Named Data Networking for Data Spaces

Y. Thomas, N. Fotiou, I. Pittaras and G. Xylomenos
Mobile Multimedia Laboratory, Department of Informatics,
Athens University of Economics and Business (AUEB), Greece
E-mail: {thomasi, fotiou, pittaras, xgeorge}@aueb.gr

***Abstract*—The Secure Named Data Sharing (SNDS) architecture is a content brokering service built on top of Named-Data Networking (NDN), leveraging its native support for multicast, multisource and caching. SNDS transparently supports IP-based content providers and consumers via a gateway that implements ETSI's NGSI-LD data spaces API, translating HTTP requests to and from the appropriate NDN messages. To fully support the query-based NGSI-LD API, we build appropriate protocols over NDN. We present a prototype implementation of the SNDS architecture and a preliminary evaluation of its features.**

***Index Terms*—NDN, NGSI-LD, Data Spaces.**

I. Introduction

Data spaces are an emerging concept that enables semantic interoperability of data, uniform data access methods and enhanced data sovereignty and trust. An integral part of a data space is the *data broker*, which facilitates data transfer from providers to consumers. We explore the performance benefits of implementing such a data broker over *Named Data Networking* (NDN), an information-centric architecture [1]. NDN supports ubiquitous caching, native multicast transport and multisource content provision, making it a good *Content Distribution Network* (CDN) underlay (e.g., [2]). Nevertheless, its Interest/Data API can only support individual data transfers. Our *Secure Named Data Sharing* (SNDS) architecture implements the *Next Generation Service Interfaces for Linked Data* (NGSI-LD) data spaces API [3] over an NDN-based data brokering service. This HTTP-based API allows end-users to make sophisticated queries, without needing another layer of processing on top.

To ease the transition to SNDS, the NDN-enabled core network providing the data brokering service, is combined with IP-enabled endpoints. The interfacing of the two architectures takes place at edge NDN nodes that receive and translate NGSI-LD API requests, encapsulated in HTTP messages, to NDN operations, and vice versa. This paper describes the SNDS architecture and explains how the NGSI-LD API is implemented over NDN, allowing IP-enabled content providers and consumers to interoperate with SNDS. We also present our prototype implementation and initial performance results.

II. SNDS Design

An overview of SNDS is presented in Fig. 1, showing a simple topology with two IP-based end-users, a content consumer and a content provider, exchanging content via an NDN-based network. The two networks are bridged at the NDN edge routers, which host the SNDS service. This service receives, translates, and forwards NGSI-LD API requests encapsulated in HTTP messages to the NDN network.

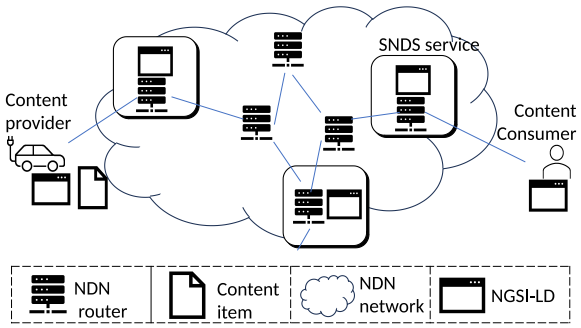

Fig. 1. The SNDS Architecture.

In NDN, everything revolves around content. Content consumers issue INTEREST messages to request content items by name, which content producers return using DATA messages. Content availability is advertised via ANNOUNCEMENT messages, which contain the prefixes of the names that a content provider serves. All NDN nodes maintain three data structures. The *Forwarding Information Base* (FIB) maps name prefixes to the *face(s)* (NICs or local ports) that should be used to forward INTEREST messages towards content sources. The FIB is populated by the ANNOUNCEMENT messages. The *Pending Interest Table* (PIT) records the face(s) from which non-satisfied INTEREST messages have arrived. Finally, the *Content Store* (CS) is a local cache for content items.

When an INTEREST message is received, the NDN node returns the content item in a DATA message if it available locally or in its CS. Otherwise, the PIT is checked to see if an entry for that name exists; if so, the PIT entry is updated with the new Interest, thus creating a multicast tree. If this also fails, the FIB is checked to see how the INTEREST must be forwarded, and a new PIT entry is created. DATA messages are routed back to the consumers hop-by-hop, consuming the pointers stored by the INTERESTS in the PITs.

The NGSI-LD API, proposed by the *European Telecommunications Standardization Institute* (ETSI), defines a query-based content resolution mechanism, offering a standardized way to build complex queries that work across cloud providers. In the NGSI-LD API, each content item is encoded in a structured file format, typically JSON, which consists of a unique ID field, a TYPE field and multiple (optional) ATTRIBUTE fields. The API offers two main lookup options: in ID matching, the lookup request explicitly specifies the

ID of a *unique* content item; in TYPE matching, the lookup request indicates the TYPE of a *set* of content items. These two lookup options can be augmented with two types of filters: the *selective* filter indicates which attributes of the content items should be returned, while the *conditional* filter indicates a set of conditions over the attributes that should be met by the returned content items; both types can be combined in a single query. For example, a TYPE matching query for content items representing vehicles passing through a traffic control point could ask for the SPEED (selective filter) of cars whose TYPE is `ford` and whose COLOR is `red` (conditional filter).

