Secure and Efficient Data Spaces: Implementation and Evaluation

Author: Fotios Bistas, fot.bistas@aueb.gr
Advisor: George Xylomenos, xgeorge@aueb.gr
Reviewer: Vasilios Siris, vsiris@aueb.gr
Reviewer: George Polyzos, polyzos@aueb.gr

Master Thesis for the M.S. in Computer Science
Department of informatics
Athens University of Economics and Business
Athens, Greece
January 2026

Contents

1 Abstract

This thesis presents the design and implementation of SeEDS (Secure and Efficient Data Spaces), a standards-compliant Data Space solution aligned with the International Data Spaces Association (IDSA) specifications for Data Spaces, using the European Telecommunication Standards Institute (ETSI) NGSI-LD API specifications. Data Spaces enable independent organizations to securely share and discover data while maintaining control, governance, and sovereignty, in contrast to centralized data platforms or proprietary CDNs.

SeEDS builds upon the Secure Named Data Spaces (SNDS) project, which leveraged the Named Data Networking (NDN) architecture to provide data-centric communication with built-in cryptographic security and in-network caching. NDN's content-based addressing and inherent data authentication make it a natural foundation for decentralized and self-sovereign Data Spaces.

The SeEDS prototype extends SNDS into a complete NGSI-LD-compliant context broker, implementing the full API, including required data operations, content filtering, and optional temporal operations for historical data access. The system was significantly refactored to improve modularity, maintainability, and architectural clarity.

This thesis documents and develops the SeEDS architecture, its internal modules, and the messaging patterns underlying each supported operation. The author contributed to the design and implementation of the complete NGSI-LD API, a persistent storage subsystem, a resilience and failure-recovery mechanism, support for temporal queries, and a modular execution framework that enables efficient integration of new capabilities. Together, these contributions strengthen SeEDS as a robust, decentralized, and standards-based Data Space solution.

2 Introduction

2.1 Data Spaces

A *Data Space*, as defined by the *International Data Spaces Association* (IDSA), is a standards-based way for independent organizations to make their data discoverable and usable to one another, while each party keeps control of its own data. Instead of moving everything into one central database, participants adopt common formats, interfaces, and policies (identity, consent, usage rules) so they can share data securely and auditably across organizational boundaries. This model aims to break down today's "data silos"[1] and enable new services that combine data from multiple sources. A Data Space

[1] Data silos are isolated collections of data that prevent data sharing

differs from a (proprietary) *Content Delivery Network* (CDN), as CDNs focus on fast distribution of content (e.g., web assets, video) through infrastructure operated by a *single* provider, while Data Spaces prioritize interoperability, governance, and data sovereignty across *many* owners with independent infrastructures. To make Data Spaces interoperable, they should implement a standards-based context broker, offering a common API for storing and accessing data. One such option is the NGSI-LD (API), proposed by the *European Telecommunications Standards Institute* (ETSI).

As part of project *Secure Named Data Spaces* (SNDS), the *Mobile Multimedia Laboratory* (MMLab) team implemented a simple Data Space based on the *Named Data Networking* (NDN) architecture. NDN is a data-centric networking paradigm that shifts communication from host-based addressing (as in IP networks) to content-based addressing. In NDN, data items are identified and retrieved directly by *names* rather than by the location of the host that stores them. Each data item is cryptographically signed by its producer, ensuring authenticity and integrity regardless of where it is fetched from. This enables efficient in-network caching, data reuse, and content-centric security; such properties are a natural fit with the decentralized and self-sovereign principles of Data Spaces. Utilizing NDN, the SNDS system supported advanced data-driven applications over a distributed and self-sovereign identity framework.

During the follow-up project, *Secure and Efficient Data Spaces* (SeEDS), the MMLab team extended the SNDS design and implementation prototype to deliver a complete IDSA and ETSI-compliant Data Space solution. The SeEDS prototype implements the entire NGSI-LD API, including all required data operations, with content filtering based on user-defined conditions, as well as the optional temporal operations that enable long-term data storage and retrieval. In addition, the entire SNDS codebase was refactored to improve readability, modularity, and overall coherence.

2.2 The NGSI-LD API

The NGSI-LD API is a standardized interface for managing and exchanging *context information* in data spaces, as specified by the ETSI. It provides a common, HTTP-based framework that lets content providers, consumers, and brokering services create, update, and query *entities*, as well as subscribe to change events. In NGSI-LD, each entity is represented in JSON-LD and usually consists of a globally unique `id`, a `type` (its class), and a set of *attributes*. Attributes are either *Properties* (values, optionally with metadata such as `observedAt` or `unitCode`) or *Relationships* (links to other entities). A JSON-LD `@context` binds the terms used to well-defined IRIs (International Resource Identifier), ensuring semantic interoperability. An example of a content item in JSON-LD format is shown in the following listing:

Listing 1: Example JSON-LD

```
{
"@id": "urn:seeds:Car:001,
"@type": "Car,
"brand":{
    "type": "Property,
    "value": "BW
},
"dateVehicleFirstRegistered": {
    "type": "Property
    "value": "201
    "emissionsCO2":{
        "type": "Property,
        "value": "2
    }
},
"@context":[
    "https://example.com/context.jsonl
]
}
```

