Named Data Networking of Things and Example Application "Flow"

***Abstract*—In this report we introduce the application Flow, a prototype home entertainment experience built using Named Data Networking (NDN). Flow provides an exploratory experience in which a player navigates and interacts with a virtual environment via person tracking (OpenPTrack), wearable devices (gyroscope), and their mobile phone. Through this application we showed an approach to achieve two fundamental functions in IoT, trust management and rendezvous, in a way that does not rely on (but can incorporate) cloud-based services. 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. To develop Flow, we designed and implemented the Named Data Networking of Things (NDN-IoT) framework, libraries in multiple languages that implement naming, trust and bootstrap, discovery, and application level pub/sub, to explore IoT-based application development in a home environment. While a high-level design of the framework and application can be found in the author's IOTDI 2017 invited paper, this report focuses on low-level details, including library workflow and interfaces, with in-application examples.**

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. Especially with emerging virtual and augmented reality technologies, IoT offers the promise of new immersive and interactive experience for end-users.

Many IoT frameworks and ecosystems have been proposed to facilitate the development of more sophisticated applications. 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.

Flow application, a prototype home entertainment experience, takes an NDN approach [1], [2], and builds on our previous work in [3] to achieve foundational functions of *trust management* and *rendezvous*. While in the authors' IoTDI 2017 paper Flow application is used to illustrate the high-level concepts, this report focuses on an expanded description of lower-level details in the implementation of Flow and its underlying libraries.[1]

[1]Background study, key design insights, and conceptual evaluations are better covered in the IoTDI paper.

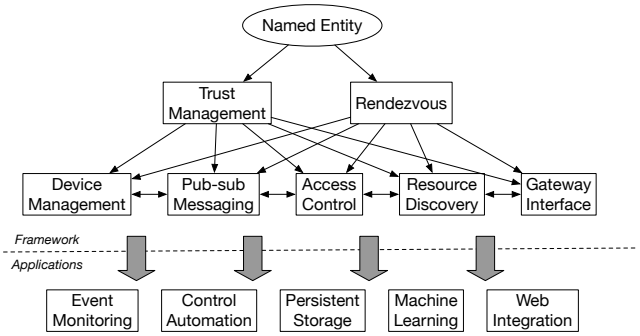

Fig. 1: Hierarchical architecture of IoT services.

The rest of the report is structured as follows. In Section II, we first give an overview of Flow application. In Sections III and IV, we present the design and implementation details of Flow, and finally conclude with SectionV.

II. Flow: overview

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.

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.

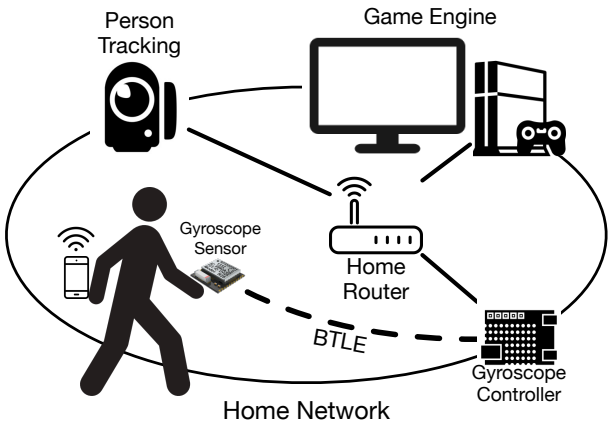

Fig. 2: Typical deployment of the Flow home entertainment experience.

While analyzing the components of Flow, we re-examined the building blocks described in Section IV of [3], and found their patterns common in the implementation of Flow application. Thus we separate the common functionality, including naming, bootstrap, discovery, and application level pub/sub, from the application and implemented them in a framework called Named Data Networking of Things Framework (NDN-IoT), which we believe may facilitate the development of traditional home automation systems and many other IoT applications.

III. Design

Section IV of the authors' IoTDI 2017 paper already gave a high-level description of namespace design, trust management and local rendezvous in Flow. 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;[2]
- *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 that produces it, since typically when a consumer in the application requests a piece of data, it cares about the received data being authentic and useful, not necessarily from where or which device this data is sent out.

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 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.

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. 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.[3] 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 names are used during onboarding, in the name of the first command interest that the AS sends out to add the device to the home network. This process is described in more detail in the coming subsection.

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 configured default device names obtained from device onboarding process. An example of the interface is later described in Section IV-A.

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.

