Breaking out of the Cloud: Local Trust Management and Rendezvous in Named Data Networking of Things

Wentao Shang
UCLA
Los Angeles, CA 90095
wentao@cs.ucla.edu

Zhehao Wang
UCLA REMAP
Los Angeles, CA 90095
zhehao@remap.ucla.edu

Alexander Afanasyev
UCLA
Los Angeles, CA 90095
aa@cs.ucla.edu

Jeff Burke
UCLA REMAP
Los Angeles, CA 90095
jburke@remap.ucla.edu

Lixia Zhang
UCLA
Los Angeles, CA 90095
lixia@cs.ucla.edu

ABSTRACT

Many emerging IoT approaches depend on cloud services to facilitate interoperation of devices and services within them, even when all the communicating entities reside in the same local environment, as in many "smart home" applications. While such designs offer a straightforward way to implement IoT applications using today's TCP/IP protocol stack, they also introduce dependencies on external connectivity and services that are unnecessary and often brittle. This paper uses the design of an IoT-enabled home entertainment application, dubbed Flow, to demonstrate how the Named Data Networking (NDN) architecture enables cloud-independent IoT applications. NDN enables local trust management and rendezvous service, which play a foundational role in realizing other IoT services. By employing application-defined naming rather than host-based addressing at the network layer, and securing data directly, NDN enables straightforward and robust implementation of these two core functions for IoT networks without cloud connectivity. At the same time, NDN-based IoT designs can employ cloud services to complement local system capabilities. After describing the design and implementation of Flow, together with a discussion on preliminary generalization of the design, as an evaluation the paper conducts a brief thought exercise of how Flow could be realized using two popular IoT frameworks, Amazon's AWS IoT service and the Apple HomeKit framework, and compares that with the real implementation over NDN.

CCS CONCEPTS

•Networks →Network architectures; Network protocol design; •Computer systems organization →Embedded and cyber-physical systems;

KEYWORDS

Named-Data Networking, Internet-of-Things, Cloud-independence

IoTDI 2017, Pittsburgh, PA, USA

DOI: 10.1145/3054977.3054993

ACM Reference format:
Wentao Shang, Zhehao Wang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. 2017. Breaking out of the Cloud: Local Trust Management and Rendezvous in Named Data Networking of Things. In *Proceedings of The 2nd ACM/IEEE International Conference on Internet-of-Things Design and Implementation, Pittsburgh, PA, USA, April 18-21, 2017 (IoTDI 2017),* 11 pages.
DOI: 10.1145/3054977.3054993

1 INTRODUCTION

Internet-of-Things (IoT) technologies are being rapidly adopted in the consumer electronics market. There has been increasing deployment of "smart" devices in the home environment to create interactive, human-in-the-loop applications and services. In addition to their use in traditional home automation systems, IoT technologies have also been applied to home entertainment–for example, with wireless inertial sensors used to track user body movement in sports games. Combined with emerging virtual and augmented reality technologies, IoT offers the promise of immersive and interactive experience for end-users. Common requirements for such systems include:

- Integration of heterogeneous devices and services from different vendors;
- Interactive user experience that emphasize real-time feedback loops;
- Easy installation and configuration; and
- Security protection, due to tight integration with the home network.

Many IoT frameworks and ecosystems have been proposed over the last few years to facilitate the development of more sophisticated applications like these. They typically provide a similar set of framework-level services, including user and device authentication and authorization, device and service discovery, device onboarding and management, publish-subscribe messaging, and remote access. On top of these common services, IoT developers can further design and implement application-specific functionality. Fig. 1 shows a common hierarchical architecture of IoT services, where "named entities" refer to users, devices, and applications that require trust management and utilize rendezvous services to get interconnected into a coherent home IoT system.

Existing home IoT systems often depend on cloud-based services provided by device vendors and/or service providers to implement

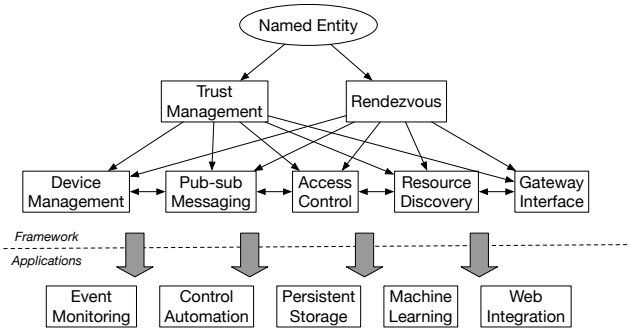


Figure 1: Hierarchical architecture of IoT services.

the framework services and application functions needed in a local IoT environment.[1] Such dependency on the cloud, for what could be inherently local functionality, introduces potential negative impacts:

- The home IoT system requires connectivity to the cloud to manage local devices and users; the user cannot install new devices or authorize new users if connectivity to the cloud is lost.
- With cloud-based services, control and resource access commands go through the cloud, which acts as the rendezvous point, even when the command issuer resides in the same local network as the target device. This introduces additional delays that may hurt interactive applications.
- Cloud-based services expose data from the home environment to external parties, introducing potential security and privacy risks.
- A simple error in the cloud can unnecessarily bring down services to a large number of IoT systems.

In this paper, we present an alternative approach by leveraging the Named Data Networking (NDN) architecture [3, 12]. We focus on enabling IoT functions to be achieved locally, with support for optional cloud services. Building upon our previous work in [4], this paper identifies **trust management** and **rendezvous** as two foundational building blocks of all IoT services, as shown in Figure 1. It develops a specific design and implementation using NDN primitives.

In Section 2, we first give an overview of representative IoT ecosystems and outline their dependencies on the cloud. We then describe in more detail these two foundational IoT services, trust management and rendezvous, which provide the basis for bootstrapping other services. We show that both of them can be supported more efficiently in IoT networks under the Named Data Networking architecture (Section 3). The key idea behind such a cloud-independent architecture design is to leverage NDN's name-based forwarding to directly operate on well-established names in the local context. This enables straightforward solutions for trust management, via schematized trust, and rendezvous, via distributed dataset synchronization.

[1]Cloud services also support features, such as voice recognition, which can be difficult to achieve locally; such advanced services are not critical to the overall design of IoT systems and thus not our focus in this paper.

The rest of the paper is organized as follows. Sections 4 and 5 present the design and implementation of Flow, an IoT-augmented home entertainment application, as a realization of the proposed IoT approach using NDN. While this implementation focuses on an interactive home entertainment application, we believe that our approach should be generally applicable to traditional home automation systems and many other IoT subdomains. To provide a qualitative evaluation, Section 6 presents a brief comparison between Flow and how a similar application could be approached using AWS IoT and Apple HomeKit services, two popular TCP/IP-based IoT frameworks.

