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 paper we present ndnBMS-PS, a distributed pub-sub communication framework for building management systems (BMS), an important area of IoT, over the Named Data Networking (NDN) architecture. ndnBMS-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. ndnBMS-PS also provides data authentication by utilizing NDN's security building blocks. 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 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 may spread across multiple buildings on an enterprise campus, potentially with tens of thousands of sensors installed on the premises to monitor electricity, lighting, temperature, humidity, and many other environmental measurements. Recent years have witnessed an extensive effort by both industry and research communities to adapt or enhance the current TCP/IP architecture to support BMS, yet many challenges remain. For example, sensing devices may have intermittent connectivity and use duty cycles to save energy, while IP communication assumes a devices being "always on" model, and the associated channel-based security requires the two communicating end points 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), including expressive naming, data-centric security, and in-network caching, can be utilized to directly address the identified challenges and enable one 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 **ndnBMS-PS** for BMS environments over the NDN architecture [4]–[6]. We have implemented ndnBMS-PS on the NDN platform [7] and evaluated it using our emulation tool Mini-NDN. This 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*. Note that (1) the number of sensors and consumers can potentially be very large; (2) each data-consuming application may be interested in a *different subset* of the sensing data; and (3) sensors and consumers may not be online at the same time. To address these data communication challenges, ndnBMS-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. ndnBMS-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 differs from TCP/IP's channel-based security model in a fundamental way: it ensures that the security protection stays with the data itself, without dependency on the secure channels (e.g., a TLS/DTLS

session) between communicating entities.

In the remainder of this paper, we first give some background on this work (Section II), and then present our system design (Section III) and evaluation (Section IV). We also review the literature of pub-sub systems proposed for other ICN architectures (Section V). Finally we conclude the paper and address future work in Section VI.

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 physically and/or logically 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 measurement data 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 different sensing data. 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 online when new data is produced.
- Different applications may have different semantics in data consumption. 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 semantically meaningful 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). 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 forwarding table, and the measurement from previous forwarding decisions. When an Interest meets a Data packet with a matching name, the Data packet is returned to the consumer.

Every 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 from either 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 a BMS system is made of a large number of sensors, actuators, and controllers. Second, NDN's data-centric security mechanism inherently secures every produced piece of data, 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 sensors and improves the scalability of the BMS data communication.

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 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, ndnBMS-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, 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 remotely analogous 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 names from anywhere the matching data items are found, since the security is attached to the data instead of data 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 communicating 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, producers in a Sync group publish data that are identified by each 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's name to its latest sequence number. The ChronoSync protocol disseminates the digest of this list via multicast so that everyone in the Sync group can detect any inconsistency of its local state and retrieve newly produced data 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. 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 ndnBMS-PS. Similar to ChronoSync, PSync assumes data names from each data stream contain a name prefix and a monotonically increasing sequence number. Different from ChronoSync, PSync uses an IBF to encode the set of the latest data name from all streams, this use of IBF allows PSync to extract differences between two participants more efficiently than ChronoSync. The consumers can subscribe to any subset of the data streams and receive the latest data generated in those streams. Each consumer maintains both its own list of subscribed data stream prefixes (called subscription list), which is efficiently encoded by a Bloom filter, and the latest IBF it has received from producers.

[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.

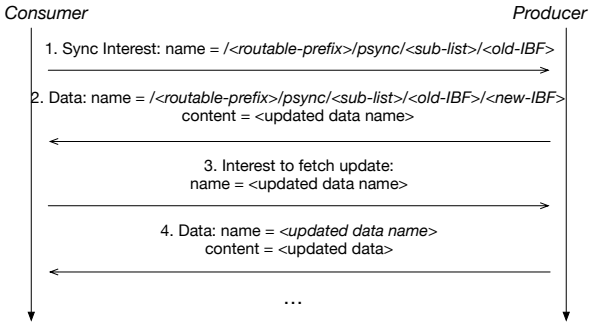


Fig. 1: PSync message exchanges between consumer and producer (<sub-list> is the consumer's subscription list encoded in Bloom filter, while <old-IBF> and <new-IBF> are the producer's previous and current dataset state encoded in IBF)

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 updated upon all new data production. When the producer receives a PSync Interest from a consumer which contains both the subscription list and the consumer's IBF, it can filter out the changes relevant to the consumer, and send back the corresponding IBF. Because PSync includes consumers' subscription information and producer's dataset state explicitly in the Interest/Data exchange, it minimizes the soft-state information kept by the producer.