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

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

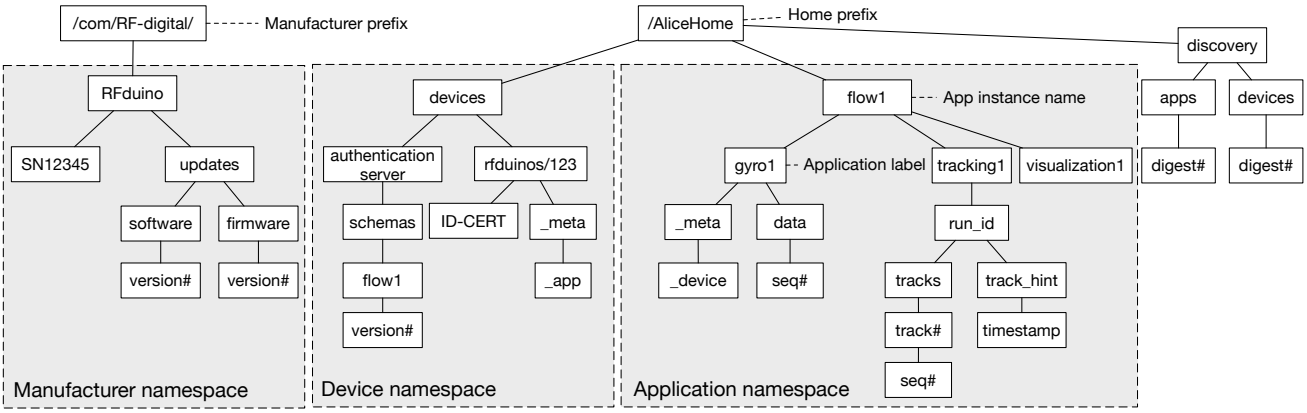

Fig. 3: Example namespace within the home environment where Flow is deployed.

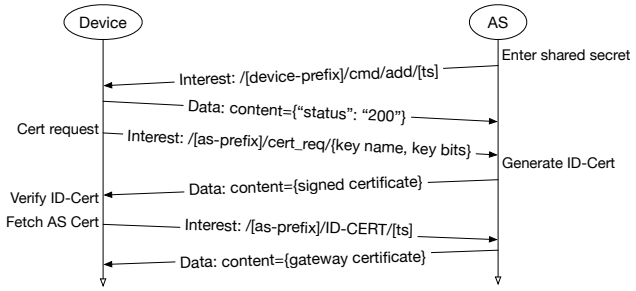

Fig. 4: Bootstrap trust relationship for new device.

The AS 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.

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. 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.

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 identity with the application prefixes it's authorized to publish under. [4] 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 described in Section IV-A.

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

[4]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.

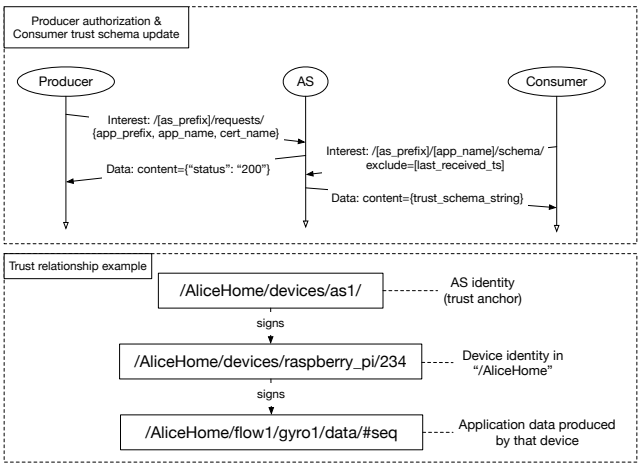

Fig. 5: Schematized trust between producers and consumers.

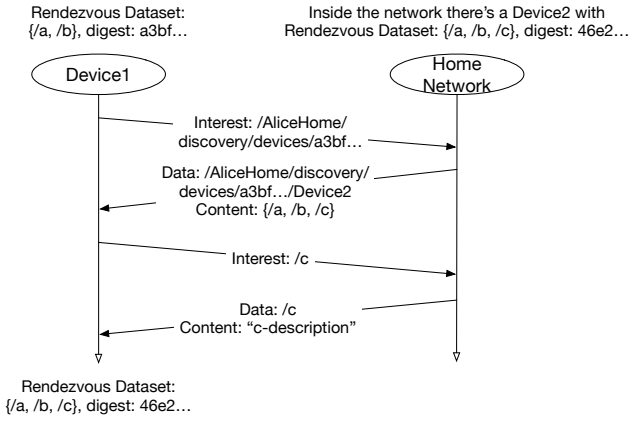

Fig. 6: Example of Flow discovery process.

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 "`/AliceHome/discovery/devices`" namespace.