2 BACKGROUND

2.1 Existing IoT Ecosystems

Many current IoT architectures and frameworks, such as *Bluetooth* [2], *ZigBee* [14], and *Google Thread* [9], have primarily focused on achieving device-to-device connectivity and interoperability. For example, Bluetooth defines a seven-layer protocol stack and how two devices can connect to each other over either point-to-point or mesh networks, discover the other's capabilities through application layer profiles, and exchange application data called attributes. Designed as silos, these architectures cannot interoperate with each other without a special gateway or translator deployed in between, limiting development and innovation of IoT technologies.

As IoT systems become more powerful and more complex, there is a growing demand for more comprehensive application-layer frameworks that can integrate and manage different types of devices across different communication technologies, enable more intelligent application logic involving a large number of devices, and provide simplified user experience in operating such systems. In this subsection, we briefly review a few popular IoT application frameworks that emerged in the last three years.

AllJoyn[2] (2013[3]) and *IoTivity*[4] (2015) are generic IoT application frameworks that aim to bridge various IoT transport technologies and provide a common language for applications and services. They both offer standardized interfaces for common IoT services such as device management, resource discovery, application-layer messaging, access control, etc.. While they started with an emphasis on proximal (i.e., local-area) communications, they also define gateway interfaces for connecting to external services both locally and in the cloud. As lower-level frameworks, they do not mandate specific solutions for trust management and rendezvous, but provide common protocol interfaces for developers to design and implement applications and services that run either locally or remotely in the cloud.

AWS IoT[5] (2015), *Google Weave*[6] (2015), *Azure IoT Suite*[7] (2015), and *Samsung SmartThings*[8] (2014) are examples of *cloud-centric* IoT ecosystems that are bound to specific cloud service providers and their implementations of all common IoT services, from authentication and device management to data processing and application

[2]https://allseenalliance.org
[3]The number in parentheses indicates the year of initial public announcement.
[4]https://www.iotivity.org
[5]https://aws.amazon.com/iot/
[6]https://developers.google.com/weave/
[7]https://azure.microsoft.com/en-us/suites/iot-suite/
[8]https://www.smartthings.com

hosting. Through tight integration with the cloud, those ecosystems provide simple and centralized solutions for trust management and rendezvous. To make devices and applications securely discover and communicate with each other, the user typically registers all local IoT devices and applications with the remote cloud service. The cloud then handles the tasks of device authentication and catalog maintenance of the functionality available in the local environment. Another benefit of such cloud-centric architectures is easy integration with advanced services requiring data access or processing power unavailable in the local environment, such as search, voice recognition, and data analytics needed by large-scale IoT systems including precision agriculture and industrial control.

A notable recent trend among such cloud-centric architectures is to move certain IoT applications and services into the local network and execute them on a local *hub* in order to tolerate intermittent cloud connectivity. For example, Amazon, Google, and Samsung all created their own home hub devices to connect local IoT devices and perform simple home automation tasks. The recently announced *AWS Greengrass*[9] (2016) even allows part of the AWS IoT control plane to be hosted on a local server, essentially creating a private cloud service close to the IoT deployment. However, local data (e.g., sensor reading, device status, system config, etc.) still need to be synchronized to the remote cloud to be consumed by cloud-hosted services.

Apple HomeKit[10] (2014) can be viewed as an example of this recent trend. HomeKit is designed as an IoT framework specifically for home automation applications that interact with Apple-certified IoT devices. Different from the pure cloud-centric ecosystems mentioned above, HomeKit's design enables and encourages local communications. The framework stores the home configuration in a local database which is synchronized across the devices that are hosting HomeKit apps (e.g., a user's iPhone or Apple TV). After obtaining permissions from the user, HomeKit apps on any of those devices can access the database to discover and communicate with the home devices directly over the local network. Hence, the rendezvous service is provided locally through database synchronization so that each device has complete knowledge of the home network. However, HomeKit still relies on Apple's iCloud service for device authentication and trust management: each user and each new device must be authenticated through Apple and obtain iCloud IDs first. In addition, the replication of the home configuration database across user's Apple devices is done indirectly via iCloud. Moreover, remote access from outside the home network requires tunneling the messages through iCloud to a local hub (e.g., an Apple TV) inside the home network.

Table 1 summarizes the approaches taken by different IoT architectures and ecosystems in providing the trust management and rendezvous services. Note that all of them are built on top of the TCP/IP protocol stack, therefore have to provide the mapping services to resolve named entities to specific IP addresses, either remotely in the cloud DNS service or locally via some zero-config protocol such as mDNS.

[9]https://aws.amazon.com/greengrass/
[10]http://www.apple.com/ios/home/

Table 1: Comparison of existing IoT architectures and ecosystems.

	Trust management	Rendezvous
Bluetooth	P2P peering	None (P2P)
ZigBee	Pre-shared master key	Broadcast
Google Thread	L2 network-wide key	None
AllJoyn / IoTivity	Interface only	Interface only
AWS IoT	Cloud service	Cloud service
Google Weave	Cloud service	Cloud service
Azure IoT Suite	Cloud service	Cloud service
Samsung SmartThings	Cloud service	Cloud service
Apple HomeKit	Cloud service	Database sync

2.2 Named Data Networking of Things

Named Data Networking (NDN) is a future Internet architecture under development. NDN replaces host-addressed IP packets with named data as the new narrow waist of the "hourglass" protocol stack. Each data object has a hierarchical name that serves as the unique identifier within the application context where the data is published and consumed. To request a data object, one sends an *Interest* packet carrying a prefix of the data name. NDN forwarders forward the Interest packet towards where the data may be found. Each forwarder along the path records the Interest and its incoming interface in a local *Pending Interest Table (PIT)*. When a matching *Data* packet is encountered, either in a forwarder's cache or from the original producer, the Data packet is returned to the requester by following the reverse path of the Interest as recorded in the PITs of the nodes along the path; those nodes may store a copy of the Data packet in their local caches after forwarding, to be used to satisfy future requests for the same data. The Data packet carries a cryptographic signature generated by its producer, together with the name of the signing key. This allows the data consumer to verify the provenance of received data regardless of its source.

In our previous paper [4], we described that, by naming and securing the *things* and data directly at the network layer, how NDN is able to provide a more straightforward and secure solution to IoT networking as compared to TCP/IP:

- The Interest-Data exchange model in NDN closely resembles the RESTful protocols such as HTTP and CoAP that are widely adopted in today's IoT systems.
- Name-based forwarding simplifies the network stack by removing the extra step of resolving application names to network identifiers (e.g., IP and MAC addresses).
- Data-centric security is more efficient and IoT-friendly than the channel-based or physical/logical isolation-based alternatives.
- Ubiquitous in-network data caching helps improve the efficiency of information dissemination, especially for resource constrained IoT environments.

In [4] we also described various higher-level protocols built on top of NDN to achieve framework functionalities such as bootstrapping and discovery, trust management, access control, multi-party communication, and global integration. The major differences between NDN- and IP-based IoT architectures are illustrated in Fig. 2. We refer readers to [4] for a complete discussion.