The NGSI-LD API supports two retrieval modes over its HTTP API: by `@id` and by `@type`. A *GET by ID* request retrieves a single JSON-LD entity given its globally unique `@id`. A *GET by TYPE* request retrieves all entities that match the provided `@type`. Since `@type` is optional in SeEDS, entities that omit it are treated as having the default `GENERIC` type. The usage of these retrieval modes is as follows:

HTTP usage

- **GET by ID**: returns one entity

```
GET /?id=urn:seeds:car:car001
```

- **GET by TYPE**: returns an array of entities

```
GET /?type=Vehicle
```

- **Implicit type (default)**: entities without @type are treated as `GENERIC`

```
GET /?type=GENERIC
```

Response shape

- GET by ID → a single JSON-LD object (the entity).
- GET by TYPE → a JSON array of JSON-LD entities.

2.3 Named Data Networking (NDN)

In NDN, everything revolves around content items. Content consumers issue *Interest* messages to request content items, which content producers return using *Data* messages; all messages carry the name of a content item. NDN names are hierarchical, but not necessarily human-readable. Content availability is advertised to the network via *Announce* messages, which contain the prefixes of the names of the content items that a content producer serves, for example, `/aueb`.

All NDN nodes (routers and hosts), maintain three data structures. The *Forwarding Information Base* (FIB) maps content name prefixes to the output port(s), also called face(s) in NDN parlance, that should be used to forward Interests towards appropriate data sources; faces can lead to both network interfaces (when the Interest must be forwarded to another node) and local applications (when the application can provide the content item). The FIB is populated by the Announce messages, via a routing protocol. The *Pending Interest Table* (PIT) records the face(s) from which active Interest messages have arrived, i.e., those Interests for which Data messages are still expected as responses. Finally, the *Content Store* (CS) is a local cache for content items.

When an Interest is received by an NDN node, the node extracts the content name and checks if the requested content item has been cached in the CS or is locally hosted; if so, the content item is returned in a Data message through the incoming face and the Interest is discarded. Otherwise, the PIT is checked to see if a previous Interest for that name is pending; if so, the PIT entry for that name is updated to also point at the face from which the Interest arrived, and the Interest is discarded. If this also fails, the FIB is checked to decide where to forward the Interest. If a matching entry is found in the FIB, the Interest is sent through the appropriate face and its incoming face is added to the PIT. Data messages are routed back to the consumers hop-by-hop, using the pointers stored by the Interests in the PITs; when a Data message is received, we look up in the PIT where the corresponding Interest(s) came from, forward the Data through the face(s), and delete the PIT entry; if the PIT entry contains multiple faces, this leads to Data multicasting.

NDN supports the cryptographic binding of names to content items, by including in each Data message a signature, covering both the name and the content, as well as a pointer to the key used for signing. Any NDN node can therefore verify the binding between the name and the content item included in a message.

NDN is the network underlay of the SeEDS prototype. It provides connectivity between the SeEDS nodes in an information-centric manner, offering native multisource, multipath, and multicast content dissemination. The SeEDS logic is implemented by the SeEDS service running in the SeEDS nodes, while the network is built upon regular NDN routers. The NDN edge routers, which run the SeEDS service, act as gateways to the IP-enabled end-users, hence presenting dual-architectural compliance, supporting both the NDN and TCP/IP protocol stacks.

2.4 Contributions

This thesis documents the SeEDS design and implementation, providing details on the overall design and modules involved, the messaging patterns for each operation and their underlying implementations. The SeEDS design and implementation were a collaborative effort from the entire SNDS and SeEDS teams. A first version of the design and its implementation was created during the SNDS project. During the SeEDS project, additional capabilities were designed and implemented on top of SNDS, and the codebase was largely rewritten.

The author of this thesis joined the team during SeEDS, and contributed to the design and implementation of a persistent storage system that allows nodes to store and serve data, the development of a resilience mechanism for recovering from node failures, the introduction of new request types such as *Temporal Queries*, and the creation of a modular framework that enables the efficient integration and execution of these extensions, along with refactoring pre-existing ones.

3 Design

3.1 The SeEDS Service

While the SeEDS Data Space implementation works over an NDN network, the NGSI-LD API uses HTTP, with clients and servers operating in an IP network. The SeEDS Service is software that bridges the IP and NDN networks, as shown in Figure 1. First, it offers an HTTP endpoint for exchanging NGSI-LD API messages with IP-based systems. Second, it connects to the *NDN Forwarding Daemon* (NFD) through a netlink socket to exchange NDN-specific messages with NDN nodes. Each SeEDS Service contains a storage instance, where it stores data like locally published IDs, subscribed Nodes, etc. To make data available across the Data Space, *Rendezvous* (RV) Nodes are responsible for locating and aggregating content, serving collections of data upon request. In our system, RV Nodes are implemented through the Broker instances described later.

