Named Data Networking of Things and Example Application "Flow"

Zhehao Wang, Jeff Burke

***Abstract*—Many emerging IoT approaches depend on cloud services to facilitate interoperation of devices and services within them, even when the primary need for communication is local in scope, as in many "smart home" applictions. While such designs offer a convenient way to implement IoT applications using today's TCP/IP Internet architecture, they also introduce dependencies between applications and Internet connectivity that are unnecessary and often brittle. This paper uses the design of an IoT-enabled home entertainment experience to demonstrate how the Named Data Networking (NDN) architecture enables cloud-independent IoT applications. It does so by enabling local trust management and rendezvous, 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 with or without cloud connectivity. At the same time, NDN-based IoT designs can also employ cloud services to complement local system capabilities. After describing the motivation, design, and preliminary generalization of the driver application, the paper concludes with a brief comparison with how it would be achieved using two popular IoT frameworks, Amazon's AWS IoT service and the Apple HomeKit framework.**

I. Introduction

Internet-of-Things (IoT) technologies are being rapidly adopted in the consumer electronics market. 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 new 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 loop;
- 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 management, publish-subscribe messaging, and remote access. 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 organized into a coherent home IoT system.

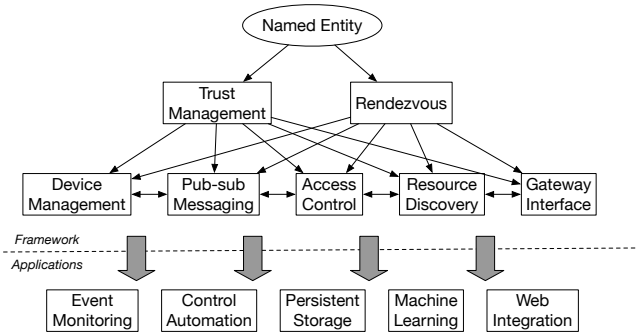

Fig. 1: Hierarchical architecture of IoT services.

The authors' cite:iotdi-2017 paper described an NDN approach [1], [2], which builds on our previous work in [3] and focuses on foundational functions of *trust management* and *rendezvous*. [1] While in the same paper the authors used Flow, an IoT-augmented home entertainment experience, to illustrate the high-level concepts, this report describes the design and implementation of Flow and its supporting libraries in more detail.

In Section II, we first give an introduction of Flow application and its requirements. Generalizing from its requirements we also propose an NDN-IoT framework, a set of IoT libraries built to support applications like Flow. In Sections III and IV, we present the design and implementation details of Flow. We conclude with a.

II. Flow: overview

In this section, we first describe an overview of the Flow application, and then the design goal of this application.

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;

[1]For background study, conceptual comparison between our approach and existing ones, please refer to this paper as well.

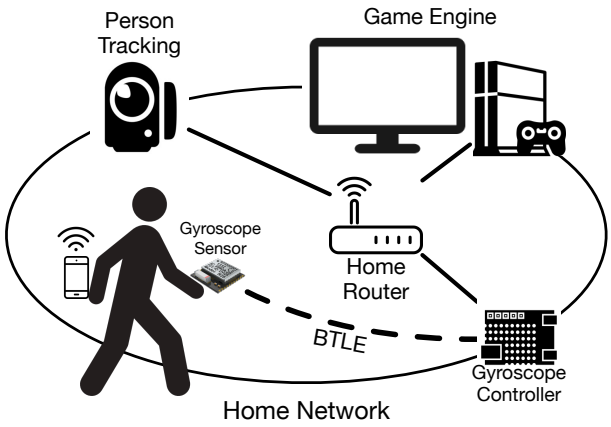

Fig. 2: Typical deployment of the Flow home entertainment experience.

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 2 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. 2) 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.[2]

The design of Flow application focuses on cloudless communication (while local reachability is a minor concern) and security (signing and verification).

