NDN, Technical Report NDN-0066, 2018. http://named-data.net/techreports.html
Revision 1: [10/2/18]

Publish-Subscribe Communication in Building Management Systems over Named Data Networking

Wentao Shang*, Ashlesh Gawande†, Minsheng Zhang†, Alexander Afanasyev‡,
Jeffrey Burke*, Lan Wang† and Lixia Zhang*
*UCLA
{wentao,lixia}@cs.ucla.edu, jburke@remap.ucla.edu
†University of Memphis
{agawande,mzhang4,lanwang}@memphis.edu
‡Florida International University
aa@cs.fiu.edu

***Abstract*—Publish-subscribe (pub-sub) has been recognized as a common communication pattern in IoT applications. In this report we present NDN-PS, a distributed pub-sub communication framework for modern building management systems (BMS), an important area of IoT, over the Named Data Networking (NDN) architecture. NDN-PS utilizes distributed NDN repositories to store and republish large quantities of BMS data that can be consumed by different applications. It employs a data synchronization mechanism to aggregate multiple data streams published by multiple sensing devices and achieve efficient notification of new data for the consumers. NDN-PS also provides data authentication by utilizing NDN's security building blocks. We have implemented a prototype of NDN-PS and are in the process of evaluating its design through an experimental deployment. This design exercise demonstrates that the information-centric architecture enables a simple design for complex IoT systems and provides superior system efficiency and security over the TCP/IP-based alternatives.**

I. Introduction

Enterprise Building Automation and Management System (EBAMS, or BMS for short) is a key IoT application used by enterprises to monitor building environment and lower costs. A typical BMS deployment spreads across many buildings on an enterprise campus, with tens of thousands of sensors installed on the premises to monitor electricity, lighting, temperature, humidity, and many other environmental parameters. Recent years have witnessed an extensive amount of effort by both industry and research communities to adapt or enhance the current TCP/IP architecture to support BMS, yet many challenges still remain. For example, sensing devices may have intermittent connectivity and use duty cycles to save energy, while IP communication assumes an "always on" model and the associated channel-based security require the two communicating end points to be online at the same time. Moreover, it is tedious and error-prone to manually set up and manage IP-based BMS systems as IP addresses do not reflect the relationship among the entities in the system, including the sensing devices, sensor data, environment being sensed, operators, and other users.

We believe that the above challenges intrinsically come from the incongruity between TCP/IP's basic communication and security model and the functional requirements of IoT applications [1]. In contrast, several built-in architectural features in Named Data Networks (NDN), such as expressive naming, in-network caching, and data-centric security, make it much easier to build scalable and secure IoT applications [2], [3] over NDN.

For a proof of concept, we have designed a publish-subscribe communication framework called **NDN-PS** for BMS environments over the NDN architecture [4]–[6]. We have prototyped NDN-PS on the NDN platform [7] and evaluated it using our emulation tool Mini-NDN. We are also verifying the design with a real-world deployment at the University of Memphis campus using Raspberry Pi's and wireless sensors. The work serves as a case study to illustrate i) the use of *NDN architectural features* to develop a specific IoT application, and ii) the unique *application design patterns* that arise from a data-centric communication model.

The core functionality of a building management system is the production and consumption of sensing data. As such, a major design challenge is *how to enable individual applications to retrieve sensing data of their interest in real time over the network*. Note that (1) each data consuming application may be interested in a *different subset* of the sensing data; (2) sensors and consumers may not be online at the same time; and (3) the number of sensors and consumers can potentially be very large. To address these data communication challenges, NDN-PS leverages distributed NDN data repositories (*repos* in short) as intermediaries to decouple data production and consumption, and incorporates an efficient pub-sub mechanism (built on top of NDN's request-response primitives) for consumers to subscribe to arbitrary data sets of interest (Section III). Through this design exercise we also demonstrate the utility of NDN Sync protocols (Section II) in facilitating distributed data production and subscription in a data-centric communication model.

Security is critical for building management systems. NDN-PS ensures data authenticity using a hierarchical trust model to verify the signature in each piece of received data (Section III-E). The authentication policy is encoded in a trust schema that leverages the expressive power of data naming [8].

This data-centric security model fundamentally differs from TCP/IP's channel-based security model in that it ensures the security property of the data itself, and does not depend on the existence of secure channels (e.g., a TLS/DTLS session) between communicating entities.

In addition to presenting our system design, implementation and evaluation, we also discuss how one can build a large scale, complex BMS data acquisition system by chaining together multiple pub-sub groups in a hierarchical way and perform data aggregation at different levels of the hierarchy (Section V). We further compare NDN-PS with the pub-sub support in traditional BMS protocols and cloud-based IoT systems, and articulate the advantages of building the pub-sub framework on top of a data-centric network architecture. We also review the literature of pub-sub systems proposed for other ICN architectures (Section VI). Finally we conclude the paper and address future work in Section VII.

II. Background

A. Data Acquisition and Access in BMS

In a typical BMS deployment, sensors are often hardwired to smart panels or controllers that are connected to a common high-speed backbone network, usually logically and/or physically isolated from other internal and external networks. Future BMS may also incorporate wireless sensors that are associated with a wired aggregator, or even connect low-cost, wireless sensors directly to the network.

In such systems, different types of sensors continuously generate a large amount of measurements such as room temperature, power consumption, and chilled water flow. Due to limited storage and processing capability on typical panels and controllers, those data points have historically been collected into dedicated storage at the enterprise level and archived for certain period of time (typically one or two years) in order to serve different data analytics applications.

Enterprise level BMS requires advanced data access support to meet various application requirements including the following:

- Different consumers may be interested in the data produced by different sensors. Requiring direct communication between each sensor/consumer pair would not scale well.
- Consumer applications may run on diverse platforms ranging from high-end servers, smartphones, to embedded systems. These applications consume the sensor data at different times and speeds and may not always be available when new data is produced.
- Different applications may have different semantics in consuming the data. For example, some applications may be interested only in the latest data generated in real time, while others would want to periodically receive a few random samples from a batch of collected data.