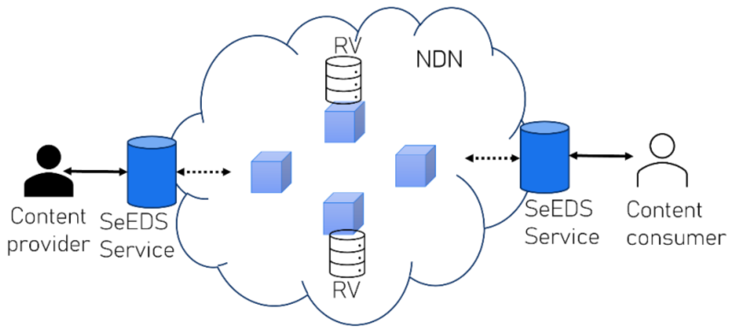


Figure 1: High level SeEDS architecture.

Figure 2 shows how the SeEDS Service is decomposed into its three logical modules: the NGSI-LD interface towards the IP world, the NFD interface towards the NDN world, and the local storage.

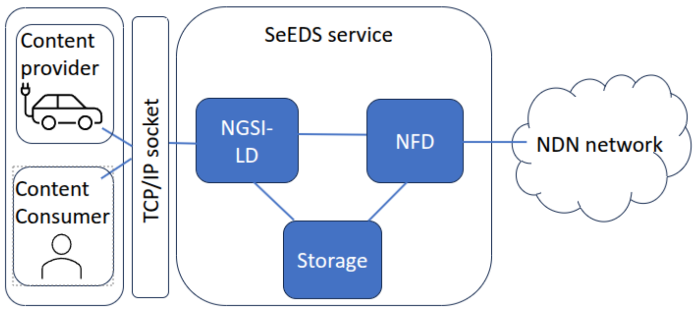


Figure 2: SeEDS Service in detail.

The SeEDS Service performs the following operations:

- Consumers can create a *HTTP GET* request to retrieve items from the NDN network. As mentioned before, these are either *GET by TYPE* or *GET by ID*. Note that the *@type* and *@id* need to be provided by the request.
- Providers can create *HTTP POST* requests to provide/announce items to the NDN network. The service ensures that the data is stored at an SeEDS node, ensuring that it can be returned when requested from some other node in the network.
- Clients can subscribe to a specific *@id* or *@type*. Since content in NDN is retrieved by its unique name rather than by host address, clients may express a request for a name even before the corresponding data is available. Once data is published under that name, the network delivers it to all subscribed clients, enabling efficient and timely notifications.
- To ensure resilience, any SeEDS node can act as a *Secondary* for a *Primary* node within the network. A *Secondary* node mirrors the data of its corresponding *Primary* node, including its storage, announced *@ids*, *@types* and *subscriptions*. This design allows SeEDS to recover the data of a failed node seamlessly, maintaining service continuity.

3.2 ID and TYPE matching

The NDN architecture can easily support the NGSI-LD requests that follow the GET by ID matching approach, as the (unique) ID of the NGSI-LD item can be mapped to a unique NDN content name. The process involves the following steps: the provider announces the content item to the NDN nodes using as a name the value of the ID field of the NGSI-LD file; the Announce messages populate the correlated FIB entries that will route the Interests for that name to the provider; the consumer emits an Interest message using as the name the ID of the NGSI-LD file; the NDN network routes the Interest packet to the provider; finally, the source (the provider or an on-path cache) responds to the Interest with a Data packet that is routed via the PIT entries (that the Interest message populated) to the consumer.

On the other hand, the NDN architecture does *not* natively support the NGSI-LD requests that follow the TYPE matching approach. Even though NDN inherently supports multisource, that is, getting a *single* piece of content from multiple sources, it does not support the gathering of *multiple* pieces of content from various sources based on a common feature, such as their TYPE. For TYPE matching requests, the SeEDS service at each node monitors the different content providers for each TYPE, with a novel communication scheme that does not require probing.

Specifically, the SeEDS Service running on each SeEDS node maintains a *registry* for one or more TYPEs; each TYPE is mapped to one node only. This decision avoids the need to synchronize registries, helps load balancing and, in turn, scaling the SeEDS. This operation is built over two meta-files/names, `TYPE_registry` and `TYPE`:

- `TYPE_registry`: This name is associated with a content item that lists all the known content providers for a given TYPE. Following NDN principles, it does not hold location-dependent identifiers, only names that can be used to request data items of this TYPE. The registry file is maintained by the SeEDS service that is responsible for this TYPE, and is transmitted to other SeEDS services that need to resolve a TYPE matching request.

- `TYPE`: This name is used to discover new providers of a given TYPE by the SeEDS service, so as to populate the registry file.

The end result is that the content IDs for all items with a specific type are maintained by one SeEDS node; when a GET by TYPE query is issued, it is sent to this node, which responds with the corresponding IDs, allowing the requester to then ask for each content item using the GET by ID primitive.

3.3 Temporal Queries

Temporal queries are a powerful tool of NGSI-LD that allows content consumers to retrieve past versions of mutable content items, that is, items that can change over time. To conduct a temporal query, the content consumer emits a request for a content ID or TYPE, adding a time-reference regarding the content's creation and some additional parameters. Specifically, the consumer specifies (a) the number of versions desired, (b) a date and (c) a before/after condition, e.g., 5 versions after 1.1.2025, or 1 version before now (implying the latest version). The SeEDS network then finds the set of versions of the content item(s) that fit the query criteria and returns them.