While analyzing the requirement of Flow, we re-examined the building blocks described in Section IV of cite:iotdi-2016 paper, and found them helpful for the implementation of Flow application, as well as traditional home automation systems and many other IoT subdomains. Thus we designed NDN-IoT framework to provide the following abstractions naming: easily support multiple levels of names bootstrap: AS component and device onboarding discovery: can support various types of devices that Flow application requires

III. Design

Section IV of cite:iotdi-2017 paper already gave a high-level description of namespace design, trust management and local rendezvous in Flow application. This section does a review of each piece, and then describes the corresponding mechanisms in NDN-IoT framework.

A. 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;[3]
- *Manufacturer namespace*: a global namespace created by the IoT device vendors and for trust bootstrapping.

This separation of namespaces is reflected in section VI.A of cite:iotdi-2016 paper: naming application data separately from the device produces it, since typically when a consumer in the application requests a piece of data, it does not care about where the data is produced, but only the data itself, as long as it's verifiable using an application defined schema. In this work we added an additional *manufacturer namespace* that is independent from the local root namespace, since we envision that manufacturers will have globally unique names for their products used during bootstrapping, over-the-air updates, and similar processes.

Fig. 3 shows an example of the Flow namespace. In addition to these three namespaces which names 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. 3) 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 to list the application data prefixes they are publishing under. The device namespace of an AS also contains the trust schema of currently active applications. Schema and trust relationship details are described later in this section.

NDN-IoT framework interface supports the separation of application namespace and device namespace, and application developers can supply arbitrary names to both, or get a

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

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

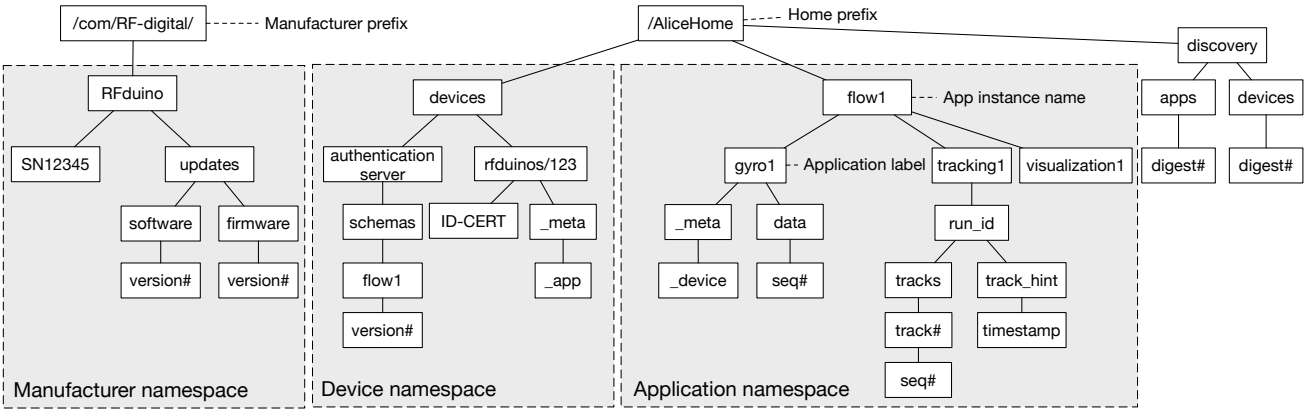

Fig. 3: Example namespace within the home environment where Flow is deployed.

configured default device names from the system. An example of the interface is given in Section IV.

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.[4] In 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.

B. 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 AS before they can start interacting with other local entities.[5]

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 his/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.

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

[5]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.