In addition to the pub-sub functionality, PSync can also be used to support full synchronization among a group of participants. Therefore, if one sets up multiple data repositories (repos) to collect sensing data, the repos can run PSyns to synchronize their dataset, so that consumers can retrieve updates from any of the repos that host the same sensor dataset, enabling a multi-repo pub-sub framework in ndnBMS-PS.

III. System Design

ndnBMS-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 ndnBMS-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 have no 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 ndnBMS-PS design aims to achieve the following goals:

- Scalability: the framework should support a large number of sensing data streams and consumers with arbitrary subscription relations.
- Availability: the framework should provide redundancy and automatic failover to enable producers publishing new data and consumers fetching 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, and support data encryptions as needed for content confidentiality. [2]

B. Design Overview

ndnBMS-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 connected to the sensor. To overcome intermittent connectivity and resource (e.g., CPU, storage, and energy) limitations, sensors publish their data into 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. 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. ndnBMS-PS 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.[3]

All data packets in ndnBMS-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 ndnBMS-PS.

ndnBMS-PS can support 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 repos are a core component in the ndnBMS-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 a few dozens of data streams may need only two or three repos to support adequate data replication, 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 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.

[2]The ndnBMS-PS framework can make use of name-based access control as described in [17].

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

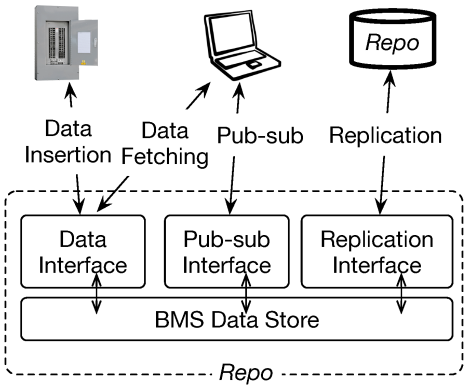


Fig. 2: System modules inside the repo

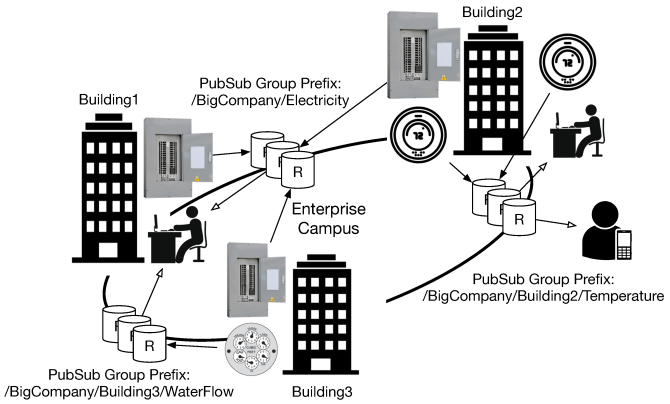


Fig. 3: Deployment of three pub-sub groups on an enterprise campus network that serve different types of BMS data: electricity, temperature, and water flow

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

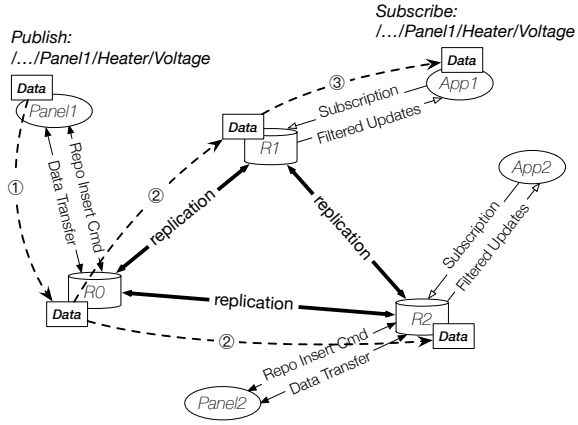


Fig. 5: Data flow in a pub-sub group

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` 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. ndnBMS-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.