Implementing temporal queries over NDN is not straightforward, since in NDN content names are permanently linked to content items. Therefore, the mutations (or different versions) of an item must be named differently, otherwise caching and forwarding, which operate solely based on names, will malfunction. Algorithmic naming, such as `/SeEDS/item1/<version>`, is the conventional solution for these issues in NDN, where `<version>` can be a sequence number or the date of creation.

Each SeEDS service uses a local storage module that enables versioning of stored content items: when a content provider periodically pushes new versions of a content item to SeEDS, the approppriate SeEDS service updates the corresponding item in the storage. Since Interest forwarding uses longest prefix matching, an NDN publisher can announce one root name, such as `/SeEDS/item1/`, receive Interests for sub-names, such as `/SeEDS/item1/version1` or `/SeEDS/item1/items=1&after=date1`, and respond with Data packets for the sub-names that will follow the (reverse) route of the corresponding Interests. Therefore, a SeEDS Service can announce a versioned content item only once, using its root name, and locally store all the versions of the content item. When the Service receives Interest packets with a temporal query included in their name, it can process the query and send back the correlated Data packets.

The payload of the Data packets depends on the query. If the data items to be transmitted are relatively small and time-sensitive (hence non-cacheable), such as temperature measurements, or the query corresponds to a specific item, such as a request for a particular version, then the payload can include the actual data; we call this *direct* mode. If the response includes multiple, large files, such as, multimedia files or ML models, or the request does not specify the exact item identifier, then the payload can include a metafile with the ID(s) of the item(s). After receiving the metafile, a SeEDS Service can explicitly request the included IDs; we call this *indirect* mode. The use of a metafile, which is common practice on Internet protocols, such as DASH and BitTorrent, comes with the latency penalty of one additional RTT, since the SeEDS Service must first get the metafile and then fetch the actual data. However, it can significantly reduce traffic footprint, as it assures that information items are transmitted using their individual names, thus being compatible with NDN's on-path caching and multicast. The selection of the payload type can be made at the SeEDS Service of the provider based on the particularities of the content; currently, we set a simple rule based on number of items and type of request. However more advanced techniques, such as AI-based solutions, can also be adopted.

In SeEDS, temporal queries are not optional: all requests must indicate the parameters for temporal query processing, namely, the number of versions, a date and a before/after condition. By default, these variables are set to “1”, “now” and “before” thus requesting the latest and, possibly, only version of an item. Versioning is completely transparent to the NDN network, which lacks support for mutable items; it is purely implemented by the SeEDS Services.

3.4 Content filtering

SeEDS supports attribute-based filtering, where consumers request a subset of attributes of the structured files, thus offering more fine-grained data transmissions. Filtering is allowed for all requests and is specified as a query parameter; by default, filtering parameters can be omitted, thus indicating a request for all attributes. Regarding implementation, two alternatives were considered: network-level and application-level.

The network-level solution makes network functions (routing, forwarding, caching, etc.) aware of the filters; this means that the attributes and the corresponding conditions are integrated in the name identifiers and affect the routing, forwarding and caching decisions of each NDN router. For instance, a provider may announce item with ID `/car1` and receive all Interest packets with names that include the parameters of the temporal query and the filters, e.g., `/car1/temp_query_params/filtering_params`. The SeEDS service of the provider will find the items that fit the temporal query parameters, apply the filtering requests, and send back the filtered data in Data packets with the same name. This solution utilizes bandwidth more efficiently, since only the requested data items are transmitted, but presents two weaknesses.

First, it significantly challenges on-path caching and multicast transport since requests are too specific; only requests for the same attributes from the same versions of the same item can be aggregated. Second, the security primitives that verify a user’s permission to access the selected fields need to be transmitted from the consumer’s SeEDS service to the provider’s SeEDS service, thus creating privacy risks.

The application-level solution makes the filters transparent to the network, delegating their processing to the consumer’s SeEDS service. The filters are indicated in the query of the consumer but are not included in the corresponding NDN Interest message. The complete structured files are delivered to the consumer’s SeEDS service and then, the filters are applied before returning the data to the consumer. This approach can lead to poor network utilization, as the structured files can be transmitted and then be dropped entirely or partially due to the conditional or selective filters, but it increases the effectiveness of in-network caching and multicast transport, which are expected to offer more gains.

For the initial implementation of the SeEDS architecture, we implemented the application-level solution, which is more straightforward, as it does not require changes in the internal signalling the SeEDS network; instead, all filter processing is implemented at the SeEDs service receiving a query. We may reconsider this decision in the future.

3.5 Event notifications (subscriptions)

Event notifications are emitted to content consumers to signal that a previously requested content item has become available. To receive a notification, a consumer must send a subscription to the SeEDS network. A subscription concerns a specific content ID or TYPE and includes a response URL, which is used to post the event back to the consumer. Subscriptions are "one-off", that is, the SeEDS network stores the subscriptions until the consumer unsubscribes, or until the requested content is posted.

The implementation of event notifications over NDN is not straightforward, due to the Interest lifecycle in NDN. NDN discards Interest packets either when the corresponding content name has not yet been announced, or when the matching Data packets are delivered, thus Interests that precede announcements are discarded. Thereupon, the SeEDS network introduces an over-the-top state management solution for storing subscriptions, until either the requested content becomes available, or the user sends an unsubscribe request.