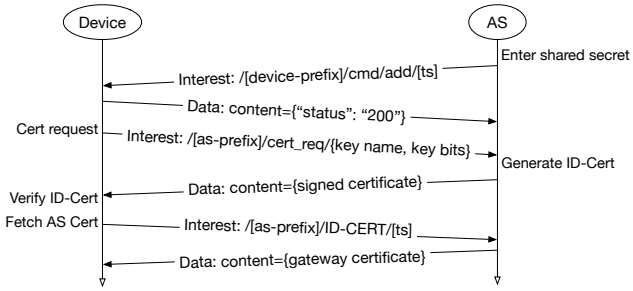

Fig. 4: Bootstrap trust relationship for new device.

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. 4. 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.

Applications like 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 devices 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. 3 "`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. The technical details of how to specify a trust schema are described in [4].

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

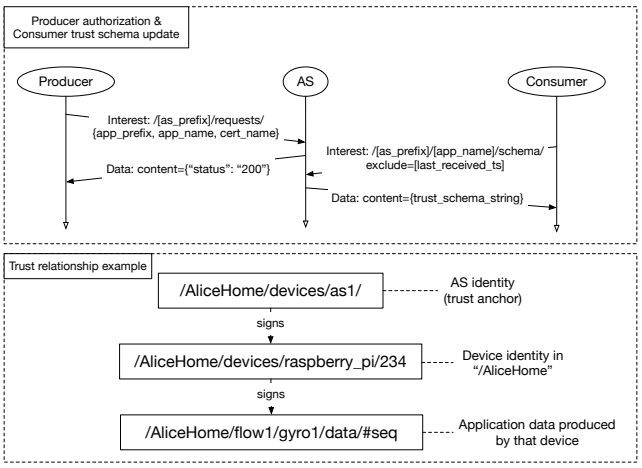

Fig. 5: Schematized trust between producers and consumers.

identity with the application prefixes it's authorized to publish under.[6] 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. 5, in which the AS signs a device identity, and the device signs a piece of application data it publishes.

NDN-IoT framework provides an example implementation of AS, client scripts for device onboarding, and an interface for producers to request publishing authorization and consumers to keep track of the application instance's trust schema. An example trust schema built by the AS, and application API calls are given in Section IV.

C. 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 nodes in the network that are interested in learning about them. The synchronization process utilizes a ChronoSync [5] recovery-like protocol to synchronize prefixes of active devices under the home entertainment "`/AliceHome/discovery/devices`" namespace.

Name discovery is performed independently on each device by lookups in the local copy of the rendezvous dataset. Once an application obtains the name prefix of the target device or application, the devices can follow the namespace structure described in III-A to construct Interests for fetching the certificates and metadata, which will bootstrap high-level service communication. An example discovery process is

[6]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.

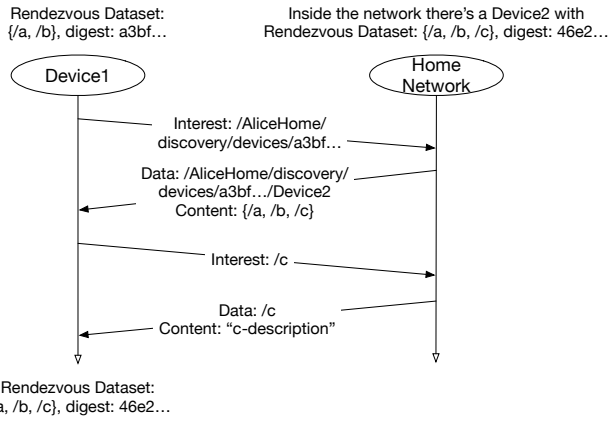

Fig. 6: Example of Flow discovery process.

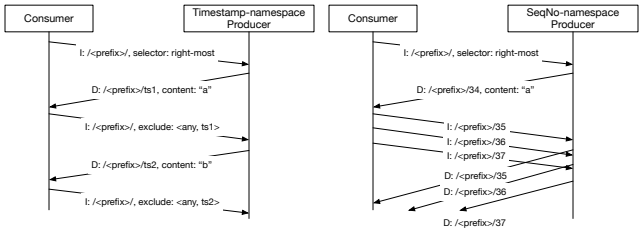

Fig. 7: Application-level pub/sub in NDN-IoT framework.

shown in Fig. 7, in which *Device1* discovers name "`/c`" from the network and fetches data from it.

