a high-performance device, and the JerryScript [7] for a resource-constrained device. As previously mentioned, this design ensures the support of heterogeneous hardware devices. Thus, the WoT can manage such variety range of physical device as a "Virtual Thing with Thing Properties."

In addition, figure 2 shows that "Thing Description" component describes "Thing Properties" in the JSON-LD format. Consequently, the WoT represents the Virtual Thing in URI convention, as such, the URL Router component in figure 2 defines these URIs to represent the Virtual Thing. We refer a "Virtual Thing" as a "node" in this paper. Moreover, the Request Handlers component accepts incoming requests, receives data and triggers the application events. The Things Data component in the WoT layer provides the Linked Data and Semantic Web technologies to represent and manage IoT data (described in Section 5). Moreover, the Data Composition component is a significant design of the Devify software framework that it can "compose" the IoT data with the endpoints, such as the cloud platform, mobile apps, and web frontends. The Devify software framework simply uses the "forwarding" technology to implement the Data Composition component (described in Section 5).

4.2 The Broker Server

In this paper, we identify several common use cases that manage emerging IoT models: (i) the application server on the resource-constrained device, (ii) on the laptop, (iii) on the smartphone, (iv) on the IoT gateway, and (v) on the cloud server. Besides, the use of the broker architecture pattern extends these IoT models to the wireless sensor network (WSN). Figure 3 shows that the Devify application servers run on an IoT device, labeled as Broker A, Broker B, and Broker C, are deployed as the broker servers to receive the time series data continuous sent by wireless sensor nodes. For examples, the *Node-7* node is a wireless temperature sensor that it periodically sends the temperature data to *Broker C* over the local area network (LAN) with the CoAP protocol. Another example is that the Devify application server is installed on a cloud server and receive real-time sensor data over the web with the WebSocket protocol.

Furthermore, the broker architecture represents an IoT architecture for ensuring the capability of handling million sensor nodes. The broker server layer provides such broker server implementations.

MQTT, a frequently referenced IoT technology, is a publish-subscribe-based lightweight messaging protocol for use on top of TCP/IP. In MQTT networks, connected nodes are managed by MQTT brokers. Also, MQTT brokers export their nodes for external visibility. Unlike MQTT, the Devify broker server does not export its nodes; it hides sensor nodes such that all nodes are internal and not visible externally.

Figure 3 also shows that in the peer-to-peer networks, wireless sensor nodes connected to the same broker are con-