Specifically, SeEDS introduces a registry for subscriptions, implemented using RV nodes. Similarly to the TYPE registry, the subscription registry is stored in a distributed fashion, with each node being responsible for a unique subset of the ID space. In our implementation, the subscription for a specific item is stored at the node that manages the ID numerically closest to the item's ID. When a matching announcement appears in the SeEDS network, the subscription registry responsible for that ID will notify the SeEDS Service that issued the subscription, so that it may ask for the corresponding content item.

For example, assume that an IP-based user connected to SeEDS Service 1 requests item "item1". Service 1 will emit an Interest for this item and, if the item was previously published, the corresponding Data will be returned; this is a typical request by ID. If the item was not previously published, then the NDN network will return an error, indicating that "no route is available" for this item, thus triggering the subscription mechanism. A user can also be subscribe to data by performing an HTTP request. Service 1 will then notify the SeEDS Service responsible for "item1" of the existence of a local subscriber; the responsible SeEDS Service will record that "a subscriber for 'item1' is connected to Service 1" in its subscription registry file. When "item1" is announced at (say) SeEDS service 2, Service 2 will also notify the service responsible for "item1" that a content item for that ID just appeared.

Thereafter, the responsible SeEDS Service will match the consumer and subscriber requests, notify Service 1, and the latter will emit an Interest packet for "item1" which can now be satisfied.

By definition, subscriptions are not allowed for past versions of an information item. Subscriptions correspond to the first version and the latest version of a yet unpublished and published information item, respectively. Note that as subscriptions are removed after being satisfied, a consumer that is interested in always receiving the latest version of a mutable content item, will need to renew its subscription after receiving each new version.

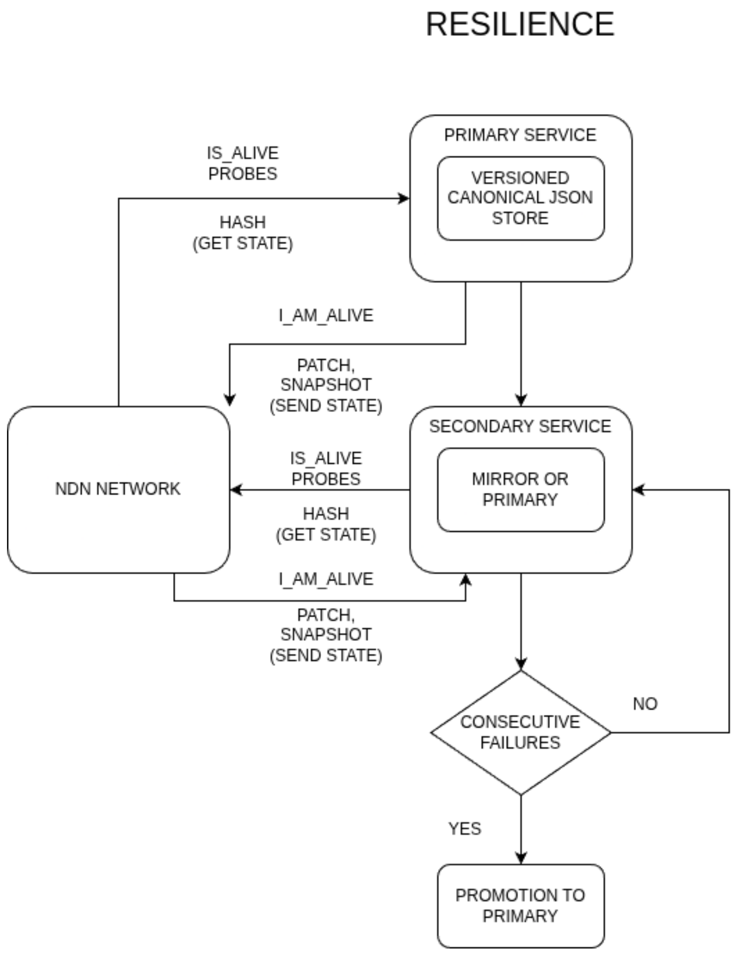


Figure 3: Resilience architecture in SeEDS.

3.6 Resilience

To ensure the continuous availability of content, the information maintained by each SeEDS service (content items and registries) can be mirrored by another SeEDS service; the latter can take the former's role during failures. The resilience layer in SeEDS ensures that a *Secondary* node continuously converges towards the state of a *Primary* node through lightweight synchronization protocols built over NDN. To achieve this, we had to rethink how state is maintained and monitored at each node, in order to allow synchronization between the Primary and Secondary nodes. At the core of the mechanism lies a versioned canonical JSON store on the Primary, which exposes its state through a set of standardized NDN Interest and Data packets. The Secondary maintains a local mirror of this store, applying updates received from the network and validating them against deterministic fingerprints.