There can be a number of means to collect BMS data points into the repos. In ndnBMS-PS, the data producers use the *Repo Insertion Protocol* [18] 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 [19] 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 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

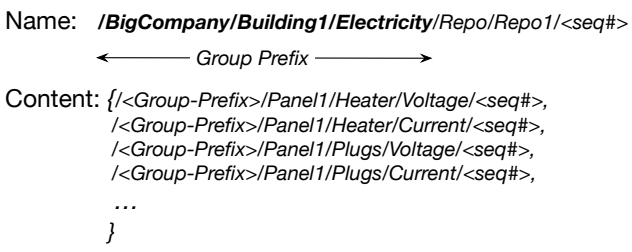


Fig. 6: Example of repo data published through PSync's full sync API for replication among repos

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* [20] pointing to the repo's unicast name prefix in order to retrieve the sensor data packets from that repo, to avoid the Interests reaching the data producer directly. 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 ndnBMS-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 ndnBMS-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 encoded as a Bloom Filter (BF) due to its space efficiency.[4] 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) in 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 carries the repo's latest IBF, which is used by the consumer to replace its IBF and "advance" its data consumption state.

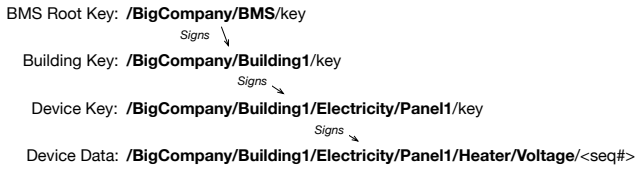
An important design benefit of using PSync in ndnBMS-PS is that the consumer's subscription state is maintained by the consumer itself rather than stored in the pub-sub repos. This allows the repos to remain stateless about the consumer subscription information, which reduces the amount of state that the repos have to maintain. It also enables the consumers to retrieve updates over the PSync protocol from any of the repos that are synchronized for 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 Interests, and the network will always forward the consumers' PSync Interest to the nearest available repo.

E. Data Authentication

In ndnBMS-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

[4]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

process [18] 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.

IV. Evaluation in Mini-NDN

We implemented the proposed system on the NDN platform [7] and evaluated it using our emulation tool Mini-NDN [21]. Our experiment topology (Figure 8) is based on the UC Berkeley campus core network [22], which has two core routers connected to border routers and every edge router connected to both core routers for robustness. Each edge router is connected to a number of repos and consumers. The link speed between routers is 1Gbps and propagation delay is uniformly distributed between 10 and 30 microseconds. To emulate unreliable wireless links, we made 20% of the links between consumers and edge routers lossy with a loss rate of 1% (some experiments have a higher loss rate). Every repo collects data from a number of producers (BMS panels), each generating a stream of data with random inter-arrival times

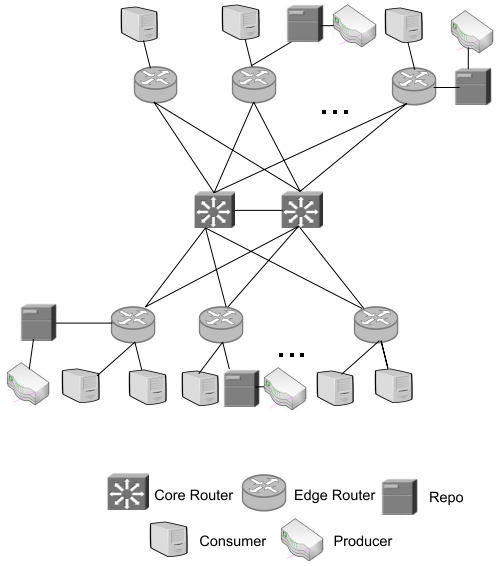


Fig. 8: Emulation evaluation topology for BMS system

between 1 and 5 minutes. Each data consumer subscribes to a fixed number of randomly chosen set of data streams.