B. NDN and BMS

NDN is a new Internet architecture that provides data-centric communication at the network layer. NDN implements an asynchronous request-response communication pattern that naturally decouples data producers and consumers. It defines two types of network layer packets: *Interest* and *Data*. Each data producer assigns a unique and hierarchical name to every Data packet it generates. Each consumer issues an Interest packet with a data name or name prefix, which is forwarded based on the name (prefix) and can be satisfied by a Data packet with a matching name. For each received Interest, NDN forwarders use *forwarding strategies* [9] to decide where to forward the Interest by taking into account the usage policies, the FIB, and the measurement from previous forwarding decisions. Each Interest packet brings back at most one Data packet; if the Interest carries a name prefix that can be satisfied by multiple Data responses, one of them is forwarded and the rest may be cached at the forwarders where these multiple responding Data packets meet.

Each Data packet carries a cryptographic signature, securely binding the content to the name, which ensures integrity and provenance of the data. As such, NDN Data packets can be retrieved either from original data producers, managed data storages (repos), or opportunistic caches, enabling asynchronous data production/consumption and significantly improving the overall scalability and efficiency in data delivery. NDN data is also *immutable*: any change to a piece of existing data produces a new version of the data with a different name.

NDN brings several important benefits to the design and implementation of BMS. First, NDN forwards Interest and Data packets using the application-layer names, which eliminates the need to configure BMS networks with IP addresses, a significant simplification when there exist a large number of connected sensors, actuators, and controllers. Second, NDN's data-centric security mechanism inherently secures every produced piece of data for its lifetime, instead of relying on physical/logical isolation and communication channel security. Finally, the in-network storage including forwarder caches and repos reduces the query load on the sensors and improves the scalability of the BMS data communication. As we will show in this paper, NDN-PS takes advantage of those benefits from NDN to achieve secure and scalable pub-sub communications.

Our previous work on NDN-BMS [10] designed a BMS data acquisition system over NDN and demonstrated the design of the data namespace, collection of the data from off-the-shelf devices into NDN repositories (repos), and data security via packet signatures and encryption-based access control. However, NDN-BMS controls data access at the granularity of individual sensor readings, and does not support publish-subscribe communication model to enable efficient and timely delivery of newly published data to consumers who subscribe to multiple but different subset of sensing data simultaneously. In comparison, NDN-PS develops a generic pub-sub communication support as a data transport service on top of the data acquisition system to facilitate heterogeneous consumer applications in accessing sensing data.

C. Data Synchronization in NDN

Data synchronization (Sync) is an important building block for distributed applications. While distributed applications may differ in their specific goals and communication patterns, they share a common need for synchronizing the application datasets among multiple parties. Since distributed applications are a generalization of 2-party communications, one may view Sync as a generalization of end-to-end reliable data delivery among multiple parties.

NDN is particularly suited in supporting multi-party communication synchronizations. Since communication in NDN is based on fetching named data, and there is a unique and secured binding between a name and its content, an NDN application can represent its state by the set of data names. Therefore NDN sync protocols can simply focus on the synchronization of the dataset names. Once all the entities in the same application obtain an identical set of data names, then they can individually decide on when to fetch which data published by others.

NDN Sync protocols bridge the gap between NDN network layer's datagram Interest-Data exchange primitive and the application layer's need for data sharing among multiple participants, in a way similar to the role of TCP which bridges the gap between IP's datagram service and applications' need for reliable delivery in today's Internet. However, Sync differs from the existing end-to-end reliable transport protocols, such as TCP, in three fundamental ways. First, Sync synchronizes application-named datasets among multiple parties, while a TCP connection delivers byte streams identified by its two endpoints. Second, nodes running Sync can fetch data by name from anywhere it finds the matching data items since the security is attached to the data instead of its container or communication channel. Third, Sync does not require all communicating parties to be interconnected at the same time, while a TCP connection delivers byte streams between two specific endpoints synchronously (i.e. both must be online at the same time). The ability to reconcile dataset differences asynchronously is especially useful in constrained environments with sleeping nodes and intermittent connectivity.

Several sync protocols have been proposed for the NDN architecture [11], including ChronoSync [12], iSync [13] and PSync [14]. In ChronoSync, the producers in the sync group publish data that are identified by the producer's name and a monotonically increasing sequence number.[1] The state of the dataset is concisely represented as a list of key-value pairs that maps each producer to its latest sequence number. The ChronoSync protocol disseminates the sequence numbers of new data via multicast so that everyone in the sync group can update its local state accordingly. iSync uses Invertible Bloom Filter (IBF) [15] to represent a set of arbitrary names by storing the hash values of the data names in the IBF. Each producer in iSync periodically advertises the digest of its IBF.

[1]This does not reduce the generality of the protocol since the applications can encapsulate the data objects named under different naming conventions in the sequentially named data packets.

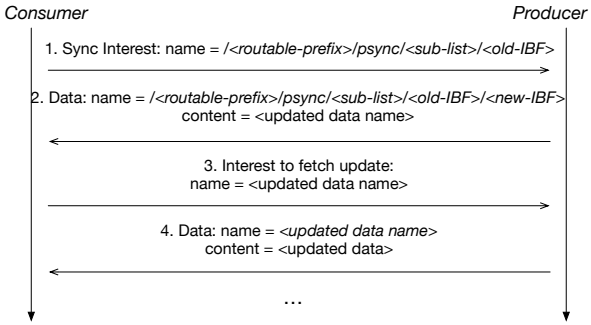


Fig. 1: PSync message exchanges between consumer and producer (<sub-list> is the consumer's subscription list encoded in Bloom filter, while <old-IBF> and <new-IBF> are the producer's previous and current dataset state encoded in IBF)

Whoever has a different digest can fetch the IBF from the producer, extract new hash values using IBF subtraction, and request the actual data names corresponding to those hashes from the producer.

PSync has a unique feature of allowing consumers to synchronize with producers on specific subset of the data namespace, which is particularly suited for NDN-PS. Similar to ChronoSync, PSync assumes data names from each data stream contain a name prefix and a monotonically increasing sequence number. PSync uses an IBF to encode the set of the latest data name from all streams. The consumers can subscribe to any subset of the data streams and receive the latest data generated in those streams. Each consumer maintains its own list of subscribed data stream prefixes (called subscription list), which is efficiently encoded by a Bloom filter.