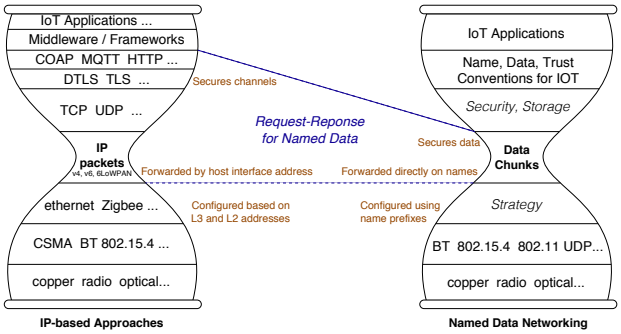


Figure 2: Protocol stack comparison of NDN and TCP/IP.

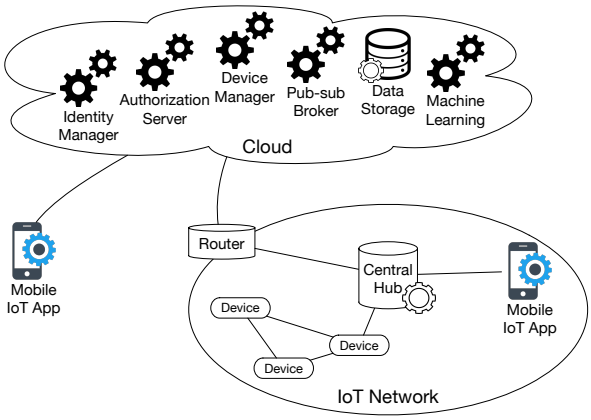


Figure 3: Typical cloud-centric IoT architecture

Expanding on our previous work, our goal in this paper is to demonstrate the design and implementation of a specific IoT system based on NDN. In particular, we will illustrate how to leverage naming within the local context to achieve trust management and rendezvous in an IoT network, which further enables a variety of local IoT applications and services.

3 CLOUD-INDEPENDENT IOT OVER NDN

In this section, we briefly review the IoT services commonly supported by remote cloud infrastructure. Then we discuss how the same functionality can be achieved more efficiently by leveraging local trust management and rendezvous over the NDN architecture.

3.1 IoT and the cloud

IoT applications and services often require a set of common services to be provided by the application-layer frameworks, as shown in Fig. 1:

- *Identity management, authentication and authorization*, making up the trust management subsystem for users, devices and services and, in connection-oriented models, access control;
- *Rendezvous and resource discovery*, enabling applications to find the devices and services they need;
- *Device management*, to handle onboarding, monitoring, configuration changes, software upgrades, etc. for constituent devices;
- *Application data messaging*, which supports data exchange through mechanisms including publish-subscribe (pub-sub), streaming, etc.; and
- *Gateways to external networks*, bridging a local IoT network to external services such as data storage and analytics, as well as the public Internet (including mobile devices) in general.

As we discussed in Section 2.1, most IoT ecosystems today rely on the cloud to implement part or all of those framework services. For example, in AWS IoT, cloud services play several critical roles that cover all of the above aspects of an IoT platform. AWS acts as an identity provider, issuing security credentials for users and devices; it provides authorization services, whereby device certificate issued by AWS contains the resource access policies prescribed by the user; and it handles device management and rendezvous, where all devices in the IoT system register with and report to AWS, which also collects state for all devices and makes it available to applications. All messages between IoT devices and services are tunneled through the cloud for pub-sub dispatching. AWS can host applications that consume the IoT data and trigger actions when certain events happen. Users can access the IoT network from public Internet, the messages are also tunneled through AWS via the message brokers.

A typical cloud-centric IoT architecture is depicted in Fig. 3. The cloud provides a convenient "central hub" for managing the interconnections of all the devices in an IoT system. By transferring the control to the cloud, a cloud-centric IoT architecture simplifies the system configuration and management tasks for users, but sacrifices the opportunity of using local communication to achieve higher reliability and efficiency. It requires local data traverse the global network unnecessarily. In "human-in-the-loop" application scenarios, control messages from the user's smartphone are first routed to the cloud for authentication and logging before being forwarded to the IoT device. This extra latency may negatively impact real-time, interactive applications, such as IoT-augmented home entertainment experiences. Furthermore, the exposure of local IoT devices and private data to the global network can be a potential security risk, making all users depend on the cloud service providers to protect their data and enforce access control properly. The connectivity to the remote servers in the cloud can also be a single point of failure. For example, a user of a home IoT systems cannot install or re-configure devices at home or access home network from public Internet if the connectivity from her home to the cloud is down, or if the cloud service experiences an outage [1].

3.2 Rethinking IoT Service Architecture

Among the IoT framework services shown in Figure 1, two play foundational roles:

(1) **Trust management**, which authenticates IoT system entities (users, devices, and applications), and authorizes how they may interact with each other; and

(2) **Rendezvous**, which provides a means for different entities to discover and interconnect with each other, over either local or global networks.

Independent from the lower layer addressing scheme, trust management and rendezvous at the application layer are built on the concept of **named entities**. The application-layer names are either specified by the users or auto-generated by the devices and applications. Those names serve as the entities' identifiers in an IoT system and allow those entities to refer to each other and interoperate. To enable authentication, the identity names must be associated with some form of credentials such as user passwords or public key certificates. Therefore application-layer trust policies can be composed in terms of names meaningful at the application layer, rather than low-level device identifiers and raw key materials. Named entities also provide the basis for rendezvous: an entity can discover other entities on the same network by learning their names.

Other IoT services can be bootstrapped from these two core services. As an example, device management is based on discovery (via rendezvous) and mutual trust establishment to enable communication between devices and managing services; resource discovery is a natural extension of rendezvous, enhanced by the ability to verify a resource's origin after the discovery; pub-sub messaging and external gateways both require the interplay of rendezvous and authentication functions. Note that those services may be interdependent; together they form a framework layer, on top of which IoT developers can create high-level applications and services.

A major benefit of hosting IoT services and applications in the cloud is simplified user experience, especially if the system has to deal with the complexity of registering devices with some local controller, connecting devices with local and remote applications, and managing security credentials – such operations can be too complicated for ordinary home users to handle. However, we believe the above-mentioned complexity is an artifact of existing implementations use of the TCP/IP protocol stack, rather than an intrinsic nature of IoT systems in general. As we illustrate in Sections 4 and 5, the usability problem can indeed be addressed in an NDN-based implementation, which directly utilizes human-friendly, hierarchical naming in supporting rendezvous and trust management services. Human-friendly naming enables intuitive understanding of the trust relationship among devices and applications for ordinary users, and a hierarchical naming structure provides proper contexts to facilitate applications in expressing and exploring the organization of the IoT system, further enabling automated trust management and rendezvous tools.