The Primary store maintains a bounded history of mutations. Each change increments a version counter, recomputes a canonical hash of the entire state, and records a snapshot to support future patch generation. It responds to queries with one of three data views: a lightweight metadata descriptor containing only the version and hash; a delta, represented as an RFC 6902 JSON Patch from one version to another; or a full snapshot of the canonical state. The Secondary mirroring as Primary accepts these responses and applies them locally. Whenever patch application fails, or when the requested patch is unavailable, the mirror falls back to a snapshot, ensuring eventual consistency, even in the presence of divergence.

The resilience system is designed to handle failures gracefully. The Secondary periodically polls the primary and if it records consecutive failures that exceed a configurable threshold, the Secondary promotes itself to Primary and terminates the polling loop. This self-promotion is communicated to the local services through IPC, ensuring that the system can continue to operate without interruption despite the failure of the original Primary Figure 3 shows how the mechanism operates in the form of a diagram.

4 Implementation

The implementation described in this thesis reflects the state of the SeEDS system as of October 2025. As the project continues to evolve, certain design choices, component interfaces, or architectural details may change to accommodate new requirements, optimizations, or future redesigns. Therefore, this documentation should be considered a snapshot of the current implementation rather than a definitive or final version.

For each request type, a dedicated subsection is provided to illustrate the internal operations that occur within the codebase – for example, message parsing, data publication, registry updates, and communication between components. Each request type is first presented from a functional perspective (which is less likely to change in the future) , followed by a detailed explanation of its internal execution flow (which is much more likely to change).

Some of the core functionalities of the SeEDS prototype, including *Announce Content*, *GET by ID*, *GET by TYPE*, and *SUBSCRIBE by ID*, had partially been implemented in earlier iterations of the codebase (during the SNDS project). However, these implementations were originally developed in a tightly coupled and less modular manner, which made the internal logic difficult to follow and maintain. As part of this thesis, these components were carefully refactored to improve readability, modularity, and overall coherence. However, the naming conventions of the original SNDS codebase were retained for the SeEDS version, so we use the terms SNDS and SeEDS interchangeable through the remainder of this thesis to denote the current Data Space implementation.

4.1 General Implementation Details

The SeEDS prototype consists of five essential components, each with its own functionality, along with eight custom libraries that achieve specific goals. Figure 4 shows the relationships between these components, which are explained in more detail in the following.

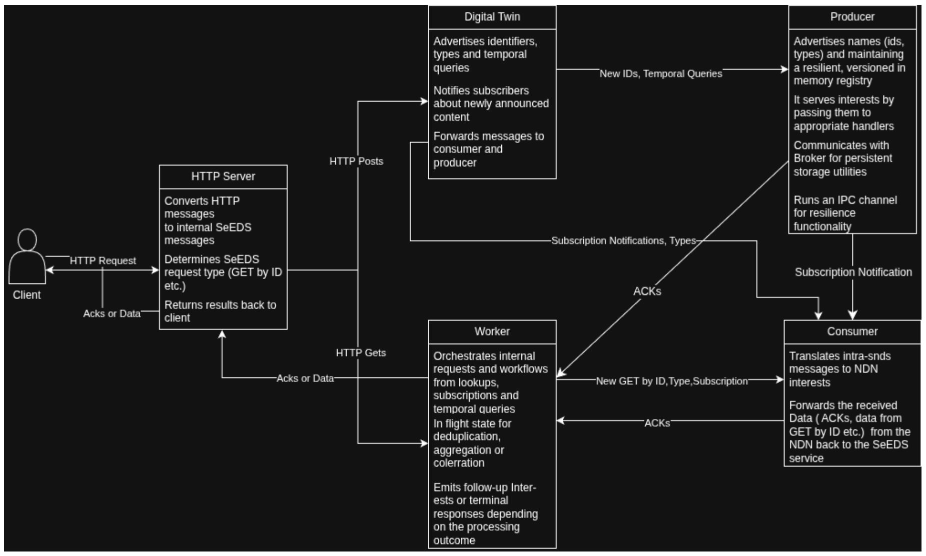

Digital Twin
Advertises identifiers, types and temporal queries
Notifies subscribers about newly announced content
Forwards messages to consumer and producer
Producer
Advertises names (ids, types) and maintaining a resilient, versioned in memory registry
It serves interests by passing them to appropriate handlers
Communicates with Broker for persistent storage utilities
Runs an IPC channel for resilience functionality
New IDs, Temporal Queries
HTTP Posts
HTTP Server
Converts HTTP messages to internal SeEDS messages
Determines SeEDS request type (GET by ID etc.)
Returns results back to client
Client
HTTP Request
Acks or Data
Subscription Notifications, Types
ACKs
Subscription Notification
Worker
Orchestrates internal requests and workflows from lookups, subscriptions and temporal queries
In flight state for deduplication, aggregation or colerration
Emits follow-up Interests or terminal responses depending on the processing outcome
HTTP Gets
Acks or Data
New GET by ID,Type,Subscription
ACKs
Consumer
Translates intra-snds messages to NDN interests
Forwards the received Data (ACKs, data from GET by ID etc.) from the NDN back to the SeEDS service

Figure 4: Basic relationship between the SeEDS components.