Figure 1 illustrates the PSync message exchanges between a consumer and a producer. The producer encodes the latest data names from all its data streams in an IBF, which is sent to the consumers in PSync replies and again sent back by consumers in PSync Interests in order to track the changes in the producer's dataset. In addition, each PSync Interest from a consumer contains its subscription list so that the repo can filter out the changes relevant to the consumer. Because PSync includes consumers' subscription information and producer's dataset state explicitly in the Interest/Data names, it minimizes the round-trip required for synchronization and reduces the soft-state information kept by the producer. Moreover, this design allows the consumers to retrieve updates from any of the repos that share the same set of sensor data, which is essential for enabling the multi-repo pub-sub framework in NDN-PS.

In addition to the pub-sub functionality, PSync can also be used to support full synchronization among a group of participants, and its use of IBF allows it to extract differences between the data names at two participants more efficiently than ChronoSync.

III. System Design

NDN-PS is a pub-sub communication framework designed to support data access over NDN in a typical enterprise BMS. In this section, we present our design assumptions, goals and detailed design.

A. Design Assumptions and Goals

We make the following design assumptions based on the BMS data access requirements stated in Section II-A.

First, we assume that the BMS sensors are hardwired to some smart panel or controller that speaks the *NDN protocol* on the enterprise network. The panels gather the readings from the sensors and package them into NDN Data packets.

Second, since the panels usually have limited storage, they need to transfer data to a long-term storage device, i.e., *NDN repo [16]*, for data archiving and access. The repos are the core components in NDN-PS that coordinate the communication between sensors and applications. We expect the repos to run on servers with enough storage to host the data generated by the sensors they are serving. The repos are connected to the enterprise network via NDN and stay online to process pub-sub requests. They typically do not have any constraint on computation capability or energy budget.

Third, the consumers in the BMS include data acquisition applications running on servers, laptops, and smartphones, as well as controller logics on the smart panels that monitor the data created by other panels. Mobile consumers on laptops and smartphones may have intermittent network connectivity (e.g., when the user closes the laptop or put the smartphone app into background) and thus do not always stay online. They may also experience other types of constraints such as low-power CPU and limited storage.

The NDN-PS design aims to achieve the following goals:

- Scalability: The framework should support a large number of data streams and consumers participating simultaneously in a single pub-sub group with arbitrary subscription relations.
- Availability: The framework should provide redundancy and automatic failover to make sure the producers can publish new data and the consumers can fetch updates as long as a subset of the pub-sub repos are running.
- Security: The framework should enable authentication for the communication between producers, consumers, and pub-sub repos.

B. Design Overview

NDN-PS provides a pub-sub communication framework on top of our earlier work NDN-BMS [10] to address the challenges in BMS data consumption. Each sensor's data points form a *data stream*, which is published under an NDN name prefix by the smart panel that the sensor attaches to. To overcome intermittent connectivity and resource (e.g., CPU, storage, and energy) limitations, the published sensor data is stored in a nearby *NDN repo* for archiving and access. In order to provide redundancy and efficiency of data retrieval, sensor data is typically replicated in multiple repos using PSync.The repos serving a particular replicated dataset and consumer applications interested in that dataset form a *pub-sub group*. For example, five buildings may publish their data to three nearby repos; the repos synchronize their data with each other, so that consumers can obtain the data for any of the buildings from any repo.

Consumer applications in a pub-sub group may subscribe to any subset of the BMS data streams that are identified by the stream name prefixes, and pull updates from one of the pub-sub repos about the newly published data in their subscribed data streams. NDN-PU uses PSync protocol to support the communication between the consumers and the pub-sub repos. For example, a pub-sub group may generate data under the prefix /Company/Building1/Electricity, where each pub-sub repo stores data streams with prefixes of the form /Company/Building1/Electricity/⟨panel⟩/⟨device⟩/⟨metric⟩/. Suppose a consumer is interested only in Panel 2's data, it can subscribe to that panel's name prefixes using PSync so that its pull requests will be answered whenever there are new data points generated under those name prefixes. Based on the notification information, the applications can then make local decisions of whether to retrieve the data, which can be done through regular NDN Interest-Data exchanges.[2]

All data packets in NDN-PS are authenticated using a hierarchical trust model expressed in the NDN names, which is aligned with real-world physical or logical structures such as campus buildings and enterprise management (Section III-E). The sensor data may be encrypted for access control [10], in which case the data decryption keys (which may be refreshed periodically) are also distributed to the consumers as data streams over NDN-PS.

NDN-PS supports the complex pub-sub relationship between many producers and consumers by aggregating the sensor data and the consumer requests at the pub-sub repos. As such, the repo is a core component in the NDN-PS framework. The number of repos in a pub-sub group is typically determined by the deployment scenario. For example, a small pub-sub group running inside a single building and generating 10s of data streams may need only two or three repos to replicate the data, while a large pub-sub group spanning across the whole campus may need five or more repos to collect BMS data from different locations and serve many consumer applications.

Figure 2 shows the modules inside a repo. The BMS panels and aggregators continuously publish *streams* of sensor readings which are pulled into remote repos using the *data interface* for archiving in the *data store*. The *Replication interface* monitors the addition of new data in the data store and synchronizes the group dataset with other repos using the full sync API of PSync. The *Pub-sub interface* enables the consumer applications to subscribe to different data streams using the partial sync API of PSync. Each consumer requests for data updates based on its own schedule and decides

[2]If the data size is small, the notification message may optionally include the new data point(s) to avoid extra messages and delay.