Unfortunately, the TCP/IP protocol stack offers IP addresses as the first-order names for devices and services,[11] accompanied by per-device or per-service public keys for (D)TLS authentication. Semantically meaningful names, such as URLs, reside at application-layer only, and have to be resolved to addresses or port numbers when the applications access data or services via the network. Numeric names are straightforward for machines to operate on, but contain no semantic meaning that can be leveraged to make trust decisions, to support rendezvous, or to provide intuitive understanding for human users and application developers.

The NDN architecture offers an elegant solution to the naming problem at the network layer. It allows the IoT services to be described locally and in a decentralized way without sacrificing security, functionality, flexibility, and usability, as described in the next subsection.

3.3 Achieving Local IoT Functions with NDN

The NDN architecture bootstraps local IoT communication by naming the entities in the context of the local IoT network. Instead of obtaining their identities from cloud service providers, the entities in the IoT network create local identities associated with asymmetric cryptographic keys that are certified by a local trust anchor. The identity certificates are all published as Data packets in the local NDN network under the identity namespace. The trust anchor is typically a root key created by the owner of the IoT system and stored securely on a local authentication server such as a control hub or a TPM-equipped smartphone.

Note that a local entity may also have other identities for external communications. For example, a device may have a manufacture-issued identifier that is used for signing the device status report or retrieving software/firmware updates; users may also have public identities (e.g., OpenIDs) that can be used for initial authentication when new users are added to the system. The practice of using different identifiers for different purposes is aligned with the *principle of least privilege*.

After the local identities are created, the NDN-IoT architecture leverages two powerful tools to provide the two fundamental services: using *schematized trust* [10] for local trust management, and *distributed sync* (e.g., [13]) for local rendezvous.

3.3.1 Trust management. IoT system trust policies can be formally described using *trust schema* to specify the relations between data names and signing key names using a domain-specific language designed for pattern matching on NDN names. The NDN software platform provides tools that automatically sign and verify the Data packets according to a given trust schema, which are pre-defined and can be integrated into the applications through client libraries.[12] The trust schema of IoT applications can list the local trust anchors and specify the trust relationship between local data and key names, which can be enforced within the scope of the local network without the intervention of cloud services. If needed, the trust schema can also include external trust anchors (e.g., the public keys of a trusted cloud provider) according to the application requirements.

3.3.2 Rendezvous. NDN Sync protocols such as *ChronoSync* [13] allow multiple devices to synchronize the namespace of the shared dataset without relying on central servers. This mechanism provides a convenient rendezvous solution where the application prefixes and device identities are published under a well-known local namespace and synchronized across multiple devices in the IoT network. The synchronization protocol can benefit from local multicast communication for efficiency, it may also work in a peer-to-peer

[11] For example, it is a common practice in cloud computing to assign one or more *virtual IPs (VIPs)* to identify cloud services.

[12] Trust-schema can also be modified dynamically and redistributed to relevant entities in the same way as any other named, signed data.

fashion when multicast is not supported by the constrained devices.[13] For large-scale IoT systems, multiple sync groups can be created under separate namespaces to insulate independent subsystems.

Through support for decentralized trust management and rendezvous, an NDN-based IoT architecture enables cloud-independence while providing essential IoT services. Applications can still benefit from cloud services whenever needed, such as archiving data, performing complex data analysis jobs, or accessing advanced services like search and voice recognition. In fact, NDN simplifies the integration with the external networks by using the universal Interest-Data exchange primitive for data communication. Cloud services become an optional component, rather than a required piece of the architecture.

Automated local trust management and rendezvous should also improve user experience by limiting the only manual step of configuration during device setup to a local one–registering the device with the local trust anchor, e.g. by scanning the barcode of a newly purchased device.Once the trust is established and a local certificate is generated, the IoT devices and applications can communicate with each other, exchange useful information, or discover new devices and applications without human intervention or cloud connectivity.

4 FLOW: A HOME ENTERTAINMENT EXPERIENCE OVER NDN

In this section, we describe the design of Flow, a home entertainment experience that leverages NDN to realize a cloud-independent, IoT-supported application, and conclude by summarizing the components of a generalized NDN-IoT framework developed based on this design. Flow is a prototype of a multi-user "exploration game", in which participants navigate and interact with a virtual world rendered in a game engine using a combination of inputs:

(1) *Indoor positioning*: participants' positions in physical space, detected by indoor positioning (person tracking), modify the virtual landscape;
(2) *Wearable sensing*: participants directly control orientation of the environment's virtual camera using gyroscopes connected to microcontrollers, which can be worn or carried;
(3) *Mobile phone interface*: participants interact with the virtual environment through controls on their smartphone, for example to share social media images in the virtual environment.

In addition to various types of IoT devices and the game engine, the system on which Flow is built also includes an *authentication server* (AS) that performs local trust management. The AS can be implemented as an app on the owner's smartphone, or a service daemon on a dedicated control hub (e.g., the home router).

Figure 4 shows a typical deployment scenario of Flow in a home network. NDN interconnectivity between different components is supported over Ethernet and Wi-Fi, through the home Wi-Fi router in a hub-and-spoke topology. Sensor devices with limited networking capability (e.g., the gyroscope in Fig. 4) may be bridged via a helper device. We assume all devices can reach each other over NDN, which is trivial in a hub-and-spoke topology.[14]

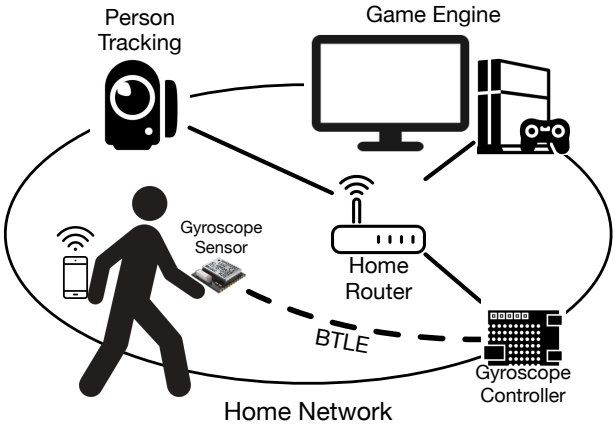


Figure 4: Typical deployment of the Flow home entertainment experience.

4.1 Naming and Identity

In Flow, data from the IoT *things* used by the application are named using three namespaces:

- *Application namespace*: a local namespace for publishing and accessing application data, e.g., gyroscope readings needed to control the environment;
- *Device namespace:* a local namespace for publishing device identity certificates and metadata;[15]
- *Manufacturer namespace*: a global namespace created by the IoT device vendors and for trust bootstrapping.

Fig. 5 shows an example of the Flow namespace. In addition to these three namespaces which name devices, things and their data, note the *discovery* branch under the local root prefix, which is used for device rendezvous and for application prefix discovery. Details of its functionality are described later.

The device and application namespaces both have as their root a home prefix that is either context-dependent (e.g., "`/AliceHome`" as in Fig. 5) or globally reachable (e.g., "`/att/ucla/dorm1/301`").

