Understanding the Namespace of NDN-Lite

immediate

ABSTRACT

1. INTRODUCTION

The NDN-Lite library implements the Named Data Networking Stack with the high-level application support functionalities and low-level OS/hardware adaptations for Internet of Things (IoT) scenarios. NDN-Lite's mainly use cases are the emerging smart homes networks. We have developed numerous applications based on this framework and will explain how application needs influence the namespace design.

2. SEPARATE FUNCTIONALITIES

We first consider what functionalities a NDN-based home IoT system should provide.

To serve as a basic NDN entity, one should have four pieces: Name, Certificate, Trust Anchor, and Trust Schema. We claimed that since they're all first delivered or installed at entities' security bootstrapping time, these trust management should remain in the same subnamespace. Then service discovery and access control are needed for the system to faciliate application interaction and content protection, corresponding subnamespace are needed. Considering the independency of the above three functionalities, they should all be kept into non-overlapping subnamespace to avoid conflicts.

Besides basic bricks for a NDN IoT system, common application or devices features should also be considered. In current smart home systems (cite) devices are often categorized into different capabilities. Applications are represented as consumers for this device provided capabilities. This style of categorization benefit platform-level Pub/Sub much and become an emerging trend of current IoT systems. NDN-Lite also follows a similar approach to bring entity's capabilities directly under the root prefix (e.g., */alice-home*) to better serve our Pub/Sub 4.

To summarize, system-level functionalities are separated into different subnamespaces to ease both trust management and data aggregation. Therefore, NDN-Lite separates overall namespace into following subnamespaces:

- */<home-prefix>/cert*
- */<home-prefix>/sd*
- */<home-prefix>/ac>*
- */<home-prefix>/<capability>*

3. DEFINE APPLICATION DATA UNIT

For NDN applications, it is important to give the definition of Application Data Unit (ADU). A clear definition of ADU can help the data transfer be more efficient. As other work pointed out, ADU definition should consider **application semantics**, **data producers** and **data granularity**.

Unclear ADU definition where a single data unit contains more than one data producers should be avoided. Previous work on namespace design (cite) has shown the how data ownership problem of IMAP data structure affect ADU definition in application translation. However, as a NDN native application, NDN-Lite does not suffer from it and no special design is needed here. Data granularity relates to the granularity of data discovery and retransmission thus should be taken carefully to avoid inefficiency. In the smart home scenario, most data can be abstracted as events and usually not large. Then decision of event-based granularity comes naturally.

Lastly we consider application semantics of ADU. The name of ADU should be informational enough to express application semantic, and concise enough to only include application semantics. Descriptors (e.g., stale or fresh) within the application and not helpful to forwarding should not appear in the ADU's name. Based this principle, we surveyed mainstream smart home applications (cite) and found essential elements required for constructing a smart home namespace.

- **Capability**: Smart home data are often related with certain platform-level capabilities (e.g., vibration, water leakage detection, pushing SMS notifications). The assication should be reflected in names. This idea is aligned with the design decision in Section 2.

- **Data type**: Whether this Data packet refers to a command or content (other prevailing types may exist but out of discussion). The different reliabilty solutions and trust polices can be applied based on data types.

- **Effective scope**: Command type packets issued by applications should contain an scope, and only inside this limited scope commands are effective. Similarly, content type packets better adopt this design to avoid confusion. For example, temperature sensed by different sensors in the same room probably vary. Thus names of their produced data should tell the difference of their sensing scope.

- **Command identifier**: Commands can further be categorized based on their semantics, and identifiers are therefore needed. Conceptually, commands for *turning the fan off* and *turning the fan on* should be named differently in the case only adult can issue the latter command. Such clearness can faciliate access control and trust policy management.

All four elements above should be included into ADU's name, in order to be expressive enough for forwarding and applications performing trust policy examination. Other less important elements (if any) should be kept out of ADU's name.

4. SUPPORT AGGREGATION

Once defined what elements should be included in ADU's name, a pratical issue emerges: in what order they are placed in name? It is actually a highly applcation relevant question posed, given the way of ordering determines way of data aggregation happens.

The corelations of the four elements mentioned also complicate the decision. Notice that *data type* can split *capability* namespace into command and content, while *capability* can also classify commands into different categorizes. Same problem also exist for *effective scope* and *command identifier*.

We start order determining by making observation that in most cases *effective scope* and *command identifier* are finer granularity of *data type* and *capability*. Hence the latter two should be at front of the former two. Then we consider aggregation apporach can benefit application most.