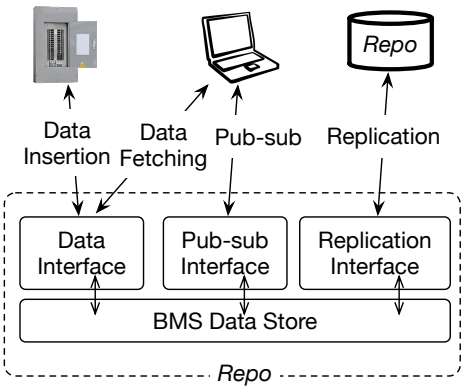


Fig. 2: System modules inside the repo

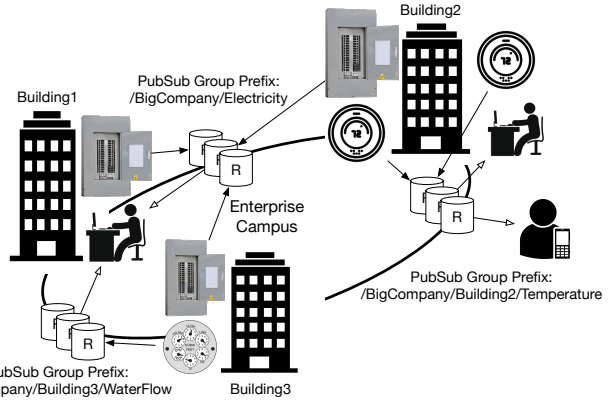


Fig. 3: Deployment of three pub-sub groups on an enterprise campus network that serve different types of BMS data: electricity, temperature, and water flow

independently whether to fetch the new data according to the application semantics. Finally, the *data interface* handles data fetching Interests from the applications.

Multiple pub-sub groups can be deployed independently on the campus network to support different applications and services either around the same location or across different buildings, as illustrated in Figure 3. Different pub-sub groups can also be concatenated together, with the BMS applications subscribing to and processing the input data in one group and publishing the output data in another group. In this section we describe only the protocol operations within a single pub-sub group. In Section V we discuss how to connect multiple pub-sub groups to build a more complex data aggregation framework for BMS.

C. Data Publication and Acquisition

The BMS panels in a pub-sub group usually publish data points that are relevant to a class of data acquisition applications with similar functionality. The data points are grouped into *streams* by their name prefixes (which also serve as stream identifiers). Figure 4 shows an example of data names in a pub-sub group with two electrical panels. The group is identified by the prefix `/BigCompany/Building1/Electricity`

Pub-Sub Group Prefix:

/BigCompany/Building1/Electricity

Data Streams:

/BigCompany/Building1/Electricity/Panel1/Heater/Voltage/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel1/Heater/Current/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel1/Vent/Voltage/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel1/Vent/Current/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel1/Switches/Voltage/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel1/Switches/Current/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel1/Plugs/Voltage/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel1/Plugs/Current/{1,2,3,...}

/BigCompany/Building1/Electricity/Panel2/Projector/Voltage/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel2/Projector/Current/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel2/Speaker/Voltage/{1,2,3,...}
/BigCompany/Building1/Electricity/Panel2/Speaker/Current/{1,2,3,...}
...

Fig. 4: Example of data names in a pub-sub group

which indicates this pub-sub group provides power usage data for `Building1` on the campus of `BigCompany`. Each panel manages different appliances in the room (e.g., heater, vent, switches, plugs, projector, speaker, etc.) and continuously generates voltage and current readings for those appliances.

NDN-PS requires the last component of the data name to be a unique *sequence number* that gets incremented by one for each new data point within a stream. This can be easily achieved if all the data from a stream is generated from the same panel: that panel can maintain a local counter that is monotonically increasing and assign the sequence numbers using the value of the counter. If multiple panels are contributing to the same stream, which is an uncommon case, some coordination mechanism (outside the scope of NDN-PS) must be employed to ensure the uniqueness and continuity of data names generated by different panels.

There can be a number of means to collecting BMS data points into the repos. In NDN-PS, the data producers use the *Repo Insertion Protocol* [17] to notify the repos to retrieve the data when they are generated (shown as step ① in Figure 5). Each panel interacts with only one of the repos in the pub-sub group during data insertion. To allow automatic failover in the face of repo failure, the repos in the same group announce the same name prefix in order to receive notifications from the panels. The NDN forwarders in the network will use the *Best Route* forwarding strategy [18] to direct each data insertion request toward the closest repo. The repo receiving the data insertion request will send an interest to the data producer to retrieve the data.

Once the repo retrieves the data point, it will replicate the data in the background to other repos in the same group using PSync's *full sync* API (shown as step ② in Figure 5). Rather than synchronizing the sensor data streams directly in the PSync group, the repo publishes a Data packet under its own name prefix which "wraps" the sensor data names in the content, and synchronizes this "repo data" instead. This additional level of indirection ensures that the size of the PSync state maintained by every repo is proportional to the

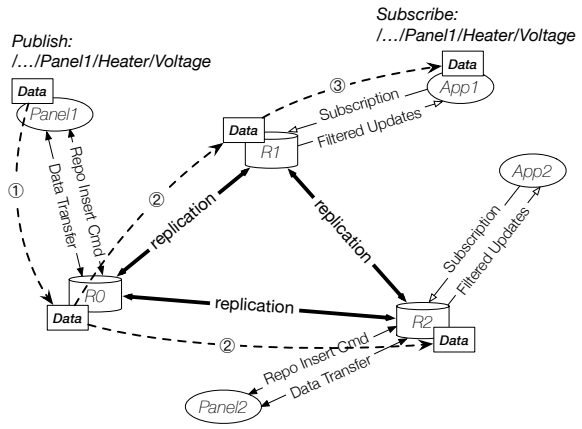


Fig. 5: Data flow in a pub-sub group

Name: ***/BigCompany/Building1/Electricity***/Repo/Repo1/<seq#>

Group Prefix

Content: {/<Group-Prefix>/Panel1/Heater/Voltage/<seq#>,
/<Group-Prefix>/Panel1/Heater/Current/<seq#>,
/<Group-Prefix>/Panel1/Plugs/Voltage/<seq#>,
/<Group-Prefix>/Panel1/Plugs/Current/<seq#>,
…
}

Fig. 6: Example of repo data published in PSync

number of repos in the group rather than the number of data streams. Another important benefit is that when many sensors generate data points at the same time, the repo can batch multiple new sensor data names in one repo data packet in order to improve the efficiency of the sync process. Figure 6 shows an example of the repo data including its name and content which contains a snapshot of the latest data names for all the data streams in the repo.

When other repos receive the notification of the new sensor data names published by some repo via PSync, they will send Interests that carry a *forwarding hint* [19] pointing to the repo's unicast name prefix in order to retrieve the sensor data packets from that repo. Here we do not want the interest to reach the data producer directly because it may not be available (e.g., in sleep mode). Alternatively, the repo may include the wire-encoded sensor data packet inside the repo data it publishes and synchronizes via PSync. This saves the round-trip for others to retrieve the sensor data packet. After receiving the new sensor data, all repos insert the data to their local data store. The PSync partial sync producer running at the Pub-sub interface listens to the repo insert event and informs any subscribed consumers with the updated IBF and missing names.