The application namespace starts with a unique instance name (e.g., "`/AliceHome/flow1`") created by the application at installation. Data produced by each component is named under an application label configured by the developer (e.g. "`/AliceHome/flow1/tracking1`"). The application label also contains a metadata subtree containing the device name that serves this data (e.g. "`/AliceHome/flow1/tracking1/_meta/_device`").

Devices publish their local identity certificates under the device namespace (e.g., "`/AliceHome/devices`"). They also publish metadata (profile) information in the "`_meta/_app`" branch under the device identity prefix, including, for example, the application data

[13] See [6] for detailed discussion on the issue with multicast in IoT.

[14] A routing protocol may be required if a sensor mesh topology is deployed inside the home network.

[15] Device metadata could include information about devices and their capabilities as well as bindings to application names.

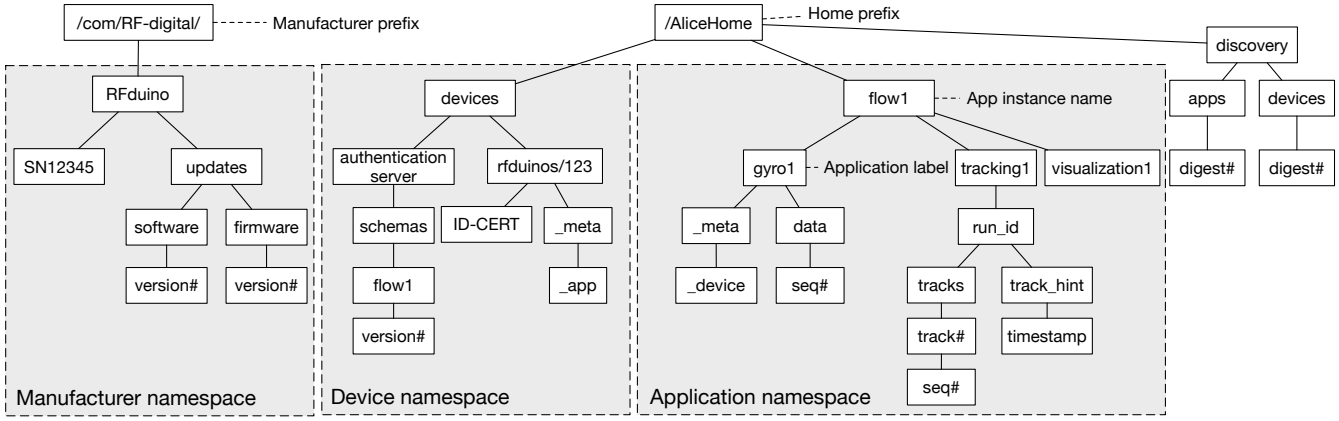


Figure 5: Example namespace within the home environment where Flow is deployed.

prefixes they use to publish. The device namespace of an authentication server also contains the trust schema of currently active applications. Schema and trust relationship details are described later in this section.

The manufacturer namespace falls under vendor-specific prefixes that are independent from the home network's local prefix. We envision that manufacturers will have globally unique names for their products used during bootstrapping, over-the-air updates, and similar processes. Manufacturers publish their own certificates under this globally unique prefix so that the devices can authenticate the data coming from the vendors such as software/firmware updates and service notifications.[16] In the research prototype of Flow, all devices are configured with vendor-provided identity names and profiles in their initial provisioning, before being connected to the home network. These are used for device onboarding.

4.2 Trust Management

Flow demonstrates a multi-step process for trusting new devices in a home IoT network and enabling their data to be used in an application. First, a device is assigned a device-level name and added to the trust hierarchy for things in the home. Then, it is configured with one or more application-level names for its data, and these names are added to application trust hierarchies. Finally, the device is configured to respond to requests in application namespaces.

The authentication server acts as the trust anchor. It can be coordinated with but does not depend on a remote cloud services. While the devices and users may have public identities outside the home environment, they all need to obtain local identities that are certified by the authentication server (AS) before they can start interacting with other local entities.[17]

The process of establishing a trust relationship between a new device and the home through the AS is similar to the Bluetooth pairing process. To bootstrap a new device, the user—or a configuration application on her behalf—provides a shared secret and a local device name. The shared secret may be a device barcode, identity communicated by NFC, or simply a PIN number. The AS sends a command Interest to the device, signed using a key derived from the shared secret, to ask that it generate a public/private key pair associated with the device's new name on the local network. The device replies with a Data packet containing an identity certificate request, also signed by the shared secret. The AS generates the identity certificate based on this request. The device, now part of the trust hierarchy, can advertise its services or participate in an application over the local network. This process is illustrated in Fig. 6. If the device has been issued a public identity certificate by its vendor, the AS may optionally authenticate its public identity, e.g., by asking the device to sign an AS-generated challenge.

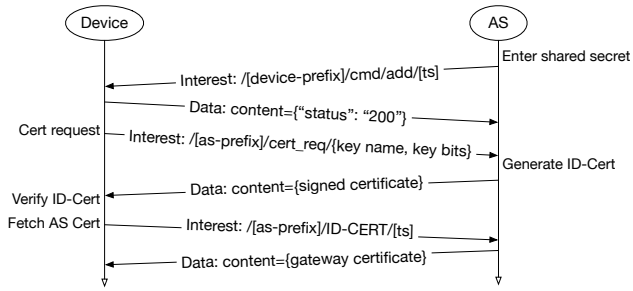


Figure 6: Bootstrap trust relationship for new device.

Applications in Flow are "installed" in a similar way to devices, with the AS signing both identity certificates and trust schema for the application. The application's trust schema expresses *what device identities are authorized to publish under what application prefixes* and is published as a normal Data object on the local NDN network. For example, in Fig. 5 "`flow1`" is a specific Flow instance and "`schema`" branch contains the trust schema of this instance. The schema name includes a monotonic version number at the end, so when there is a change in the schema a newer version is published.

[16] Reachability of data in this prefix is not addressed here but can be accomplished through encapsulation supported by the home router, for example.

[17] The public identities may be used to assist the onboarding process, but will not be required for local communication once the initial configuration has finished. For example, a new user can authenticate with the AS using her public identity (e.g., OpenID or Google/Facebook account) before creating her local identity that is used solely by the home environment.

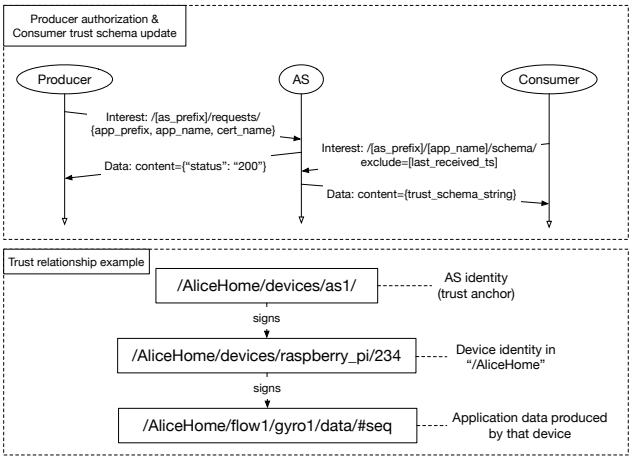


Figure 7: Schematized trust between producers and consumers.