1. HTTP server (`snds_http_server.cpp`). It provides the HTTP interface for sending and receiving SeEDS messages using `cpp-httplib`. It converts incoming HTTP requests intra-SeEDS messages and returns the corresponding responses over HTTP. The server parses query and body parameters using the `QUERY_PARAM_LIST` macros, determines the request type (e.g., *GET_BY_ID*) based on the HTTP verb and parameters, sends the resulting message to the appropriate service queue, and finally waits for the response, so as to to return its payload in an HTTP response. The implementation is available in the following files:
   - `snds_http_server.hpp`
   - `snds_http_server.cpp`
2. Worker (`snds_worker.cpp`). It orchestrates the internal request and response workflows for subscriptions, lookups (by ID or TYPE), and temporal queries, which are all `GET` requests. It receives intra-SeEDS messages from multiple components: the HTTP server (the `GET` requests), the Consumer component, which expresses interests on the NDN network, receives the resulting data and either sends back different ACK types or the data itself to the Worker, and the Producer component, which sends different ACK types back to the Worker. It also maintains lightweight in-flight state for operations like deduplication, correlation, and aggregation. The internal state consists of the following STL containers:

- `subscribed_by_id_users`
- `subscribed_by_type_users`
- `pending_ids_to_type_map`
- `payloads_for_get_by_type`
- `pending_get_by_id_requests`
- `pending_get_by_type_requests`
- `pending_temporal_queries`
- `temporal_buffers`

It also emits follow-up Interests or terminal responses, depending on the processing outcome. For example, a GET by TYPE request causes the transmission of a `REGISTRY_PAYLOAD` intra-SeEDS message, which contains the `@ids` for the corresponding `@type`, to the Worker, which proceeds to perform a GET by ID for each returned `@id`. The implementation is available in the following files:

- `snds_worker.hpp`
- `snds_worker.cpp`

3. Digital Twin (`snds_digital_twin.cpp`). It is responsible for advertising identifiers, types, and temporal queries within the SeEDS system. It listens for intra-SeEDS service messages (advertisements) such as `BY_ID_ADVERTISEMENT`, `BY_ID_AND_TYPE_ADVERTISEMENT` and `TEMPORAL_QUERY_ADVERTISEMENT` and generates the corresponding NDN Interests and NDN names through helper utilities. It forwards these to either the Consumer component to express Interests or to the Producer component to announce NDN names. In the case of an advertisement by ID, it first passes on the `@id` to the Producer to advertise it (that is, announce the NDN name). After that it finds, through an agreed upon hash function, the appropriate scope (meaning the node) of the `@id` to inform any subscribers to that `@id` that it was announced in the network. This is done through an Interest after passing onto the Consumer. In the case of an advertisement by ID and TYPE, it first forwards the new `@id` and its corresponding `@type` to the Consumer, to allow the Consumer to find (through an NDN interest) the responsible node for the `@type` and then insert the new `@id` into the `@type` registry. It also forwards a notification message to the Consumer, so that the node that is responsible for the `@type`, receives it and notifies the subscribers of the `@type` that new data is available, also via an NDN interest. Lastly, for a Temporal Query (query is not exactly accurate here as its more like a Temporal Announcement) it simply passes it on to the Producer to be advertised. Finally, it maintains a record

of locally announced IDs to prevent duplicate advertisements on the SeEDS network. The implementation is available in the following files:

- `snds_digital_twin.hpp`
- `snds_digital_twin.cpp`

4. Consumer (`snds_consumer.cpp`). It is the consumer-side service responsible for translating (or just simply receiving, if they are in the correct format) intra-SeEDS messages from the Worker and Digital Twin components, into NDN Interests and forwarding the received Data back into the intra-SeEDS service bus. It listens for many types of intra-SeEDS messages – such as those derived from `GET`/`POST` requests, which are essentially GET by ID and GET by TYPE requests, subscription events, meaning notifications, and new subscriptions, or temporal queries, and converts them into NDN Interests using the parsed-interest utilities. Upon receiving Data from the NDN network, the Consumer either forwards the results back into the worker queue as intra-SeEDS messages or, if this is not needed, it returns different types of ACKs. For example, when a node is successfully notified via a subscription (i.e., it has subscribed to a specific `@id`), it sends an ACK to confirm receipt. This ACK is sent back to the node's Consumer that received the new Data and issued the notification. The implementation is available in the following files:

   - `snds_consumer.hpp`
   - `snds_consumer.cpp`

5. Producer (`snds_producer.cpp`). It acts as an NDN producer in the SeEDS system. It is responsible for advertising names, received through the Digital Twin component; these can advertise/announce `@types`, `@ids` and temporal announcements. The NDN names announced for `@ids` and `@types` are registered in the in-memory STL containers and stored in the Broker's (check the 4.1 snds_libraries section for a description of the Broker) permanent storage. These containers are:

   - `TypeRegistry`
   - `SubscribersByID`
   - `SubscribersByType`
   - `Announcements`