III. Mapping the NGSI-LD API to NDN

The NDN architecture can trivially support ID matching NGSI-LD API requests, as the (unique) ID of the content item can be used as an NDN name. On the other hand, it does not directly support TYPE matching NGSI-LD API requests which may return a set of items. Even though NDN offers multisource, that is, getting one content item from multiple sources, it does not support the gathering of *multiple* content items from various sources. Two types of approaches have been proposed for this problem, a passive and an active one.

The *passive* approach relies on sniffer applications that monitor ANNOUNCEMENT messages, creating a list of the available providers per content name. This approach requires modifications to the NDN core functions, as it needs access to low-level messages, and it requires tracking the *physical nodes* that provide the content. The *active* approach relies on catalog applications, which maintain the available sources per content type. In this approach, a content provider sends two ANNOUNCEMENT messages: the first uses the unique item's ID and the second the TYPE of the content item. The catalogs periodically emit INTEREST messages using the TYPE of the content as the name. NDN forwards them to the new provider, thus creating an end-to-end communication path. Then, the provider retracts its ANNOUNCEMENT, to allow more providers to be found. This periodic probing, however, adds computational and communication overheads.

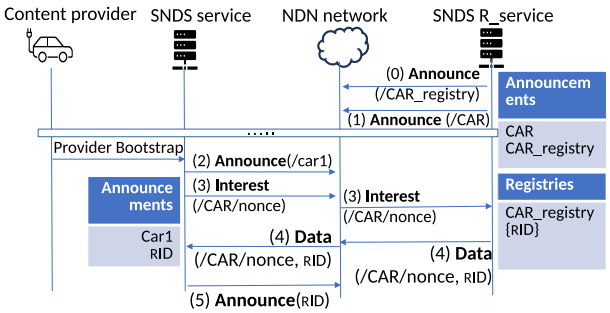

Fig. 2. Announcing content: a provider serves ID = CAR1 and TYPE = CAR.

For TYPE matching requests, SNDS introduces a novel approach that requires neither changes to NDN nor periodic overhead. Each SNDS node runs an R_SERVICE which is responsible for registrations of *different* TYPE values; this avoids the need to synchronize registries and helps load balancing. The R_SERVICE relies on two special content items: /TYPE_REGISTRY and /TYPE. The /TYPE_REGISTRY name represents a content item that lists all known content providers for a given TYPE. Following NDN principles, it contains *names* that can be used to request content items of that TYPE. The /TYPE name is used to discover new providers of a given TYPE by the corresponding R_SERVICE, so as to populate its registry.

In Fig. 2 we show how the availability of an item with TYPE=CAR and ID=CAR1 is announced. Initially, when the R_SERVICE at the NDN node responsible for the CAR type starts operation, it announces the names /CAR_REGISTRY (msg. 0) and /CAR (msg. 1). When a new content provider makes a content item of that type available, the SNDS service at the closest NDN node first announces the content's ID, e.g., /CAR1 (msg. 2), and then sends an INTEREST for the provider's TYPE, e.g., /CAR/NONCE (msg. 3); the nonce disambiguates INTEREST messages from different providers of the same type.

When the INTEREST message of the provider is delivered to the appropriate R_SERVICE, the R_SERVICE creates a random name for that provider, in our example /RID, adds it to the CAR_REGISTRY file, and responds with a DATA message containing that name to the provider (msg. 4). Finally, the provider announces the /RID name (msg. 5), which serves as an alias for /CAR1 that handles TYPE matching requests.

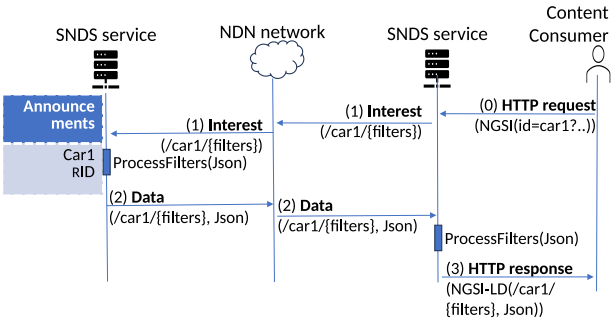

Fig. 3. Requesting content by ID: a consumer requests ID = CAR1.

Fig. 3 shows how ID matching requests are resolved. An IP-based content consumer sends an NGSI-LD API request for ID=CAR1 over HTTP to its closest NDN edge router (msg. 0). The SNDS service translates it to an INTEREST message that concatenates the ID and the specified filters (if any); in our example, /CAR1/FILTERS (msg. 1). The INTEREST message is delivered to the SNDS service at the content provider's end, which returns the structured file in a DATA message (msg. 2). The SNDS service at the content consumer's end translates the response to an NGSI-LD message, encapsulates it into an HTTP response, and returns it to the consumer (msg. 3).