NDN-IoT provides an implementation of this distributed discovery mechanism, whose details are further described in Section IV.

D. Application-level pub/sub

Pub/sub is identified recurring pattern in applications cite:iotdi-2016. To facilitate application development NDN-IoT framework implements application-level pub/sub in two commonly used namespaces:

Sequence-number namespace, in which producer publishes time series data whose last name component is a consecutive sequence number. The framework pipelines interest with the sequence numbers of the next few data packets in this namespace.

Timestamp namespace, in which data's last name is a timestamp. The framework uses outstanding interest and range exclusion to fetch the next piece of content. [7]

The workflow of both is shown in Figure **??**.

E. Supporting constrained devices

Flow application involves a producer running on constrained devices not powerful enough to perform asymmetric signing operations quickly, and may not have a forwarder implementation dedicated to their platforms. For those devices

[7]If exclusion filter is deprecated, the framework could switch to using a manifest of data names for consuming in a timestamp namespace

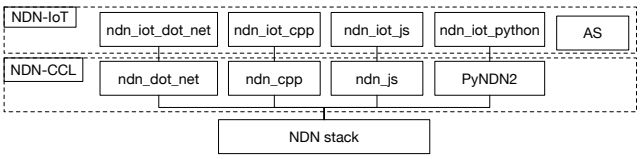

Fig. 8: The structure of NDN-IoT framework.

we introduce a helper that bridges them to the NDN home network.

In the framework, the pairing between constrained device and its helper is established using a shared secret key, manually put into both devices (not unlike the device onboarding step between a device and the AS). The helper then generates a public/private key pair on behalf of the constrained device to be associated with the latter's device identity. The constrained device may run a simple publisher which pushes data (signed by the shared secret) to the helper, and the helper repackages the data and signs the data using constrained device's private key, and then publishes the data on the home network.

An example of this pattern is described in Section IV.

IV. Implementation

We implemented the NDN-IoT framework and a prototype of Flow application separately. This section describes details in both the framework and application components.

A. NDN-IoT framework

The NDN-IoT framework is a set of libraries in Python, C++, JS and C# built on top of the NDN Common Client Libraries. Each library in the framework implements the naming, trust management and rendezvous in Section III from three functional blocks: bootstrap, discovery and application-level pub/sub. The overall structure of the framework is shown in Figure 8.

The rest of this subsection covers each functional block, and a description of the AS implementation available in Python and JavaScript.

The AS functionality is implemented based on the team's previous work on NDN-pi (citation: ndn-pi TR). The framework provides a server implementation in Python, which we run on Raspbian platform in Flow application. The server implementation also updates the codebase to work with PyNDN2 2.0b4, with major updates to security module interface, including using CCL library's built-in public/private key storages. [8] Authentication clients are available in both Python and JavaScript in order to support application components running on Ubuntu, OSX, Raspbian, and browser (Chrome and Firefox) platforms. Compared with the earlier work in NDN-pi, this implementation added a port in JavaScript to support browser applications. The JavaScript port uses the library's IndexedDbIdentityStorage and IndexedDbPrivateKeyStorage for key storage (this means the key storage in browser is origin-based, thus in our installation the webpage that adds this "device" and the webpage that publishes application content should be hosted under the same origin). Recall that in NDN-pi the client device generates a device ID[9]. In browser application's case we generate a random string in IndexedDB if one doesn't already exist, to use as the device identification of the device that currently runs the browser.