Name discovery is performed independently on each device by lookups in the local copy of the rendezvous dataset. The device sends a multicast Interest with the digest of its local rendezvous dataset attached to the name. Receivers of this interest, upon seeing a different digest, will reply in a Data packet with its own set of names. The device can then obtain the name prefix from the set difference, and 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 shown in Fig. 6, in which *Device1* discovers name "`/c`" from the network and then fetches data from it.

NDN-IoT provides an implementation of this distributed discovery mechanism, whose interface is further described in Section IV-A.

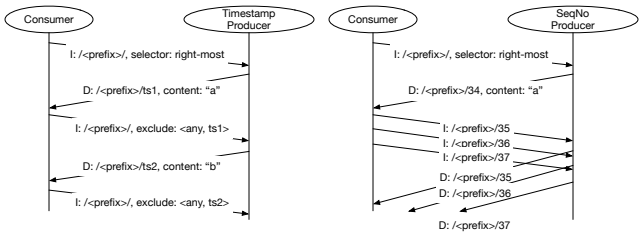

Fig. 7: Application-level pub/sub in NDN-IoT framework.

D. Application-level pub/sub

Pub/sub is identified as a common pattern in IoT applications in [3]. To facilitate application development NDN-IoT framework implements application-level pub/sub in two commonly used namespaces:

Timestamp namespace, in which producer publishes time series data whose last name component is a timestamp. The framework uses outstanding interest and range exclusion between *<Any, previously received timestamp>* to fetch the next piece of content. [5]

Sequence-number namespace, in which data's last name is a sequence number. The framework pipelines interest with the sequence numbers of the next few data packets in this namespace.

The workflow of both is shown in Figure 7.

E. Supporting constrained devices

Flow involves producers 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 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-A.

IV. Implementation

This section describes details in both the NDN-IoT framework and Flow 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 naming, trust management and rendezvous in Section III, which are

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

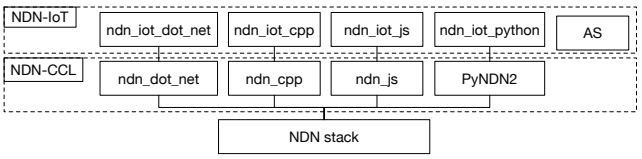

Fig. 8: The structure of NDN-IoT framework.

organized into 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 an AS implementation available in Python and JavaScript.

Bootstrap follows the suggestion in section VI.B and VI.C of [3], 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.[6]
3) Producer setup: given an application prefix, the Bootstrap module requests authorization from the AS to publish under that prefix following the description in Section III-B. If the AS authorizes the request, it adds an entry to the application instance's trust schema to reflect the updated trust relationship, and publishes a new version for all consumers of that instance to fetch.

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

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

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

Trust anchor (AS) certificate is installed to the added device during the onboarding 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.[7] 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. And the signer's certificate should be verified according to the previous rule. This rule is automatically populated by the AS, if one component in this instance indicates that it will call Discovery.

```
rule
{
  id "/AliceHome/flow1/gyros"
  for "data"
```

[6]The application trust schema evolves over time, since new device names may be added and authorized to publish under certain application prefixes

[7]In this iteration of Flow, a single level of device certicates are the only certificates involved in the trust relationship in Section III-B, thus this simplified rule would suffice.

```
  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 publishing request from "`/AliceHome/devices/rpi2`".

Discovery is implemented following the description in Section III-C. This mechanism differs from the suggested in Section VI.B of [3], in which the device queries the AS for currently active devices. We use a sync-based mechanism so that the AS does not have to actively maintain a list of devices online.

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("/AliceHome/devices/rpi2")
discovery.publishObject(deviceName, description)
```