Fig. 4 shows the corresponding solution for TYPE matching requests. We start again with an NGSI-LD API request for TYPE=CAR over HTTP to the closest NDN edge router (msg. 0). If that node is not running the R_SERVICE for this type, an INTEREST message is sent for the corresponding registry, in our example, /CAR_REGISTRY (msg. 1). The node holding the registry responds with a DATA message containing the RIDs for *all* publishers of content with this type (msg. 2). The SNDS service parses this file, sends an INTEREST message

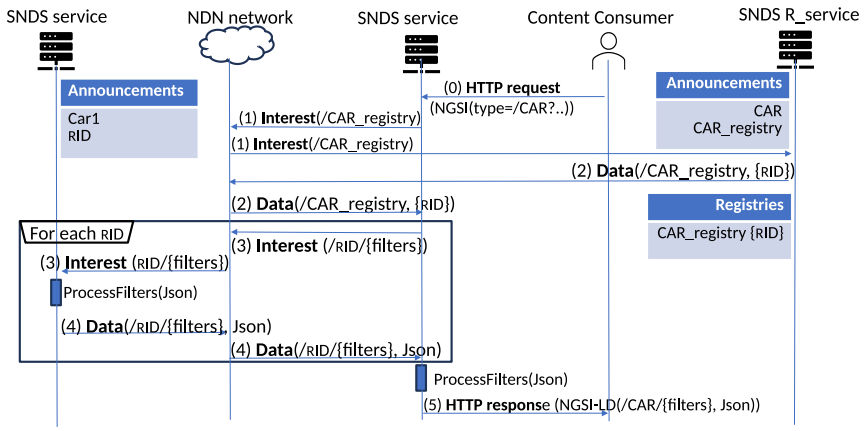

Fig. 4. Requesting content by TYPE: a consumer requests TYPE = CAR.

TABLE I
RESPONSE LATENCY OF CORE SNDS OPERATIONS (MS).

Operation	Collocated	Separated
Announce	452	1122
Request by ID	548	585
Request by TYPE	860	1297

(msg. 3) and receives a DATA message (msg. 4) for each such RID. After receiving all the DATA messages, the SNDS service translates the results into an NGSI-LD response, which is returned encapsulated in an HTTP message (msg. 5).

IV. IMPLEMENTATION AND EVALUATION

Our prototype implementation of the SNDS service is built on top of the open-source NDN implementation [4]. We have implemented the NGSI-LD service, which advertises and fetches content, the proxies which handle NGSI-LD requests encapsulated in HTTP messages, and simple applications for the content consumer and provider. Our implementation is currently running on Mini-NDN, an NDN implementation for the Mininet emulation platform.

Using our prototype implementation and Mini-NDN, we preliminary assessed the latency of the SNDS service. The test NDN topology follows a star pattern, where four NDN nodes running the SNDS service are connected via a single NDN node (the hub), using links with a 100 ms propagation delay. The IP-based consumers and providers are connected to the edge NDN nodes, which are dual-homed.

We measured the response latency of three basic operations: announce content, request by ID and request by TYPE. For each operation, we considered two scenarios: in the *collocated* scenario, the content provider or consumer is attached to the SNDS node whose R_SERVICE handles the corresponding content type; in the *separated* scenario, they are attached to another node. The response latency is the time between messages 2-5 of Fig. 2 for content announcement, and all the messages of Fig. 3 and Fig. 4 for content requests. The results are presented in Table I, showing a small performance penalty in the separated case, since the closest SNDS node must talk to another node handling the appropriate R_SERVICE.

V. CONCLUSIONS AND FUTURE WORK

We presented the SNDS architecture, which provides an efficient content brokering service over an NDN underlay, using NGSI-LD API queries over HTTP for content retrieval. We outlined the components of the architecture, and explained how we translate between the NGSI-LD API queries and NDN operations. We also described our prototype SNDS implementation and provided some initial performance results.

Future work involves deploying our scheme over the global NDN testbed. We plan to assess more extensively the performance advantages of SNDS due to NDN's support for multicast, multisource content provision and ubiquitous caching.

ACKNOWLEDGEMENT

The work reported in this paper has been partly funded by the EU's Horizon 2020 Programme through the subgrant Secure Named Data Sharing (SNDS) (NGISARGASSO-2023-CALL1-9-SNDS) of project NGI SARGASSO (grant agreement No 101092887).

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 ACM CoNEXT*, 2009.

[2] C. Ghasemi, H. Yousefi, and B. Zhang, "ICDN: An NDN-based CDN," in *Proceedings of the ACM Conference on Information-Centric Networking*, 2020, pp. 99–105.

[3] "Context information management (CIM); NGSI-LD API," ETSI Draft. https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.07.01_60/gs_CIM009v010701p.pdf, 2023.

[4] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto *et al.*, "NFD developer's guide," *Dept. Comput. Sci., UCLA, Tech. Rep. NDN-0021*, 2014.