D. Data Subscription

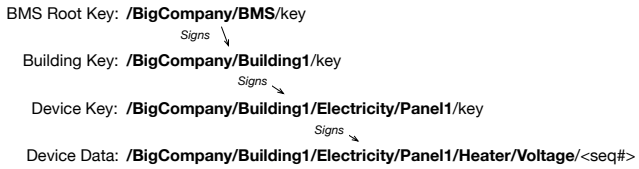
Data subscription in NDN-PS allows the consumer applications to receive updates efficiently from a subset of the data streams published in the pub-sub group. Behind the scene, the consumers use PSync Interest messages to retrieve the latest data name of each subscribed stream from the repos. Once they obtain the new data name with the latest sequence number, the consumers can decide whether to fetch the new data according to the application requirements (which corresponds to step ③ in Figure 5). The separation of the notification of names from the retrieval of actual data allows NDN-PS to accommodate different data consuming semantics (e.g., sequential fetching, latest-data first, random sampling, etc.) without forcing every consumer to receive all the data.

The subscription state of the consumer consists of two parts: (a) the list of name prefixes of the subscribed data streams and (b) the latest state of the whole dataset known by the consumer (which may not be up to date). In PSync, the subscription list is typically encoded as a Bloom Filter (BF) due to its space efficiency.[3] The state of the repo is represented using an Invertible Bloom Filter (IBF) computed over the latest names in the data streams stored at the repo, which is sent back to the consumers in PSync reply messages (see Figure 1). When a repo receives new data (either from the BMS panels or from other pub-sub repos via the PSync replication channel), it replaces the data name of the updated stream in the IBF with the latest one.

When requesting for updates in the subscribed data streams, the consumer sends its current state (including the BF-encoded subscription list and the previously received IBF) along with the request. To generate a PSync reply, the repo subtracts the received IBF from its own IBF to detect the streams that have been updated, filters the updated streams with the subscription list, and returns the latest data names of the subscribed streams to the consumer. Since the sequence number in the data name continuously increases, the name of the latest data can serve as an implicit notification of all the previous data published in the same stream. The reply also carries the repo's latest IBF, which is used by the consumer to replace its IBF and "advance" its data consumption state.

An important design benefit of using PSync in NDN-PS is that the consumer's subscription state is maintained by the consumer itself rather than stored in the pub-sub repos. This allows the repos to remain stateless about the consumer subscription information, which reduces the amount of state that the repos have to maintain. It also enables the consumers to retrieve updates over the PSync protocol from any of the repos that are synchronizing the same set of data streams (using PSync's full sync API) without worrying about loss of subscription state or having to wait for the consumer state synchronization among the repos. Multiple repos in the same pub-sub group use the same name prefix to receive PSync

[3]There can be other ways to encode the name prefixes, for example, a range [NP1, NP2] can efficiently represent all the name prefixes between two name prefixes NP1 and NP2 based on some ordering criteria.



(a) Sensor certification chain

(b) User device certification chain

(c) Pub-sub repo certification chain

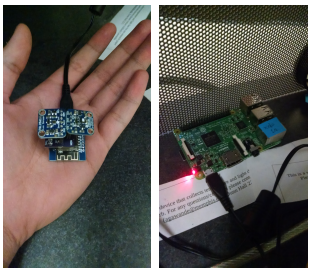
Fig. 7: BMS certification chain examples

Interests, and the network will always forward the consumers' PSync Interest to the nearest available repo.

E. Data Authentication

In NDN-PS, the public keys of all the entities (panels, user devices, and repos) are certified using a hierarchical trust model expressed with the names of the signing keys. Signatures generated by the trusted entities are verified by following the certification chain up to a common trust anchor or, eventually, the BMS root key. The key signing chain for the panels is aligned with the hierarchical structure of their physical locations in the enterprise buildings, as is shown in Figure 7a. The key signing chain for the user devices is instead aligned with the logical management hierarchy in the enterprise, which is shown in Figure 7b. The repo uses a different key for each pub-sub group it participates in, following the *principle of least privilege*. The repo key is signed by the pub-sub group key, which is further signed by the building key or the BMS root key (see Figure 7c), depending on whether the pub-sub group is located in a single building or spans across the campus.

The data publishing process requires the mutual authentication between the producers and the pub-sub repos: on one hand, the repos need to authenticate the sensor data before adding them to the local storage; on the other hand, the producers need to verify that the confirmation in the Repo Insertion process [17] comes from a legitimate repo in order to make sure the data is successfully archived in the pub-sub group. The consumers also need to authenticate the PSync replies from the repos and the sensor data fetched from the network. In addition, the repos need to authenticate each other during the PSync message exchanges. In a traditional TCP/IP-based pub-sub system, implementing such complicated authentication steps would require TLS channels between all communicating parties. In contrast, the data-centric security paradigm and the expressive naming in NDN enable a more powerful and elegant solution with a simple, hierarchical trust model that allows NDN nodes to authenticate any data produced on the network.



(a) ESP Sensor (b) Raspberry Pi

Fig. 8: Hardware in BMS implementation

IV. Implementation and Deployment

We developed a real BMS implementation to demonstrate the feasibility of our design. Our prototype consists of Raspberry Pi's and sensor devices placed in classrooms and offices.

Figure 8 shows our hardware. We assembled each sensor module using Wemos D1 mini, an ESP-8266 Wi-Fi chip board [20], a temperature and humidity sensor [21], a light sensor [22], and a custom interface shield, costing $24 in total. We use the ndn-cpp-lite library [23] for the ESP chip [24] on each sensor module to serve data for three different kinds of sensors: temperature, humidity, and light. The main motivation behind using a custom assembled sensor module is to keep the cost down as well as having greater control.