Application-level pub/sub follows the suggestion in Section VI.F of [3], 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("/AliceHome/flow1/gyros")
consumer.consume(prefix, onData, onTimeout,
 ↪  onVerifyFailed)
```

The AS 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 a Raspberry Pi serving as AS in Flow application. The AS server implementation updated 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.

AS clients are available in both Python and JavaScript in order to support application components running on various (Ubuntu, OSX, Raspbian, and browser) platforms. The JavaScript client uses library's IndexedDb for key storage, and the key storage in browser is origin-based. [8] Recall that we use a manufacturer given identity to onboard a device, which in browser application's case is not available. Instead we generate and store a random string if one doesn't already exist, to use as the manufacturer name of the device currently running the browser.

B. Flow application components

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

1) *Indoor positioning*: We use OpenPTrack,[9] a multi-camera person tracking system. The NDN producer for OpenPTrack[10] (written in C++) publishes the position of each person at a 30Hz rate, along with lower rate metadata about active tracks. A consumer of track data first fetches the active track IDs contained in the metadata, named in a timestamp namespace, and then uses the IDs to construct interest names to fetch the actual position, named in a sequence number namespace. The interest and data exchange in this process is shown in Figure 10, between the visualization component and the indoor positioning components (key 1A and 1B).
2) *Wearable sensing*: We use an RFduino 22301 with gyroscope MPU6050 attached to provide virtual camera control. The RFduino follows the pattern for constrained devices described in Section III-E, and we introduced a Raspberry Pi as its helper. The RFduino runs a minimum NDN producer, implemented with the ndn-cpp-lite library[11], which generates data at roughly 2Hz

[8] Which means in our installation the webpage that adds this "device" and the webpage that publishes application content should be hosted under the same origin.

[9] http://openptrack.org/about/

[10] https://github.com/OpenPTrack/ndn-opt/

[11] https://github.com/named-data/ndn-cpp/

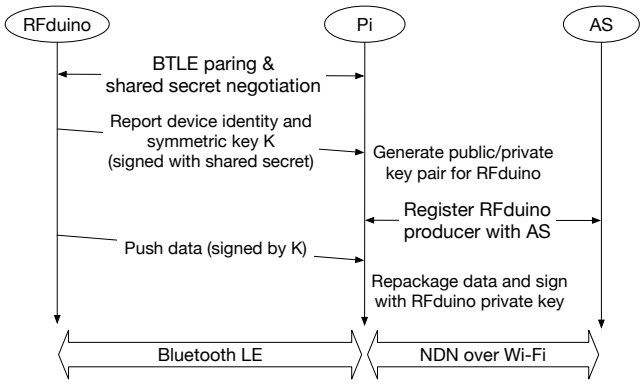

Fig. 9: RFduino data publishing with assistance of Raspberry Pi controller

rate. This process of pairing the RFduino and its helper is shown in Fig. 9.

Message exchanged between the Raspberry Pi helper and the rest of the system is shown in Figure 10.

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). Keys 2A and 2B in Figure 10 reflects these two types of command interests.
4) *Visualization*: We use the Unity3D[12] 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. A screenshot of the visualization component is also shown in Figure 10, in which a user has tilted the orientation of the virtual camera, an image is being dropped on his location in the virtual environment, and another user, represented by the small green dot in the distant, is being tracked and rendered on the virtual environment.

The implementation for both NDN-IoT framework and Flow application are available online.[13] With documentations dedicated to the framework[14], its interface[15], installation[16], and the application[17]. We installed two instances of the Flow application testbed at UCLA and Huawei. Figure 10 shows a summary of our installation at Huawei.[18]