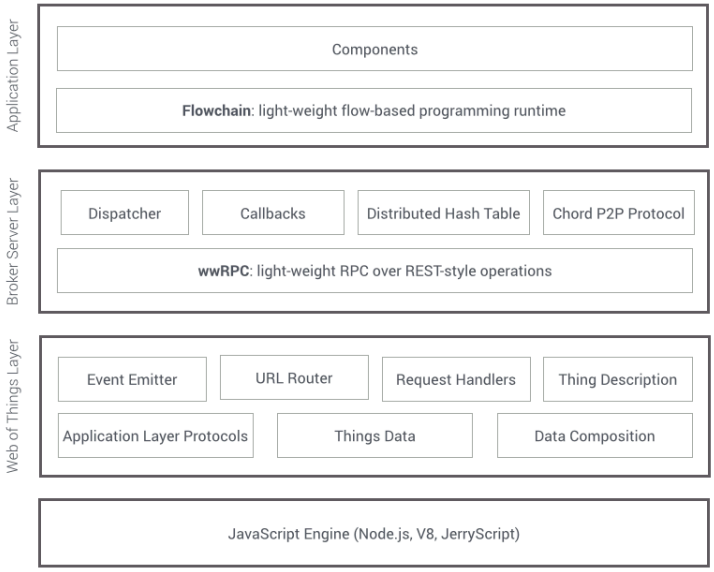

Figure 2: The Devify Software Components

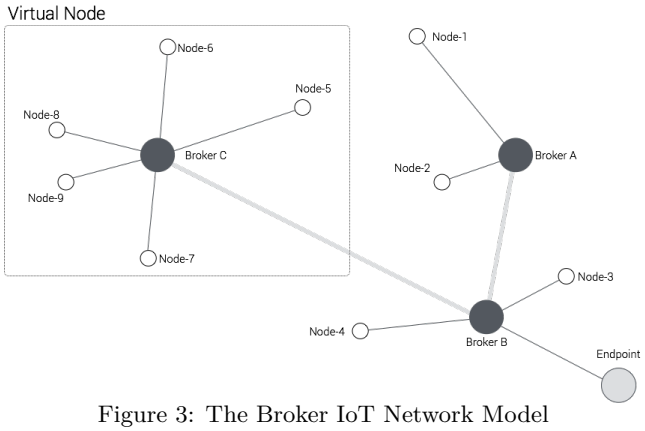

Figure 3: The Broker IoT Network Model

ceptually "grouped" together with the broker server as a "virtual node." In summary, a virtual node comprises a broker server and its corresponding nodes. In this manner, the broker server can be responsible for managing its connected sensor nodes.

5. DEVICE INTEROPERABILITY

5.1 The Peer-to-Peer Network Implementation

Figure 2 show that the DHT and the Chord protocol are key components of the Devify software framework. The IoT nodes or "virtual nodes" are organized as a peer-to-peer network in the DHT by using the Chord protocol. Also, the web-to-web RPC (wwRPC) component is a significant design that it offers the REST-style RPC operations and collaborates with the DHT to provide the ability of device interoperability. Subsequently, writing a broker server is simple by using the Devify programming framework. Listing 1 shows a program to start the devify server on an IoT device and subsequently joins a peer-to-peer network.

```
// Require Broker class in Devify Platform
var DevifyBroker = require‘(devify.’io).Broker;

// To instantiate a broker server instance
var broker = new DevifyBroker({
    host: ‘’192.168.0.1, port: 8000,
    join: {address: ‘’192.168.0.100, port: 8000}
});

// To start the broker server
broker.start();

// The virtual node is up and listening
```

Listing 1: A Broker Server Sample Code

A broker server also offers periodic health and failure checks for nodes. As mentioned previously, the broker hides all implementation details. Figure 2 has indicated that our work uses the Chord protocol for a peer-to-peer network. The Chord messages are exchanged between IoT nodes via the *wwRPC* component. And the dispatcher component receives and dispatches these RPC messages. Listing 2 shows the API implementation for sending data to the successor node over the peer-to-peer network. Besides, the API includes the Linked Data context in advanced and extends the RPC messages with this context before sending data. As described in Section 4.1, the Things Data component uses JSON-LD as the primary semantic web technology to structure the data into a key-value pair data object.

```
/*
 * Send data to the peer-to-peer network.
 */
Node.prototype.send = function(data) {
    var key = ChordUtils.hash(data);

    var message = {‘’
        @context: ‘http://devify.io/context.jsonld’,
        id: key,
        type: Chord.FIND_SUCCESSOR,
        data: data
    };

    return this.send(successor, message);
};
```

Listing 2: The API for Sending Data

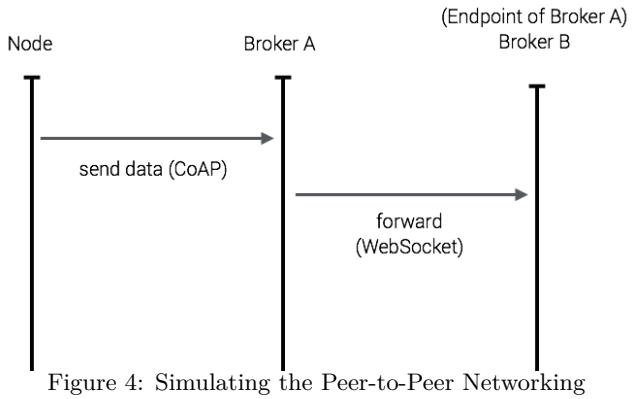

Figure 4: Simulating the Peer-to-Peer Networking

5.2 Simulating the Data Transfer

Figure 4 simulated the process of sending IoT data to the peer-to-peer network: the sensor data is sent to one broker within the peer-to-peer network and subsequently routed over the Chord ring to *Broker A*. Moreover, the broker server can forward the data to an "endpoint." Ordinarily, the endpoint is a cloud platform, such as Dropbox and Twilio or another broker server.

Consequently, the application server uses CoAP and WebSocket URIs to represent the Virtual Thing for a node; thus, every node can be addressed using unique URI's. Listing 1 shows that the URL Router component of the software framework defines such URIs to represent a Virtual Thing.