Bootstrap follows the suggestion in section VI.B and VI.C of the IoTDI '16 paper, and helps with device and application identity setup. Its abstractions include:

1) KeyChain setup: given a device identity (and optionally an AS name), construct a KeyChain and set up the default device certificate name for this application instance. This KeyChain is later used for signing and verification of all application data.
2) Consumer setup: given an application prefix, the Bootstrap module keeps outstanding interest for the application's trust schema (add example), and updates the local copy whenever a later version is received and verified.[10]
3) Producer setup: given an application prefix, the Bootstrap module requests authorization from the AS to publish under that prefix by sending a command interest including this device's identity and the prefix it wants to publish for (add example). If the AS authorizes the request, it adds an entry stating to the application trust schema to reflect the updated trust relationship, and publishes a new version for all consumers to fetch.

In practice, the application code to set up a gyroscope data producer on a Raspberry Pi may look like the following.

```
from ndn_iot_python import Bootstrap
deviceName = Name("/AliceHome/devices/rpi2")
dataPrefix = Name("/AliceHome/flow1/gyros")
appName = "flow1"
face = Face()
bootstrap = Bootstrap(face)
bootstrap.setupDefaultIdentityAndRoot(deviceName,
  ↪  onSetupComplete, onSetupFailed)

def onSetupComplete(certificateName, keyChain)
  bootstrap.requestProducerAuthorization(dataPrefix,
  ↪  appName, onRequestSuccess, onRequestFailed)

def onSetupFailed(message)
  print(message)
```

And in a Flow application instance "flow1", a trust schema built by the AS may contain the following sections.

```
trust-anchor
{
  type "base64"
  base64-string "Bv0DD..."
}
```

[8]We now use BasicIdentityStorage and FilePrivateKeyStorage for key storage, and ConfigPolicyManager for data verification, whereas the old implementation, developed at a time when PyNDN security library module was evolving, used custom derived classes (for example, IotPrivateKeyStorage).

[9]The server uses this ID to construct the initial "add device" interest name. This ID was implemented as the CPU serial of a Raspberry Pi.

[10]The application trust schema may evolve over time, when new device names are added and authorized to publish under certain application prefixes

Trust anchor certificate is installed during the bootstrap process. This rule is automatically populated by the AS.

```
rule
{
  id "Certs"
  for "data"
  filter
  {
    type "regex"
    regex "^[^<KEY>]*<KEY><>*<ID-CERT>"
  }
  checker
  {
    type "customized"
    sig-type "rsa-sha256"
    key-locator
    {
      type "name"
      name "/AliceHome/devices/gateway/KEY/ksk
-1485314801/ID-CERT"
      relation "equal"
    }
  }
}
```

This rule mandates that all certificates should be signed by the gateway.[11] This rule is automatically populated by the AS.

```
rule
{
  id "discovery-data"
  for "data"
  filter
  {
    type "regex"
    regex "^[^<discovery>]*<discovery><>*"
  }
  checker
  {
    type "customized"
    sig-type "rsa-sha256"
    key-locator
    {
      type "name"
      regex "^[^<KEY>]*<KEY><>*<ID-CERT>"
    }
  }
}
```

This rule mandates that all discovery data should be signed. This rule is automatically populated by the AS.

```
rule
{
  id "/AliceHome/flow1/gyros"
  for "data"
  filter
  {
    type "name"
    name "/AliceHome/flow1/gyros"
    relation "is-prefix-of"
  }
  checker
  {
    type "customized"
    sig-type "rsa-sha256"
    key-locator
    {
      type "name"
      name "/AliceHome/devices/rpi2/KEY/dsk-
1485374576/ID-CERT"
      relation "equal"
    }
  }
}
```

This rule mandates that data under application prefix "`/AliceHome/flow1/gyros`" should be signed by device "`/AliceHome/devices/rpi2`". This rule is added to the trust schema after the AS approves the request from "`/AliceHome/devices/rpi2`".

Discovery uses a simple sync-based discovery similar to ChronoSync recovery mechanism.