The technical details of how to specify a trust schema are described in [10].

When a device that produces data is installed, it sends a command Interest to the AS that includes the application prefix it intends to publish under and its own local identity. If the request to publish data in the home network is granted, the AS will update the trust schema with the authentication rules for data published by this device. The rule binds a device identity with the application prefixes it's authorized to publish under.[18] Schematized trust enables fine-grained control over what devices can publish what data for which application instances. Consumer devices fetch the latest trust schema over the network via NDN and follow the rules to authenticate the data packets published in the network. The producer authorization process, as well as an example of the resulting trust relationship, is shown in Fig. 7, in which the AS signs a device identity, and the device signs a piece of application data it publishes.

4.3 Rendezvous

Flow also demonstrates a name-based, distributed rendezvous mechanism for devices and applications to discover each other over NDN. As described in the previous section, the key idea is to synchronize the set of device and application names (called the *rendezvous dataset*) across the devices in the network that are interested in learning about them. The synchronization process utilizes the decentralized and serverless ChronoSync [13] protocol to effectively synchronize prefixes of active devices and applications under "`/AliceHome/discovery/devices`" and "`/AliceHome/discovery/apps`" sub-namespaces, respectively.

When a new device is installed in the Flow system, it joins both the "devices" and "apps" sync groups and announces its local identity and application prefixes in the rendezvous dataset, which is propagated via ChronoSync across the network. Applications running on each device look up the local copy of the rendezvous dataset directly using a common API. Once an application obtains the name prefix of the target device or application, it can follow the namespace structure described in 4.1 to construct Interests for fetching the certificates and metadata, enabling it to bootstrap high-level service communication.

[18]This binding addresses potential collision in application labels–for example, by default the AS does not authorize a second device to publish under an application namespace claimed by another.

4.4 Generalizing IoT functionality in NDN

Through the design of Flow, we explored how to use NDN to provide the functionality discussed in Section 3.1 without reliance on any cloud services, and generalized it in a framework called *NDN-IoT*, which provides the following features besides trust management and rendezvous services:

- *Device management*: In addition to device onboarding and bootstrapping, NDN-IoT also supports device monitoring by publishing device status (e.g., power level, CPU/memory usage, software/firmware versions, etc.) periodically. The information can be visualized by a user-friendly dashboard application running on the user's smartphone or computer which sends out Interests to fetch the status of interested devices either periodically or on demand.
- *Pub-sub messaging*: The pub-sub messaging protocol in NDN-IoT currently supports two types of application data naming schemes. If the publisher names the data using continuous *sequence numbers* (e.g. "`/AliceHome/flow1/gyro1/data/[0,1,2,...]`"), the subscribers pipeline their Interests, following a built-in congestion control algorithm, to fetch the data using full names that include the trailing sequence numbers. If the data name contains *timestamp* (e.g. "`/AliceHome/devices/pc1/_status/20170104T1130`") that cannot be predicted, the subscribers issue Interests using the prefixes of the data (excluding the timestamp component) periodically or immediately after the previous data is received, in order to keep outstanding PIT entries in the network. Some of the authors are involved in an ongoing research project that investigates the use of sync protocols for implementing pub-sub semantics in IoT environment [7].
- *Gateways to external network and services*: The local IoT system can request data from the public Internet using globally reachable names. Meanwhile, its local data can be made available to the public Internet by publishing under a globally reachable name prefix directly or having a gateway service that generates data named under a globally reachable prefix to encapsulate data from the local system (e.g. "`/att/ucla/dorm1/301/flow1/gyro1/data/1`" → "`/AliceHome/flow1/gyro1/data/1`").
- *Access control*: NDN-IoT provides access control capability using the *Name-based Access Control* framework [11], which encrypts the content of the data and passes the decryption key to authorized consumers only (encrypted separately with each consumer's public key). The detailed specification of the access control protocol for NDN-IoT is currently under development.

5 IMPLEMENTATION

We have implemented the NDN-IoT framework and a prototype of the Flow entertainment system to verify the design introduced

in the previous section. The implementation follows a modular structure and differentiates between the framework-level services that we believe are common to many home IoT systems and the functionality that supports Flow-specific application logic. The NDN-IoT framework is implemented as a set of libraries using JavaScript, Python, C# and C++. It is built on top of the NDN Common Client Libraries [8], providing further abstractions to facilitate application development in a home IoT environment.

In our prototype, each of the Flow application components is implemented as the following:

(1) *Indoor positioning*: We use OpenPTrack[19], a multi-camera person tracking system. The NDN producer for OpenPTrack[20] (written in C++) publishes the position of each person at a 30Hz rate, along with lower rate metadata about active tracks.
(2) *Wearable sensing*: We use an RFduino 22301 with gyroscope MPU6050 attached to provide virtual camera control. The RFduino cannot perform asynchronous signing operations quickly enough, so we introduced a Raspberry Pi controller as a gateway for bridging RFduino to the NDN home network. The data exchanged between RFduino and Raspberry Pi is signed with a shared secret key negotiated after Bluetooth pairing. The Raspberry Pi generates a public/private key pair on behalf of the RFduino to be associated with the RFduino's device identity. The RFduino runs a minimum NDN producer, implemented with the ndn-cpp-lite library[21], which generates data at roughly 2Hz rate. When new data is generated, the RFduino pushes the data (signed by the pre-negotiated shared secret) to the Raspberry Pi controller over the Bluetooth LE channel. The controller receives the data, repackages the data and signs the data using RFduino's private key, and then publishes the data on the home network. The RFduino data publishing process is shown in Fig. 8.
(3) *Mobile phone interface*: We employ an Android phone that loads a control webpage (written in NDN.JS [5]) in a mobile browser to interact with the virtual environment. The phone sends out two types of command Interests: the first one matches an OpenPTrack track ID with that of the mobile, and the second one drops an image onto the virtual environment where the user's avatar is standing. ID matching is introduced so that the visualization knows the location of the user's avatar (identified by a track ID) when an image drop command Interest is issued by the same user (identified by the mobile's ID).
(4) *Visualization*: We use the Unity3D[22] game engine for visualization. The game engine runs C# NDN data consumers that receive person tracking and virtual camera control data, and a producer that receives image dropping command Interests from the mobile Web interface.

The implementations for both NDN-IoT framework and Flow application are available online[23]. We installed two instances of the Flow application testbed at UCLA and Huawei. Fig. 9 shows a diagram of the system and its message flows after all devices are bootstrapped with an authentication server, which in our installation is running on another Raspberry Pi device.