A significant design of the software framework is that the dispatcher component uses an event-driven concurrency model to handle RPC messages. Currently, many developers avoid the multi-thread model and employ an event-driven approach to concurrency management [19]. Therefore, and due to the scalability limits of threads, the wwRPC subsystem uses an event-driven model.

```
[coap|ws]://[hostname]/object/[name]/[id]
```

Listing 3: Virtual Things in the URI Convention

5.3 HANDLING CHURN IN THE DHT

The churn, the continuous activities of node join and leave, is an essential characteristic of a peer-to-peer network. This paper addresses the problem by using the periodic nature of Chord stabilization algorithm and extending the algorithm to handle churn in the DHT [15] In a Chord network, the stabilization algorithm periodically checks its successor node and update the DHT. Therefore, the IoT node can select its successor node for measuring churn and simply uses the timeout calculation technique for handling churn [18].

Listing 4 shows the MIT Chord algorithm [1] and the extension to handle churn. In the stabilizing process, the IoT node keeps correcting its successor node and notifies the successor node about its predecessor. We extend this process so that the successor node can notify the IoT node about its alive.

```
// It is called periodically.
// n asks the successor
// about its predecessor.
n.stabilize()
    x = successor.predecessor;
    if (x is in (n, successor))
        successor = x;
    successor.notify(n);

// 'n thinks it might be our predecessor, and
// n notify 'n about its alive.
n.notify('n)
    if (predecessor is nil or 'n is in (predecessor, n))
        predecessor = 'n;'
    n.notify_ttl();

// n updates the 'successors TTL.
n.notify_ttl()
    n.successor_ttl = MAX_TTL;
```

Listing 4: Handling Churn

Additionally, the IoT node decreases the TTL in a fixed time interval. If the TTL drops to zero, the IoT node handles churn by removing the successor node from the peer-to-peer network and subsequently the Chord stabilization process will correct the successor node. Also, the failure nodes can spontaneously intend to join the peer-to-peer network, and the periodic stabilization algorithm will update the DHTs. We summary this process in Listing 5.

```
// Failure check
setInterval(function check_successor() {
    // Decrease 'successors TTL
    n.successor_ttl = n.successor_ttl - 1;

    // Remove the successor node
    // if the successor node has already left the network.
    if (n.successor_ttl < 1)
        n.successor = nil;

    // The periodical stabilization algorithm will find
    // the new success node.
}, 1000);
```

Listing 5: The TTL Update Algorithm

6. THE FBP PROGRAMMING MODEL

This paper shows that an IoT application can simply use the flow-based programming (FBP) paradigm to define the applications as networks of black box which process exchange data across predefined connections by message passing [3]. Thus, Devify also provides an FBP engine to support such programming paradigm. Also, FBP is naturally component-oriented that the applications are structured with predefined components connecting as data processing networks.

6.1 Applying Flow-based Programming Paradigm

The IoT network mainly comprises data, the flows, and the connectivity; therefore, several studies [13] mentioned that the IoT needs the data-driven approach to developing IoT applications. With the increase of data volumes by the IoT devices, such data-driven applications become an important role in the IoT domain.

Compare the FBP to other broader programming paradigms, such as the object-oriented programming; the FBP is in the data-driven approach. Therefore, Devify has already implemented a light-weight FBP runtime (*scheduler*) support data-driven programming required by current IoT devices. Also, as previously mentioned, the FBP runtime is written in JavaScript that it can support the variety of IoT hardware devices.

Also, the FBP also manifests loosely data coupling which the concept admits a more fine-grained software architecture than comparing to object-oriented design. In short, the FBP

Figure 5: The FBP Programming Model

paradigm shows additional advantages for the IoT through our experience: (i) where the IoT application is comprised of highly reusable components, and (ii) in which the data processing network is constructed as asynchronous processes.

6.2 Developing IoT Application Servers

This paper presents a software framework that simplifies the creation of IoT applications by reusing existing web technologies and applying the FBP programming paradigm. For example, setting up a sensor node to gather information and communicate with other nodes requires only a few lines of code. Moreover, to build a complete peer-to-peer and decentralized IoT network also requires only a few lines of code.

The FBP paradigm defines applications as networks and exchanges data across predefined connections [12]. The Devify software framework utilizes the FBP paradigm for IoT application development. Thus, the Devify software framework can be used to write flow-based IoT applications.