Another important observation made is *capability* and *effective scope* are more expressive and providing more data collection dimensions than the other two. More expressive components should be at smaller depth inside the nametree, to avoid communication overhead of Pub/Sub. An example is given in Fig 1 with three *capability* are invovled. One have three dimensiosns of freedom to access namespace from /<*home-prefix*> when

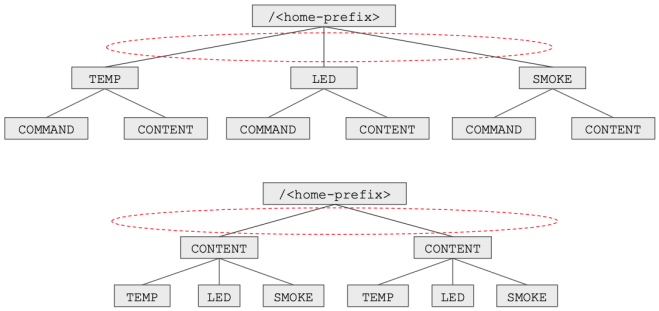

Figure 1: Aggregation Approaches Comparison

capability's position is upper than *data type*. In contrast, the other way of aggregation only provide two dimensions of freedom for namespace accessing at the entry point. In the first organization, received data of subscription on the first depth are either commands or content. However, data are less guessable when the same subscription happens on the second organization since *capability* has numerous types and more expressive. Same analysis applies on *effective scope* and *command identifier* situation, and *effective scope* prevails in our application scenarios.

Finally we get the total order benefits aggregation most: /<*home-prefix*>/<*capability*>/<*data-type*>/<*effective-scope*>/<*data-id*>. This order also better fit the functionality separation feature in Section 2. We argue this order is the outcome of current NDN-Lite applcation needs of accessing namespace with maximized freedom, and this order may change if applcation needs shift in future.

5. PUTTING TOGETHER

After ADU definition is given and overall namespace design determined, it's natural to come up the namespace structure by putting previous ideas together. In this detailed namespace illustrated in Fig 2, *Effective scope* are represented with series of name components ranging from zero to two, and timestamp is appened to achieve the Data name uniqueness.

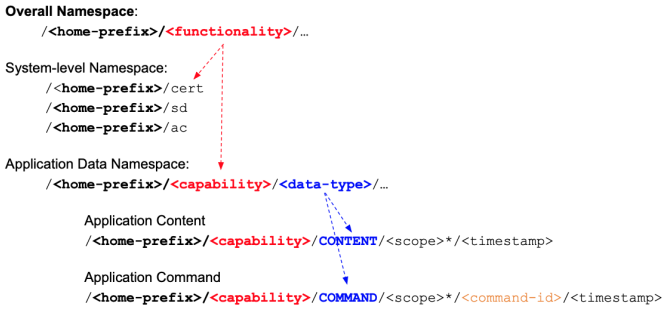

Figure 2: Overall NDN-Lite Namespace

The last problem not solved is to name identities.

To simplify certificate and trust management, identity names are only allowed binded with one capability. Multipurpose sensers or applications will host multiple certificates and sign packets with corresponding identities as a result. Another factor to consider the finest granularity in data publishing. Content type data, unlike commands often explicitly assigned to an effective scope in data production time, thus need to assign a default fine-grained scope for applications falling back. We assign this scope in identity names during the bootstrapping time.

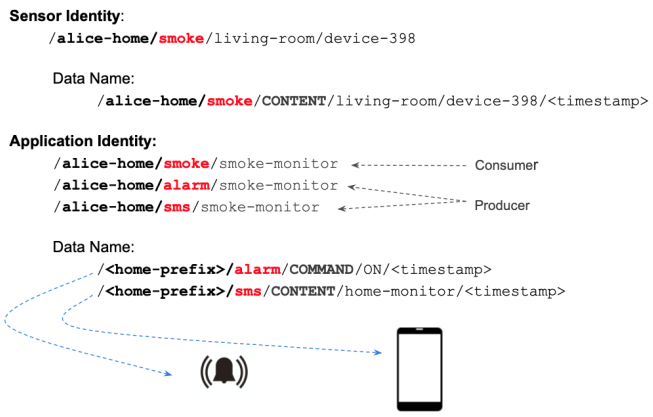

Figure 3: Example Application: Smoke Monitor

The complete workflow is demonstrated by a case study in Fig 3 with a smoke monitoring application. Smoke detector installed in the living room is named with */alice-home/smoke/living-room/device-398*. Its encryption key for publishing data under prefix */alice-home/smoke* is obtained from access controller residing in */alice-home/ac*. To avoid ambiguities in naming in detection data publishing, data names falls back to *effective-scope* with */living-room/device-398*. The smoke monitor application is installed in the home hub, and three different identities are bootstrapped by this application. Through subscribing prefix */alice-home/smoke*, smoke monitor receive smoke detection data and perform trust schema checking and data decryption. Its decryption key is obtained from access controller in the same way with smoke detector. Then two data are published under */alice-home/alarm* and */alice-home/sms*. Alarm command issued should be effective in the entire home since not specific *effective-scope* is given. Meanwhile, user's smartphone which subscribes */alice-home/sms* will receive the notification.

6. CONCLUSION

In this report , we introduce the namespace of NDN-Lite, the lightweight NDN stack implementation concentrating on smart home scenarios. Various significant factors are discussed and strong evidence are shown how application needs shape namespace structure. We will continue udpate this namespace documentation as NDN-Lite's evolution.

7. REFERENCES