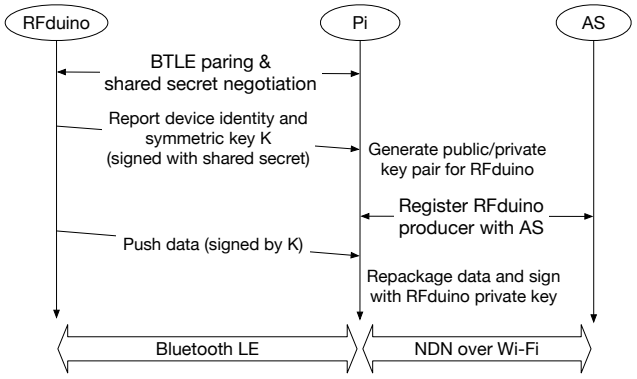


Figure 8: RFduino data publishing with assistance of Raspberry Pi controller

6 EVALUATION

In this section, we qualitatively compare Flow and the NDN-IoT framework with conceptual implementations of a similar gaming system over AWS IoT and Apple HomeKit (using TCP/IP architecture). The goal of this side-by-side comparison is to highlight the differences between the proposed architecture and the current practice in the industry, and to articulate how the cloud-independent IoT system can benefit from the NDN architecture. Fig. 10 shows three different designs of home entertainment system over AWS IoT, Apple HomeKit, and NDN-IoT, respectively.

As shown in Fig. 10a, all the local devices must be certified by the AWS IoT Registry service in order to join the IoT system. Application data generated by the person tracking device and the user smartphone go through the pub-sub message brokers (e.g., via MQTT) in the cloud, which then routes the messages to the game engine back in the local network, or other AWS services in the cloud according to user-defined rules. There is a significant amount of work performed by the AWS infrastructure to map user-defined device names to the endpoints of the underlying TLS tunnels, maintain state about device configuration and latest status, present a consistent view of the local IoT network to all applications and services both locally and in the cloud, manage the pub-sub relationship between data producers and consumers, and enforce authentication and access control policies during message forwarding.

Being the least cloud-dependent among the existing IoT ecosystems, HomeKit limits an application like Flow's dependency on the cloud to two key services only: authenticating devices and users during the bootstrap phase, and synchronizing the home configuration database across multiple devices. The game engine device in Fig. 10b can look up its local copy of that database to discover the person tracking device and gyroscope sensor in the same local network. There is a separate auto-configuration process based on mDNS to discover network addresses and set up secured connections among devices over local Wi-Fi or Ethernet. Although its

[19] http://openptrack.org/about/
[20] https://github.com/OpenPTrack/ndn-opt/
[21] https://github.com/named-data/ndn-cpp/
[22] https://unity3d.com
[23] https://github.com/remap/ndn-flow

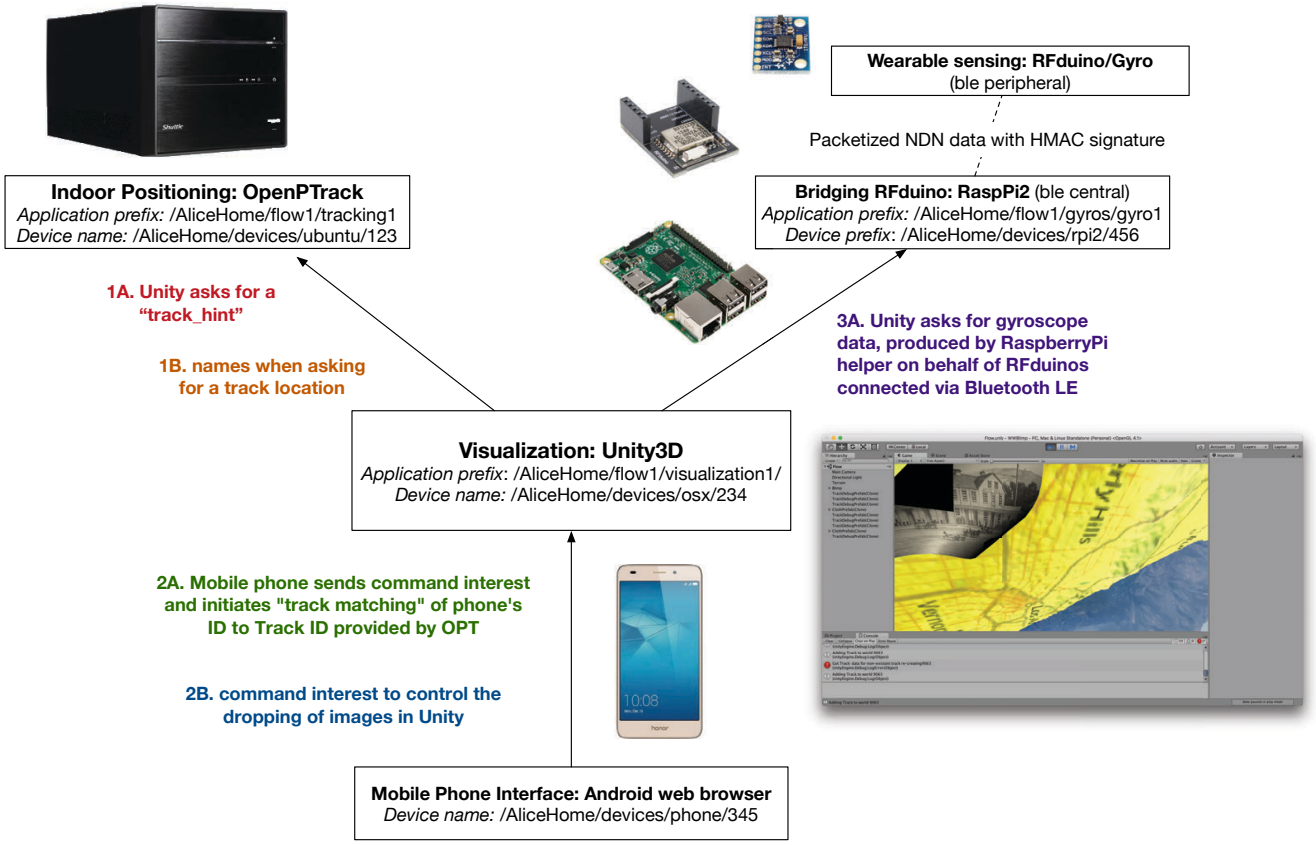
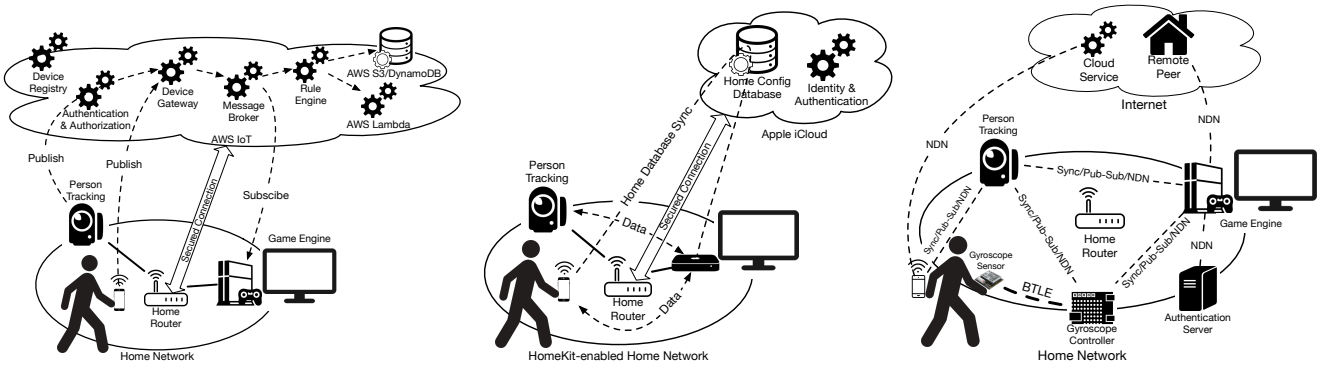