For better visualization of data names in the system, we also implemented a namespace tree visualizer, which recursively expressed interest with previously received name components excluded, to probe data packets in the network.[19]

V. Conclusion and Future Work

This report describes the design and implementation of Flow, an IoT home entertainment experience; and NDN-IoT framework, a set of libraries built to facilitate the development of IoT applications.

Their design showed a cloudless approach to two fundamental functions in IoT, trust management and rendezvous, both achieved by exchanging application-defined hierarchically named and secured data packets at the networking level.

We implemented the framework and application, and demonstrated a prototype in Huawei in early 2017.

Further tasks remain for both the framework and the application. Data access control should be introduced in the framework. The NDN team has proposed an encryption based access control mechanism in [6], which is a good starting point for experimenting inside the framework. Discovery mechanism in the framework stands to be improved. For simplicity in implementation, current mechanism constantly does ChronoSync recovery, whose efficiency can be improved by, for example, using ChronoSync directly as one-level of indirection for name discovery. Library interface may require more thoughts. With more applications using the framework, we can better identify what abstraction are easier for application developers, or what other general application building blocks can be added to the framework. Finally, more efforts are required in implementation and documentation of the framework and application. For example, although the design conceptually supports more levels in the trust relationship, this feature is not yet tested by any application components.

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.

[12] https://unity3d.com

[13] https://github.com/remap/ndn-flow

[14] https://github.com/remap/ndn-flow/tree/master/framework

[15] https://github.com/remap/ndn-flow/tree/master/design/docs

[16] https://github.com/remap/ndn-flow/blob/master/design/DRAFT_FLOW_TechGuide.docx

[17] https://github.com/remap/ndn-flow/tree/master/application

[18] A screen recording of the running system can be found at https://www.youtube.com/watch?v=MNI_qd75SPQ

[19] The code is available at https://github.com/zhehaowang/namespace-tree. Since the installation of Flow it has been applied to other projects such as NDNFit and mini-BMS, and has gone through major UI revisions.

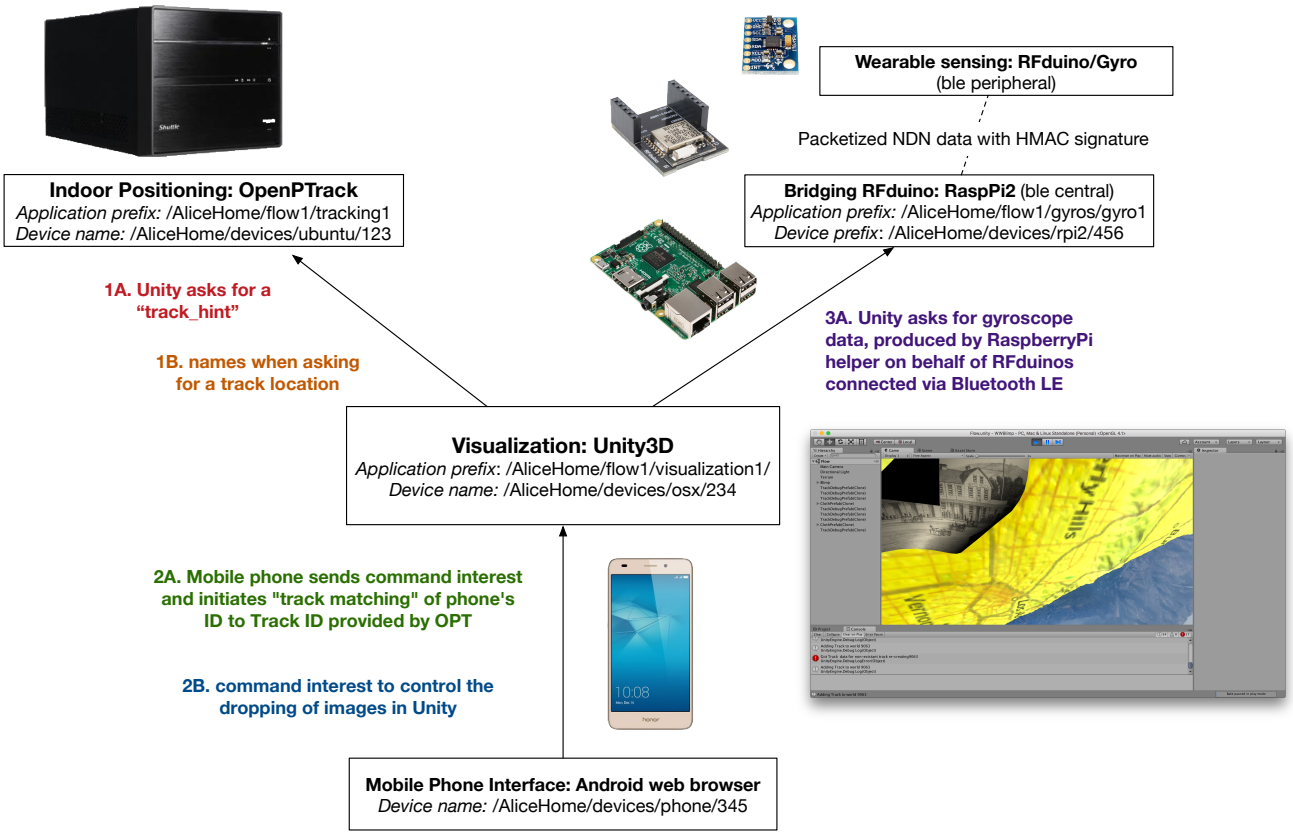

Diagram Key for NDN Interest-Data Exchange

1A.
Interest: /AliceHome/flow1/opt1/<run_id>/metadata/, exclude: <last_received_timestamp>
Data: Interest name + <timestamp>, content: {"id": 45, "seq#": 312}

1B.
Interest: /AliceHome/flow1/opt1/<run_id>/tracks/<track_id>/<seq#>
Data: Interest name, content: {"x": 1.0, "y": 0.9, "z": -0.3}

2A.
Command interest: /AliceHome/flow1/unity1/<action:match, id:alice_phone>
Data: Interest name, content: {"status": "200", "data": "<html>track 45, show links</html>"}

2B.
Command Interest: /AliceHome/flow1/unity1/<action:link_click, id:alice_phone, link:img_3>
Data: interest name, content: {"status": "200"}

3A.
Interest: /AliceHome/flow1/gyros/gyro1/, exclude: <last_received_timestamp>
Data: Interest name, content: {"p": 0.3, "y": 0.5, "r": 0.1}

Fig. 10: Application components and message flows in Flow

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