To evaluate how well the system scales with the number of monitored buildings and end users, our experiments include varying number of data streams, data consumers and repos. We focus on the *data fetching delay*, i.e., the delay from when a data point is produced to when the data is obtained by a consumer that has subscribed to the data, which includes up to four delay components: (a) the delay for the producer to insert the data into the repo, (b) the delay to propagate the data from the producer's repo to the repo with which the consumer is sync'ed (if the two repos are different), (c) the delay to inform the consumer of the new data name, and (d) the delay for the consumer to fetch the data. In the first part, the producers use the Repo Insertion Protocol that takes 1.5 RTT to insert a Data packet into a repo, which is typically constant in our emulation environment. Therefore our experiments measure only the latter three parts of the delay. Below we present our experiment results.

A. Number of Data Streams per Repo

In this experiment, we run five repos and vary the number of data streams per repo from 500 to 1500. We then measure the data fetching delay across 20 consumers, each subscribing to 200 data streams, over a 10-minute time period. We present our results using the Tukey boxplot, i.e., the two ends of the whiskers represent the lowest data point within 1.5 IQR of the lower quartile and the highest data point within 1.5 IQR of the upper quartile, respectively.

We can see in Figure 9 that (1) the lowest data fetching delays are slightly over 9 ms, which correspond to the cases where the consumer subscribes to the same repo into which the producer inserts data; (2) the median delay increases only from 0.05 second to 0.06 second while the number of streams grows from 500 to 1500 per repo; and (3) the 75th-percentile delay is below 0.15 second and the highest whisker endpoint

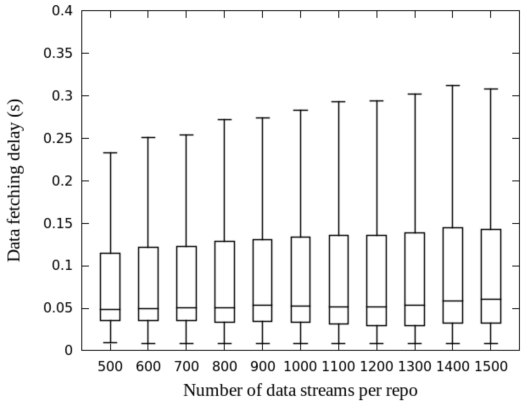


Fig. 9: Data delay vs. number of streams per repo (5 repos, 20 consumers, 1% link loss rate)

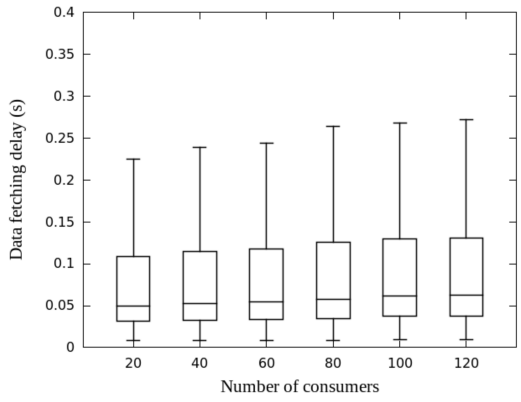


Fig. 11: Data delay vs. number of consumers (10 repos, 1000 data streams/repo, 1% link loss rate)

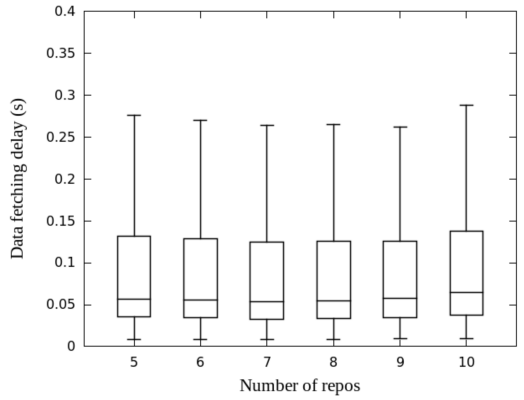


Fig. 10: Data delay vs. number of repos (1000 data streams/repo, 1% link loss rate)

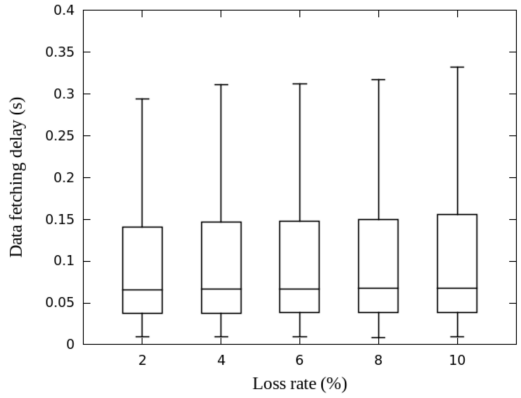


Fig. 12: Data delay under different loss rates (10 repos, 1000 data streams/repo, 120 consumers)

is around 0.32 second for 1400 streams per repo. These results demonstrate that the system scales very well when the overall data production rate increases.