We use Raspberry Pi 3 Model B [25] with 1.2GHz 64-bit quad-core ARMv8 CPU and 1GB RAM to run the following software components: (a) a data collector that pulls data from the sensors, (b) a repo that uses a PSync Full Sync Producer to synchronize the collected data with each other, and (c) a PSync Partial Sync Producer that publishes data for subscribers (see Figure 9). The data collector program on each Pi collects data by sending interests to multiple sensor modules and inserts it into the repo. The repo validates the data based on the BMS trust hierarchy (Figure 10) before accepting it. We modified the NDN repo-ng [16] implementation to have a notification stream which the Full Sync Producer and Partial Sync Producer can subscribe to. Upon receiving a notification from the local repo, the Full Sync Producer propagates the changes to other repos. When the Full Sync Producer receives an update from another repo, it sends an insert command to the local repo. The local repo then fetches the data from the other repo. When the Partial Sync Producer receives a notification

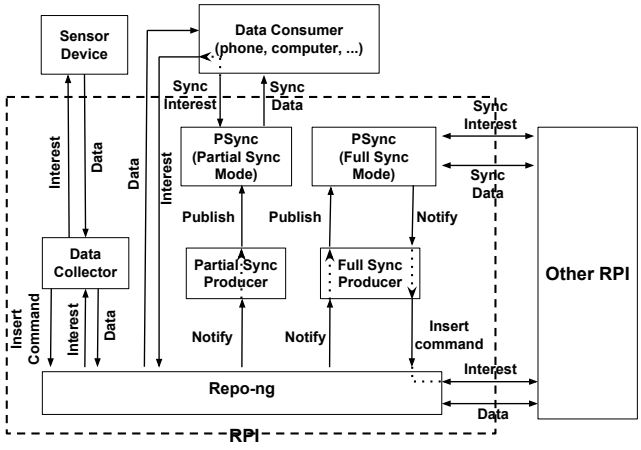


Fig. 9: Module interactions in implementation

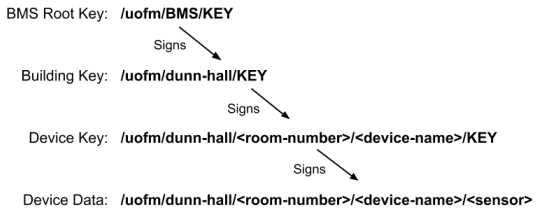


Fig. 10: Trust hierarchy in BMS implementation

from the repo, it notifies those consumers that have subscribed to the data. The consumers then fetch the data directly from the repo and validate it.

We have deployed the system in 30 classrooms and offices in Dunn Hall at the University of Memphis with a total of 6 Pi's and 30 sensor modules on the three floors (each Pi collects data from 5 sensor modules). We plan to extend the deployment to a larger scale as we gain more experience.

V. Discussion

A powerful design pattern in many BMS applications is to connect multiple data aggregators and filters to perform pipelined data analytics and event processing. This can be naturally implemented over NDN-PS by concatenating multiple pub-sub groups where some consumers in a pub-sub group also serve as the producers in other groups. While this paper only describes the communication mechanism within a single group, we envision that most real-world applications in BMS environments will require some form of interconnection of multiple pub-sub groups. For example, in a large enterprise campus the fine-grained raw data collected from the sensors in each building are usually consumed and processed by the data aggregation services deployed close to the panels and controllers. Then the digested data is fed into a different pub-sub group and consumed by the next-stage processing jobs that further aggregate the data across multiple streams (e.g., computing the total power consumption of the whole building by adding up the power measurements from each room). Multiple data aggregation layers can be chained together via NDN-PS to build a hierarchical data storage, with each layer presenting a view of the BMS system at a different granularity.

Note that, as an optimization, the data aggregation service can be integrated into the repo, which eliminates the need for deploying standalone data aggregation services that communicate with the repos over PSync as in the original design. It also allows the repos to drop the raw data points after processing them and store only the aggregated data.

A BMS data acquisition and analysis system can even go beyond simple aggregation and filtering by bridging multiple pub-sub groups to form arbitrary data flow graphs that can support complex data processing frameworks such as MapReduce [26]. Each node in the data flow graph runs a data processor that either filters and repackages the sensor data (*"map"*) or aggregates data from multiple inputs (*"reduce"*). The processing results are transferred to other nodes efficiently over NDN-PS. This enables the system to apply more advanced data querying and transformation techniques such as the Structured Query Language (SQL) by compiling high-level operators into a series of primitive MapReduce tasks. Such a system can be deployed across the enterprise BMS network to ingest data from multiple buildings and generate reports for the whole campus.

VI. Related Works

A. Pub-sub Support in traditional BMS

Many existing building management and IoT systems in the industry have provided basic pub-sub communication support. BACnet [27], a popular BMS protocol, allows the clients to subscribe to periodic or change-of-value notifications from the devices. ZigBee Cluster Library [28], the foundation for ZigBee Smart Energy and Home Automation profiles, also supports similar pub-sub operations through the attribute reporting mechanism. OPC Unified Architecture [29], another industrial machine-to-machine communication protocol, also provides subscription services for the clients to subscribe to value changes, events, and computed aggregates.

Most of those systems aim at providing direct push notifications from the producers to the consumers, which is suitable for closed-loop control and low-latency communication. However, there are a number of well-known issues with the push-style pub-sub mechanism in existing industrial BMS protocols. Take the popular BACnet protocol as an example. BACnet allows the devices to send change-of-value (COV) notifications to the clients that have subscribed to the COV events of certain device properties. This may potentially flood the network with notification messages when multiple related properties change at the same time due to some cascading effect (e.g., the HVAC system stops working, causing the temperatures of multiple rooms in the building to rise). Another common problem is that the clients may miss the push notification if they go offline, since they have no control of when the notification will be

generated.[4] Moreover, the client subscription information is kept as soft state by the devices and may be lost when the device restarts.

In contrast, NDN-PS is designed to support scalable pub-sub communication for enterprise BMS data acquisition applications that consume both realtime and historical data points. The subscription communication is consumer-driven so that the applications can fetch the data via PSync based on their own schedule. Storing and replicating the sensor data in the pub-sub repos lifts the burden of keeping historical data and handling consumer requests from the BMS panels. Since the consumers maintain their subscription information by themselves, they do not need to worry about losing their subscription state when the pub-sub repos restart or get replaced. Finally, NDN-PS utilizes NDN's built-in data-centric security mechanism to authenticate the sensor data, which provides better flexibility and efficiency compared to the traditional security solutions in BMS with physical/logical isolation (which prevents the applications from accessing the BMS data from different networks) and secured channels using shared keys (which requires establishing pair-wise secure channels).