In practice, after setting up the Bootstrap object, the application may use the following code to include a discovery module.

```
from ndn_iot_python import Discovery, Observer

class MyObserver(Observer):
  def onDiscovered(name, description):
    print(name.toUri() + description)

keyChain = bootstrap.getKeyChain()
certName = bootstrap.getDefaultCertificateName()
syncPrefix = Name("/AliceHome/discovery/devices")
discovery = Discovery(face, keyChain, certName,
 ↪  syncPrefix, MyObserver())

description = "My Raspberry Pi device"
deviceName = Name("/home/devices/rpi2")
discovery.publishObject(deviceName, description)
```

Application-level pub/sub follows the suggestion in section VI.F of the IoTDI '16 paper, and provides the following abstractions:

1) Consumer for timestamp namespace (/prefix/[timestamp]): this consumer uses oustanding interest with range exclusion to ask for latest piece of data, and upon data retrieval and successful verification, updates the range exclusion with the received timestamp.
2) Consumer for sequence-number namespace (/prefix/[sequence-number]): this consumer pipelines interest for the next few sequence numbers, and upon data retrieval and successful verification, issues an interest for the sequence number after the last one in the pipeline.

In practice, after setting up the Bootstrap object, the application may use the following code to include a sequence number consumer module.

```
from ndn_iot_python import AppConsumerSequenceNumber
keyChain = bootstrap.getKeyChain()
pipelineSize = 5
consumer = AppConsumerSequenceNumber(face, keyChain,
 ↪  pipelineSize)
prefix = Name("/home/flow1/gyros")
consumer.consume(prefix, onData, onTimeout,
 ↪  onVerifyFailed)
```

[11] In this iteration of Flow, device certicates are the only type of certificate involved in the trust relationship in Section III-B.

B. Flow application components

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

1) *Indoor positioning*: We use OpenPTrack,[12] a multi-camera person tracking system. The NDN producer for OpenPTrack[13] (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[14], 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. 9.
3) *Mobile phone interface*: We employ an Android phone that loads a control webpage (written in JavaScript) 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[15] 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 implementation for both NDN-IoT framework and Flow application are available online.[16] We installed two instances of the Flow application testbed at UCLA and Huawei. Fig. 10 shows a diagram of the system and its message flows after all devices are bootstrapped with an AS, which in our installation is another Raspberry Pi.

[12]http://openptrack.org/about/
[13]https://github.com/OpenPTrack/ndn-opt/
[14]https://github.com/named-data/ndn-cpp/
[15]https://unity3d.com
[16]https://github.com/remap/ndn-flow

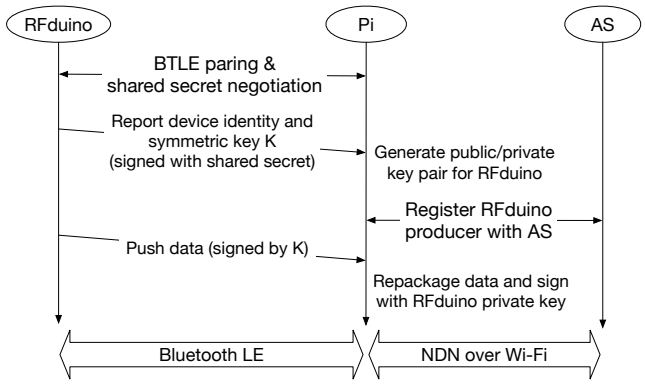

Fig. 9: RFduino data publishing with assistance of Raspberry Pi controller

V. Application scenario

Flow application and NDN-IoT framework are designed as an example to illustrate what the whole picture of an NDN-IoT application looks like (to users, service providers). During the design phase, we envisioned the following scenarios

References

[1] 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 ACM International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2009, pp. 1–12.
[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 44, no. 3, pp. 66–73, Jul. 2014.
[3] 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 (Invited Paper)," in *Proceedings of the 1st IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI)*, Apr. 2016, pp. 117–128.
[4] Y. Yu, A. Afanasyev, D. Clark, k. claffy, V. Jacobson, and L. Zhang, "Schematizing Trust in Named Data Networking," in *Proceedings of the 2nd ACM International Conference on Information-Centric Networking (ICN)*, 2015, pp. 177–186.
[5] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized Dataset State Synchronization in Named Data Networking," in *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–10.