Figure 9: Application components and message flows in Flow



(a) A conceptual implementation over AWS IoT (b) A conceptual implementation over HomeKit (c) The actual implementation over NDN-IoT

Figure 10: Comparison of IoT-enabled home entertainment systems over three different ecosystems.

reliance on cloud services is significantly reduced as compared to the AWS-based system, the HomeKit-based implementation still suffers from usability problem when the home network is disconnected. Without cloud connectivity, the user would not be able to add new devices or invite new users. Further, configuration changes made to the existing devices would not be synchronized to other local devices, leading to an inconsistent view of the system among IoT applications and services.

Compared to the two cloud-based systems, the Flow application and NDN-IoT framework depicted in Fig. 10c show the following advantages:

- Users can add or remove IoT devices (e.g., upgrading to a new game control stick) at any time without requiring connectivity to the public Internet.
- New devices are discoverable by existing applications and services as soon as it obtains certification for its local identity and starts advertising itself in the local "discovery" sync group.
- Messages exchanged between the game controller and the game engine are forwarded over the local network (typically relayed through the home Wi-Fi access point) and experience minimum delay, improving real-time gaming experience.
- In the case of online gaming, the data generated by remote players is retrieved in the same way as the local data is consumed. The Interest-Data exchange between players in different geo-locations can be forwarded along the shortest path through the Internet between the players' home networks, without relaying through the cloud service.
- Like all other NDN applications, the Flow game system secures the data itself at production, which gives the applications full control over how their data is protected, rather than depending on remote services in the cloud to execute their security policies correctly.

7 CONCLUSION AND FUTURE WORK

The Internet-of-Things bring a revolution to local communications as much as in global communication. However the intrinsic limitations in the address-based TCP/IP architecture, and its lack of security support in particular, make cloud-based IoT implementations a "path of least resistance". As IoT becomes an essential part of our everyday life, the implications of its cloud-dependency must be addressed. As we were finalizing this paper, a benign operational error brought down Amazon cloud service [1] and impacted a large number of IoT services as well as other applications. This incident reminds us that cloud services are not immune to failures, further underscoring the value of this work. Robust solutions are also likely to expand the market to the long tail of dwellings (from houseboats to motorhomes), and to "the next billion" in the emerging markets with challenging communications conditions, where the cloud may be accessible but not always reliable.

It is with these opportunities in mind that we have developed the cloud-independent design described in this paper. We have shown a specific design for the two fundamental functions in IoT, trust management and rendezvous, in a way that is independent from, but can readily incorporate, cloud-based services. As a proof of evidence, we have implemented a running home entertainment application. We built our solution on top of Named Data Networking architecture, assuring resiliency of local "smart" IoT features in face of external failures. At the heart of the design are application-defined hierarchically named and secured data packets exchanged at the networking level, from which trust management and rendezvous can be built.

Significant challenges remain for the IoT and NDN research communities, many of which have been mentioned earlier. For example, designs for easy-to-use, encryption-based access control in networks with heterogeneous computational capabilities of individual devices remain to be developed, as do designs for global access to the manufacturer namespaces in our examples. Application development paradigms for Named Data Networking of Things, and NDN applications in general, largely remain as an open area of future research.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under award CNS-1345318 and CNS-1455850, and by Futurewei Technologies.

REFERENCES

[1] Amazon Web Services. 2017. Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region. https://aws.amazon.com/message/41926/. (March 2017).
[2] Bluetooth Special Interest Group (SIG). 2014. Bluetooth Specification Version 4.2. (Dec 2014). Available at https://www.bluetooth.com.
[3] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *Proceedings of the 5th ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*. 1–12.
[4] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang. 2016. Named Data Networking of Things (Invited Paper). In *Proceedings of the 1st IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 117–128.
[5] Wentao Shang, J. Thompson, M. Cherkaoui, J. Burkey, and Lixia Zhang. 2013. NDN.JS: A JavaScript client library for Named Data Networking. In *Proceedings of IEEE INFOCOMM 2013 NOMEN Workshop*. 399–404.
[6] Wentao Shang, Yingdi Yu, Ralph Droms, and Lixia Zhang. 2016. *Challenges in IoT Networking via TCP/IP Architecture*. Technical Report NDN-0038. NDN Project.
[7] Wentao Shang, Minsheng Zhang, Alexander Afanasyev, Jeff Burke, Lan Wang, and Lixia Zhang. 2017. Publish-Subscribe Communication in Building Management Systems over Named Data Networking. (2017). Manuscript submitted to ACM TCPS Special Issue on Internet of Things.
[8] Jeff Thompson and Jeff Burke. 2014. *NDN Common Client Libraries*. Technical Report NDN-0024, Revision 1. NDN Project.
[9] Thread Group. 2015. Thread Stack Fundamentals. White Paper. (July 2015). Available at http://threadgroup.org/.
[10] Yingdi Yu, Alexander Afanasyev, David Clark, kc claffy, Van Jacobson, and Lixia Zhang. 2015. Schematizing Trust in Named Data Networking. In *Proceedings of the 2nd ACM International Conference on Information-Centric Networking (ICN)*. 177–186.
[11] Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. 2016. *Name-Based Access Control*. Technical Report NDN-0034, Revision 2. NDN Project.
[12] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *ACM SIGCOMM Computer Communication Review (CCR)* 44, 3 (July 2014), 66–73.
[13] Zhenkai Zhu and A. Afanasyev. 2013. Let's ChronoSync: Decentralized Dataset State Synchronization in Named Data Networking. In *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP)*. 1–10.
[14] ZigBee Alliance. 2012. ZigBee Specification. ZigBee Document 053474r20. (September 2012). Available at http://www.zigbee.org.