B. Pub-sub Support in Cloud-based IoT

Modern IoT systems (including but not restricted to building management systems) are increasingly relying on cloud-based frameworks such as AWS IoT[5] and Google Cloud IoT[6] to perform device management and data processing. Those cloud-based IoT systems enable the sensors to publish their data to the remote cloud, where the data is stored and fed into other services also running in the cloud via a generic pub-sub pipeline designed and optimized for the data center environment. The decoupling of data acquisition and consumption is similar to what is proposed in NDN-PS and allows the IoT systems to support different types of data consumers (e.g., actuators, monitoring services, data analytical frameworks) at large scale.

The drawbacks of relying on cloud services for achieving IoT functionality have been articulated in our earlier paper [30], including the strong dependency on the availability and security of the cloud platforms and additional latency imposed by the indirection through the cloud. While the earlier paper describes a cloud-independent IoT system as an alternative solution for the "smart home" applications, this paper proposes a scalable pub-sub framework for building management system (which usually covers a larger scope than a typical smart home) that builds on top of the same spirit of leveraging local communication to achieve more efficient and secure data transportation among different components of the system.

[4]The BACnet standard does not require the devices to buffer the notifications if no acknowledgement is received from the client.

[5]https://aws.amazon.com/iot/

[6]https://cloud.google.com/solutions/iot/

C. Pub-sub over ICN

Several frameworks have been proposed so far for ICN architectures to support end-to-end pub-sub semantics. COPSS [31] achieves pub-sub communication using push-based multicast mechanism similar to PIM-SM. When publishing data, the publisher sends a publication message towards some Rendezvous Point (RP). The publication message contains the Content Descriptor (CD) of the published information. The CD is a hierarchical name that allows subscription at different granularities. COPSS adds the Subscription Table (ST) to the NDN forwarders to keep track of downstream data subscribers. The subscriber sends a subscription message towards the RP to establish a forwarding path. Forwarders along the path will record the subscribed CD and the downstream interface in the ST using bloom filters. The data from the publisher is then pushed to all subscribers following ST entries instead of PIT entries as in normal NDN forwarding.

iHEMS [32] modifies the NDN routers to implement persistent subscription using long-lived forwarding information. The subscribers send subscription requests that persist in the router's PIT for some time t. Any data published during that time t will be forwarded to the subscribers without consuming the PIT entry. The authors of iHEMS acknowledge that persistent PIT entry may affect the traffic control and flow balancing, but argue that the problem can be mitigated by choosing the persistence interval t very carefully. To support secure group communication, iHEMS proposes to encrypt confidential data with a group key shared among the publishers and subscribers. In addition, iHEMS relies on dedicated directory service to maintain the list of data names published in the network, through which the publishers and subscribers discover each other.

Both COPSS and iHEMS allow more than one data packet returned for each pending Interest over a single link, essentially breaking the flow balance principle of NDN [33]. Although this approach reduces the number of interests, it is susceptible to congestion and denial-of-service attacks.

PSIRP/PURSUIT [34] is an ICN architecture that provides native pub-sub support at the network layer. In PSIRP/PURSUIT, the data (or information) is identified by a pair of flat labels: the rendezvous identifier (RID) and the scope identifier (SID). To publish information, the publisher needs to choose the scope for the publication and create the RID for the publication. Then the publication message containing the RID and the SID is forwarded to the rendezvous node within the scope of SID, which will store and manage the RID. A subscriber first learns about the RID and SID, then issues the subscription request to the rendezvous point of that RID. A forwarding path is then created between the publisher and the subscriber (through the rendezvous point) and future information will be sent over this channel. The packet forwarding in PSIRP/PURSUIT can be implemented efficiently using MPLS-style label matching.

NDN-PS differs fundamentally from the above proposals in the following aspects: (1) it is built on top of NDN's Interest-

Data semantics and achieves efficient subscription without breaking the flow balance principle by aggregating multiple subscriptions into a single PSync Interest; (2) it is designed specifically for the BMS environments and takes into account the practical requirements such as hierarchical naming and trust management, data archiving and replication, efficient subscription to multiple data streams, and accommodation for different data consuming semantics, all of which are essential to the BMS applications.

HoPP [35] was recently proposed for pub-sub in ICN-based IoT networks. It uses Content Proxies (CP) to decouple producers and consumers similar to how Repos are used in NDN-PS. However, it introduces a Prefix Advertisement Message to set up FIB entries for the CPs, and a Name Advertisement Message to propagate data from producers to the CPs. In contrast, NDN-PS does not rely on intermediate nodes to process any special messages; the Sync and Repo protocols run on application nodes (consumers, producers and repos) using basic interest/data exchanges.

VII. Conclusion

In this paper we present NDN-PS, a distributed pub-sub communication framework for building management systems over NDN. NDN-PS extends our previous work on secure data acquisition in NDN-BMS to support data subscriptions from consumer applications running on different platforms and with interests in different data as well as different data consumption semantics. PSync, a new addition to NDN's data sync arsenal, enables NDN-PS to replicate collected sensor data across multiple repos to provide adequate redundancy and scalable data retrieval, and it allows data consumers to receive notifications of new sensor readings generated by multiple BMS panels. Last but not least, all sensor data readings, as well as all packets generated by the repos, are authenticated through cryptographic signatures and can be verified through a hierarchical trust model.[7]

NDN-PS is a comprehensive pub-sub design based on NDN's Interest-Data primitive. It retains the fundamental design principles of the NDN architecture such as flow balancing and hierarchical naming. This design exercise further confirms (a) the expressive power of naming in NDN, such as embedding the building hierarchy and sensor types in the prefixes of the pub-sub groups; (b) the usefulness of naming conventions, such as the use of sequence numbers to improve the efficiency of data synchronization; (c) the utility of NDN Sync protocols, which simplifies application design and supports asynchronous multi-party data sharing; and (d) the simplicity of data authentication using a hierarchical trust model based on hierarchical data naming and organizational and system relationships.