Developers can write IoT application code using the FBP paradigm and JavaScript. With the FBP paradigm, an IoT application is described by "FBP components" and their corresponding connections. Figure 5 defines an IoT application server as a "graph" in the JSON format, and the listing 6 shows such implementation.

```
var graph = {''
    type: ''coapBroker,''
    connections: [
        {
            upproc: 'io.devify.'sms,
            upport: ''out,
            downproc: 'io.devify.'console,
            downport: ''in
        }
    ]
}

broker.start({ graph: graph });
```

Listing 6: The Devify Application Server in the FBP Paradigm

As described in Figure 2, the application layer provides an FBP-like runtime engine with a unidirectional data flow design. The unidirectional data flow reduces the complexity of device interoperability. Consequently, the FBP component has a single output port and a single input port. The "connection" is established from the "outPort" of one component to the "inPort" of another component. The FBP-like runtime engine is responsible for executing the "graph" and processing the data flow. Moreover, the FBP components are highly decoupled; thus, developers can build and publish the reusable components.

7. THE IOT APPLICATIONS

7.1 The Wireless Sensor Networks Example

Figure 6 show that the peer-to-peer network organizes IoT devices named as N1 to N8 in the DHTs as a Chord ring. In the Chord peer-to-peer network, each data is hash by the

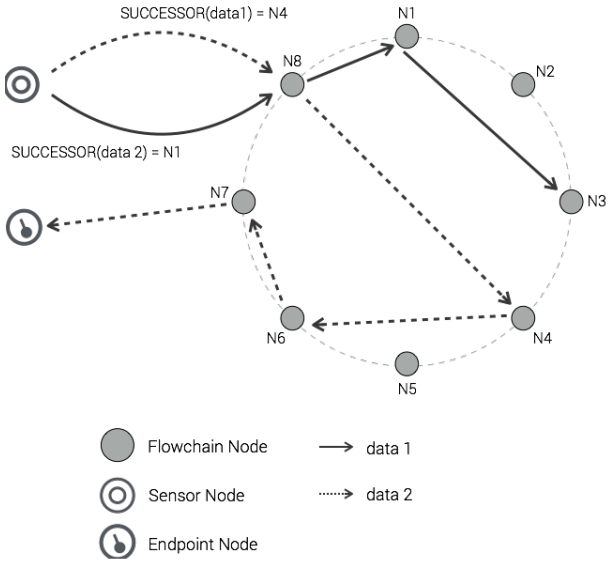

Figure 6: The Decentralized Wireless Sensor Networks

SHA1 hash function resulting in data "key." Then, the peer-to-peer network routes the data to the data's "successor" node over the Chord ring. Moreover, figure 6 demonstrates that a wireless sensor node is deployed to transfer temperature data to the *Endpoint* periodically. In contract to use the centralized IoT model, figure 6 uses a decentralized IoT model, and the most important processes of this decentralized IoT example are summarized as follows:

- Each node starts a Devify application server, and subsequently join the peer-to-peer network.
- N8 is the virtual node that manages the wireless sensor.
- The *geometry* of IoT nodes, named N1 to N8, are organized as a ring topology in this peer-to-peer network.
- The N8 is named in HTTP URIs so that the wireless sensor node can send the data to N8 by such URIs.
- The wireless sensor node sends *data 1* to N8: N8 hashes the received data by SHA1, lookups the successor node of the data by the hash key in the DHT, and routes the data to N1. In this step, the successor node is N4.
- The data is routed to N4: N8 communicates with N4 by using the wwRPC component, as described earlier, the wwRPC component provides REST-style RPC operations for the peer-to-peer network.
- The data is eventually routed to N7.
- N7 forwards the data to *Endpoint*. As described in 5, cloud platform is the example of the endpoints.

7.2 IoT Broker Server in the Cloud

Section 4.3 describes that this paper develops several IoT models that developers can also deploy the Devify application server in the cloud. As such, this section gives a demonstration of the use case: one sensor node attempts to send IoT data streams to multiple clients. Figure 7 shows that many clients, named Viewer Client in this example, need to view the IoT data from the single sensor node. In this case, the sensor node, named Sender Client in this example, sends the data streams over the web to WebSocket broker

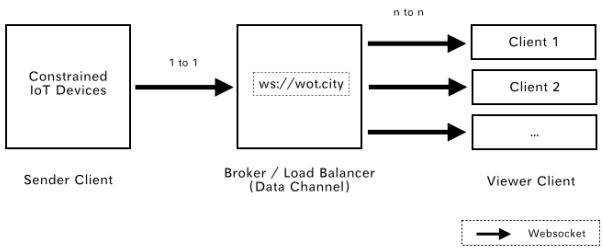

Figure 7: The Broker Server in the Cloud

server. As described earlier, the WebSocket broker server is deployed in the cloud server. In this example, the viewer client could be the mobile apps (iOS and Android), and web apps (HTML5).