B. Number of Repos

We varied the number of repos from 5 to 10 while keeping the other parameters fixed (e.g., 1000 data streams per repo and 200 subscriptions per consumer) in the second experiment. Note that we also maintain a fixed ratio between consumers and repos here, i.e., 12 consumers/repo. One can observe from Figure 10 that *the data fetching delay remains stable as the number of repos increases*, suggesting that we can scale out our system to store more data streams and serve consumers by adding more repos.

C. Number of Consumers

In the third experiment, we varied the number of consumers from 20 to 120, while maintaining 10 repos and 1000 data streams per repo. Figure 11 suggests that *the data fetching delay does not increase significantly even though we increase the number of consumers by a factor of 6*. This is due to two reasons: (1) the consumers interact with the repos using the PSync protocol, which does not maintain per-consumer state and uses IBF to do efficient set difference operations; and (2) the consumer-repo delay and its variations are much smaller than those in the repos' data synchronization process. In our current prototype, the pub-sub repo is implemented as a single-thread process that has to process the consumer requests sequentially. We believe the performance difference in these experiments can be further improved once we change the repo's PSync module to be multi-threaded.

D. Link Loss Rate

In the final experiment, we varied the link loss rate from 2% to 10%, while running 10 repos with 1000 data streams per repo and 120 consumers. Note that we set 20% of the consumer links to be lossy. If the requested data is lost, a consumer retransmits the Interest up to 3 times if it does not get the data. The results in Figure 12 show that *the higher loss rates did not have much impact on the data fetching delay*. This is because the lost data is already cached at the edge router and can be easily recovered in the next retry.

V. Related Work

Several frameworks have been proposed so far for ICN architectures to support end-to-end pub-sub semantics.

COPSS [23] 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 [24] 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 [25]. Although this approach reduces the number of interests, it is susceptible to congestion and denial-of-service attacks.

PSIRP/PURSUIT [26] 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.

ndnBMS-PS differs fundamentally from the above proposals in the following aspects: (1) it is built on top of NDN's Interest-Data exchange 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 [27] 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 ndnBMS-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, ndnBMS-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.

VI. Conclusion

In this paper we present ndnBMS-PS, a distributed pub-sub communication framework for building management systems over NDN. ndnBMS-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 ndnBMS-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.[5]

ndnBMS-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.

Acknowledgment

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

[5]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 [28].

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/.

[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 *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 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] Z. Zhang, Y. Yu, S. K. Ramani, A. Afanasyev, and L. Zhang, "NAC: Automating access control via Named Data," in *Proc. of MILCOM*, Oct. 2018.

[18] NDN Project Team, "Basic Repo Insertion Protocol," http://redmine.named-data.net/projects/repo-ng/wiki/Basic_Repo_Insertion_Protocol, 2014.

[19] 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 Developer's Guide," NDN Project, Tech. Rep. NDN-0021, Revision 10, July 2018.

[20] 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.

[21] NDN Project Team, "Mini-NDN Website," http://minindn.memphis.edu.

[22] University of California, Berkeley, "Campus network core topology," https://nettools.net.berkeley.edu/pubtools/legacy/net/netinfo/newmaps/campus-topology.pdf, 2016.

[23] 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.

[24] J. Zhang, Q. Li, and E. M. Schooler, "iHEMS: An information-centric approach to secure home energy management," in *2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)*, Nov 2012.

[25] NDN Project Team, "NDN Protocol Design Principles," http://named-data.net/project/ndn-design-principles/.

[26] 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.

[27] 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.

[28] Y. Yu, A. Afanasyev, and L. Zhang, "Name-Based Access Control," NDN Project, Tech. Rep. NDN-0034, Revision 2, Jan. 2016.