We will continue to analyze, evaluate and expand the prototype deployment at University of Memphis. One area we will explore is the management of the deployed testbed such as auto-configuration of the IoT devices and monitoring of their failures using NDN-based tools. Another area is edge computation and concatenation of pub-sub repos.

[7]Data confidentiality and access control can also be supported using data encryption, as is described in our previous work [10], or the more recent Name-based Access Control [36].

Acknowledgment

This work has been supported by the National Science Foundation, under awards CNS-1344495, CNS-1629769, CNS-1345318, and CNS-1629922.

References

[1] W. Shang, Y. Yu, R. Droms, and L. Zhang, "Challenges in IoT Networking via TCP/IP Architecture," NDN Project, Tech. Rep. NDN-0038, February 2016.

[2] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang, "Named Data Networking of Things," in *Proceedings of 1st IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2016.

[3] Y. Zhang, D. Raychadhuri, L. A. Grieco, E. Baccelli, J. Burke, R. Ravindran, G. Wang, B. Ahlgren, and O. Schelen, "Requirements and Challenges for IoT over ICN," Internet Engineering Task Force, Internet-Draft draft-zhang-icnrg-icniot-requirements-01, Apr. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-zhang-icnrg-icniot-requirements-01

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, 2009.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[6] A. Afanasyev, T. Refaei, L. Wang, and L. Zhang, "A Brief Introduction to Named Data Networking," in *IEEE MILCOM 2018*.

[7] NDN Project Team, "NDN Codebase Platform," http://named-data.net/codebase/platform/, 2016.

[8] Y. Yu, A. Afanasyev, D. Clark, kc claffy, V. Jacobson, and L. Zhang, "Schematizing Trust in Named Data Networking," in *Proceedings of the 2nd International Conference on Information-Centric Networking*, September 2015.

[9] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A Case for Stateful Forwarding Plane," *Computer Communications*, vol. 36, no. 7, pp. 779–791, Apr. 2013.

[10] W. Shang, Q. Ding, A. Marianantoni, J. Burke, and L. Zhang, "Securing Building Management Systems using Named Data Networking," *IEEE Network*, vol. 28, no. 3, pp. 50–56, May 2014.

[11] W. Shang, Y. Yu, L. Wang, A. Afanasyev, and L. Zhang, "A Survey of Distributed Dataset Synchronization in Named-Data Networking," NDN Project, Technical Report NDN-0053, May 2017.

[12] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking," in *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, Oct 2013.

[13] W. Fu, H. Ben Abraham, and P. Crowley, "Synchronizing Namespaces with Invertible Bloom Filters," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, May 2015.

[14] M. Zhang, V. Lehman, and L. Wang, "Scalable Name-based Data Synchronization for Named Data Networking," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, May 2017.

[15] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the Difference?: Efficient Set Reconciliation Without Prior Context," in *Proceedings of the ACM SIGCOMM 2011 Conference*, 2011.

[16] NDN Project Team, "repo-ng: Next generation of NDN repository," https://github.com/named-data/repo-ng, 2017.

[17] ——, "Basic Repo Insertion Protocol," http://redmine.named-data.net/projects/repo-ng/wiki/Basic_Repo_Insertion_Protocol, 2014.

[18] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moi-seenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto, C. Fan, C. Papadopoulos, D. Pesavento, G. Grassi, G. Pau, H. Zhang, T. Song, H. Yuan, H. B. Abraham, P. Crowley, S. O. Amin, V. Lehman, and L. Wang, "NFD Developers Guide," NDN Project, Tech. Rep. NDN-0021, Revision 7, oct 2016.

[19] A. Afanasyev, C. Yi, L. Wang, B. Zhang, and L. Zhang, "SNAMP: Secure namespace mapping to scale NDN forwarding," in *Proceedings of 18th IEEE Global Internet Symposium (GI 2015)*, April 2015.
[20] "Wemos D1 mini," https://wiki.wemos.cc/products:d1:d1_mini.
[21] "Temperature and humidity sensor," https://www.adafruit.com/product/1899.
[22] "Light sensor," https://www.adafruit.com/product/439.
[23] NDN Project Team, "ndn-cpp-lite GitHub," https://github.com/named-data/mini-ndn.
[24] ——, "esp8266ndn GitHub," https://github.com/yoursunny/esp8266ndn/.
[25] raspberrypi.org, "Raspberry Pi 3 Model B," https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.
[26] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, ser. OSDI'04. Berkeley, CA, USA: USENIX Association, 2004, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251254.1251264
[27] BACnet Advocacy Group, "BACnet Homepage," http://www.bacnet.org/, 2016.
[28] ZigBee Alliance, "ZigBee Cluster Library Specification," http://www.zigbee.org/download/standards-zigbee-cluster-library/, May 2012.
[29] OPC Foundation, "OPC Unified Architecture," https://opcfoundation.org/about/opc-technologies/opc-ua/, 2016.
[30] W. Shang, Z. Wang, A. Afanasyev, J. Burke, and L. Zhang, "Breaking Out of the Cloud: Local Trust Management and Rendezvous in Named Data Networking of Things," in *Proceedings of IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, April 2017.
[31] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. K. Ramakrishnan, "COPSS: An Efficient Content Oriented Publish/Subscribe System," in *Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Oct 2011.
[32] J. Zhang, Q. Li, and E. M. Schooler, "iHEMS: An information-centric approach to secure home energy management," in *Smart Grid Communications (SmartGridComm), 2012 IEEE Third International Conference on*, Nov 2012.
[33] NDN Project Team, "NDN Protocol Design Principles," http://named-data.net/project/ndn-design-principles/, 2016.
[34] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From PSIRP to PURSUIT," in *Broadband Communications, Networks, and Systems*. Springer, 2010, pp. 1–13.
[35] C. Gündoğan, P. Kietzmann, T. C. Schmidt, and M. Wählisch, "HoPP: Robust and Resilient Publish-Subscribe for an Information-Centric Internet of Things," in *Proceedings of the IEEE Conference on Local Computer Networks (LCN)*, 2018.
[36] Y. Yu, A. Afanasyev, and L. Zhang, "Name-Based Access Control," NDN Project, Tech. Rep. NDN-0034, Revision 2, Jan. 2016.