Moreover, this example also attempts to give a use case of the Devify framework that the constrained devices can stream time series data over the web to multiple clients. This example addresses the problem of handling connections: a resource-constrained device has limited memory and computation power that it can offer to accept and handle too many View Client connections. Therefore, we use a cloud server to deploy the Devify application server.

Furthermore, the wireless sensor node can find the WebSocket broker server by the URIs. In this example, the broker server is deployed at the *wot.city* server that, as described in Section 4, the application server can represent the cloud server with the following URIs:

```
ws://wot.city/object/554785c7173b2e5563000007/send
ws://wot.city/object/554785c7173b2e5563000007/viewer
```

Listing 7: The Broker Server URIs

8. CONCLUSIONS AND FUTURE WORK

This paper has practically implemented the Devify software framework available as an open-source project, and it consists of three sub-projects. Accessible at (i) wotcity.io (https://github.com/wotcity), (ii) devify.io (https://github.com/DevifyPlatform), and (iii) flowchain.io (https://github.com/flowchain). Despite the Devify software framework, IoT devices in a decentralized IoT network may require a new model to exchange data. Given the blockchain's private ledger nature, it is natural that Devify will use the blockchain technology to provide secure and trusted data exchange. Therefore, we have already begun to build Flowchain [2], a blockchain-based decentralized IoT platform, by using the Devify software framework as the underlying system.

REFERENCES

[1] Chord (peer-to-peer). https://en.wikipedia.org/wiki/Chord_(peer-to-peer).

[2] A distributed ledger for the internet of things. https://flowchain.io/.

[3] Flow-based programming. https://en.wikipedia.org/wiki/Flow-based_programming.

[4] Flow-based programming for javascript. https://noflojs.org/.

[5] Flow-based programming for the internet of things. https://nodered.org/.

[6] Flow-based programming runtime for microcontrollers (arduino). http://microflo.org/.

[7] Ultra-lightweight javascript engine for the internet of things. https://github.com/Samsung/jerryscript.

[8] Web of things interest group charter. https://www.w3.org/2014/12/wot-ig-charter.html.

[9] M. Blackstock and R. Lea. Toward a distributed data flow platform for the web of things (distributed node-red). *Proceedings of the 5th International Workshop on Web of Things - WoT '14*, 2014.

[10] F. Buschmann, M. Kircher, and D. C. Schmidt. Pattern oriented software architecture, 2011.

[11] J. Chen. Flowchain: A distributed ledger designed for peer-to-peer iot networks and real-time data transactions. *To appear in 2nd International Workshop on Linked Data and Distributed Ledgers*, 2017.

[12] J. P. Morrison. Data stream linkage mechanism. *IBM Systems journal*, 17(4):383–408, 1978.

[13] L. Oleksandr and K. Alexandr. Applying flow-based programming methodology to data-driven applications development for smart environments. pages 216–220, 2014.

[14] D. Raggett. An introduction to the web of things framework. https://www.w3.org/2015/05/wot-framework.pdf.

[15] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. 2004.

[16] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-P. Calbimonte, M. Riahi, K. Aberer, P. P. Jayaraman, A. Zaslavsky, and I. P. e. a. Žarko. Openiot: Open source internet-of-things in the cloud. *Interoperability and Open-Source Solutions for the Internet of Things*, pages 13–25, 2015.

[17] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, 2003.

[18] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. *Proceedings of the 6th ACM SIGCOMM on Internet measurement - IMC '06*, 2006.

[19] M. Welsh, D. Culler, and E. Brewer. Seda: An architecture for well-conditioned, scalable inter-net services. *ACM SIGOPS Operating Systems Review*, 35(5):230, 2001.

[20] H. Ziekow. In-network event processing in a peer to peer broker network for the internet of things. *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, pages 970–979.