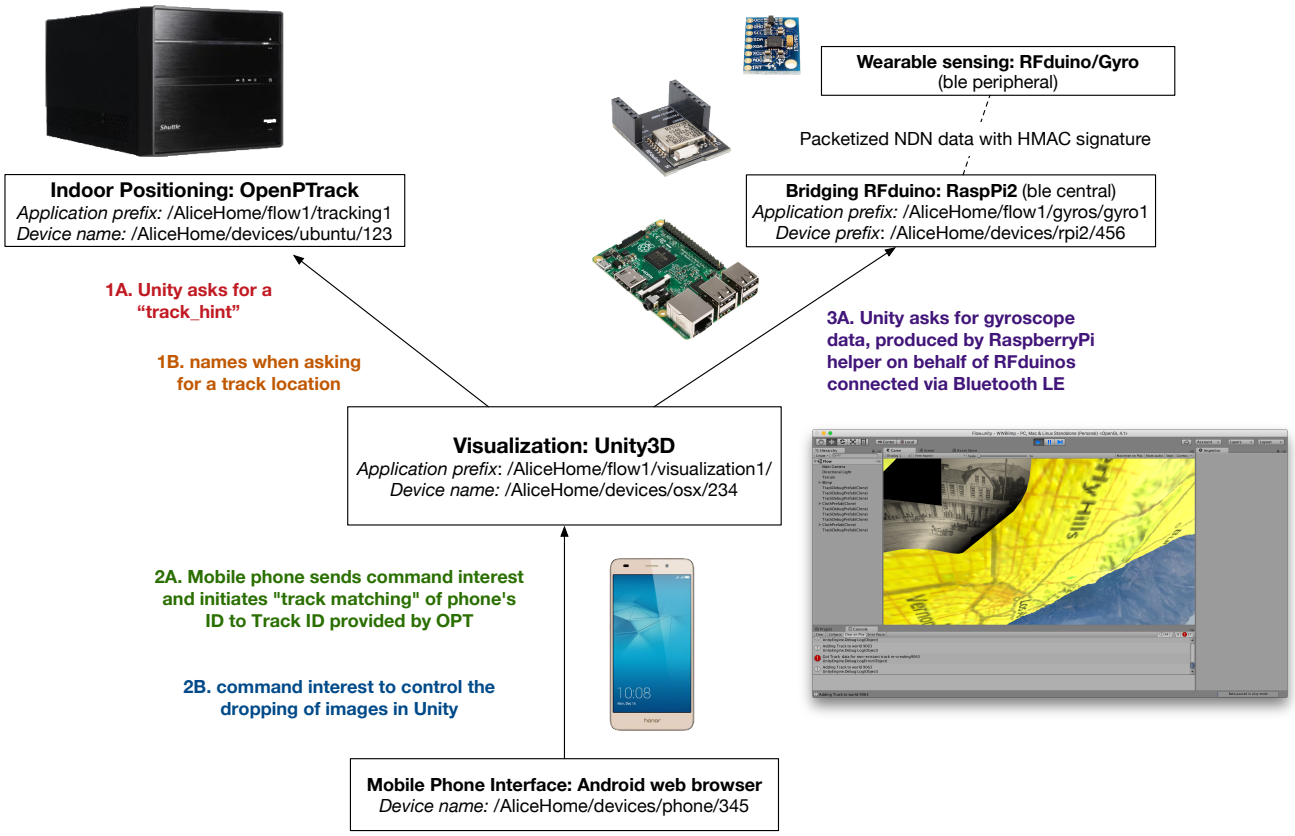

Fig. 10: Application components and message flows in Flow