   The Producer also serves Interests directed towards its announced names, by continuously listening to the NDN network. This means that if a Node has announced a new `@id=car1`, the Producer Component listens for Interests directed towards this name. For example, when a GET by ID request for `@id=car1` arrives at a node, the node expresses an Interest for `@id=car1`. The responsible node then processes this Interest using the appropriate handler. Finally, the Producer maintains a resilient, versioned in-memory registry, which is essentially the implementation of the versioning scheme for the resilience mechanism. It tracks the STL containers' states and the HTTP Broker's state (for the HTTP Broker description check the 4.1 snds_libraries section) . For example, when a new subscription is inserted into the `subscribed_nodes_for_id` container, the versioning scheme advances the version and records the operation required to reach the new state. In this case, the operation corresponds to adding a new subscription. The Producer also stores and retrieves data from the HTTP Broker that handles queries such as `GET`/`POST`, temporal query, or State, and maintains a bounded journal of the Broker's state for rehydration or failover recovery. When resilience mode is enabled, it operates an IPC server that manages synchronization, snapshots, and role transitions between the Primary and Secondary states. The implementation is available in the following files:

- `snds_producer.hpp`
- `snds_producer.cpp`

The implementation contains 8 libraries that assist with different types of functionality, which are reused across components.

1. `snds_interest`: This library provides utilities for creating and parsing SNDS-style NDN Interest names. It defines a taxonomy of Interest "command" components (e.g., `temporalQuery`, `version`, `getByType`) via the `INTEREST_MESSAGE_TYPE` X-macro list and its corresponding enum and string table. The core type, `ParsedInterestName`, represents a decomposed Interest name with explicit fields for the prefix, logical name, command, and parameter list. The Interest created by this library has the form `prefix/name/command:param1&param2&`. The library includes helpers for URL encoding/decoding, splitting and normalizing `key=value` parameters, converting an `ndn::Name` into a `ParsedInterestName`, and reconstructing custom NDN names or HTTP-style query strings from the parsed structure. Together, these utilities ensure that all SeEDS components interpret and construct Interest names in a consistent way. The implementation is available in the following files:

   - `parsed_interest_name.hpp`
   - `parsed_interest_name.cpp`

   In addition, the header `parsed_interest_to_temporal_query.hpp` provides a mapping layer that converts a `ParsedInterestName` into a `temporal_query::TemporalQuery` object. It interprets each parameter token as a `key=value` pair, decoding percent-escapes (due to the URL-encoding) and populating the temporal query fields such as `id`, `type`, `date`, `timeframe`, `count`, and `filters`, while preserving unknown keys in an `extras` map for forward compatibility. This enables a direct, convention-based translation from NDN Interest naming to the temporal-query semantics used by the system. The implementation is available in:

- `parsed_interest_to_temporal_query.hpp`

2. `snds_inter_process_communication`: This library provides low-level utilities for the TCP-based inter-process communication used by the SeEDS resilience subsystem. It offers helper functions for encoding and decoding 32-bit integers in big-endian format, which are used to frame messages with a fixed-length header. In addition, it provides a templated `send_json` routine that serializes `nlohmann::json` objects, prefixes them with a 4-byte length field, and transmits the resulting frame over any synchronous `Boost.Asio` stream (such as a TCP socket). This library forms the basis of the lightweight IPC protocol used for exchanging snapshots, patches, and control messages between Primary and Secondary roles. The implementation is available in:

   - `snds_ipc_tcp.hpp`

3. `snds_broker`: The Broker is a lightweight HTTP server backed by MongoDB that provides persistent storage and index-based retrieval for the SeEDS system. It accepts JSON payloads through the `/add` endpoint and writes them to both versioned, which contains multiple timestamps for a single `@id` and unversioned, which contains the latest timestamp for an `@id` collections, enabling temporal queries. For example if the current time is 31/01/2026 and a new JSON payload arrives and it has no specified timestamp, it will insert the 31/01/2026 timestamp in the unversioned collection overriding the previous entry and additionally append it to the versioned collection. The server also exposes several read interfaces, including `/get` for insertion order latest retrieval, `/temporal-query` for time-sliced historical reads, and `/state` for computing per-collection fingerprints, through a specified hash function. The `temporal-query` API is the most important, allowing the `Producer` component to retrieve records based on specified timestamps. For example it can ask the `Broker` to retrieve the 10 closest records to the 31/01/2026 timestamp. The broker also offers dump and restore functionality via `/dump`, and supports mirroring data into secondary collections for use in the resilience subsystem.

This means that the dump API exports the database contents into a JSON array and is used for transmitting the collections into secondary nodes. Low-level utilities for timestamp parsing, canonical JSON normalization and hashing, aggregation pipeline construction, and collection dumping are factored out into `broker_utils.cpp`. Together, these files implement the authoritative persistence layer of SeEDS, enabling versioning, replay, and consistent state recovery across system restarts or failover events. The implementation is available in the following files:

- `broker.hpp`
- `broker.cpp`
- `broker_utils.cpp`

4. `snds_query_params`: This library provides a small set of helpers and X-macros for declaring, extracting, and applying HTTP query parameters in a consistent way across the entire SeEDS codebase. It defines canonical string constants for all supported query keys (such as `id`, `type`, `date`, `timeframe`, `count`, `filters`, and subscription-related flags), together with a traits-like `QueryParam<T>` extractor specialized for `std::string` and `bool`. The central X-macro list `QUERY_PARAM_LIST` acts as the single source of truth for all parameters, specifying their local variable name, wire key, whether they should be URL-decoded, and their C++ type. On top of this list, the header builds convenience macros to declare a full set of `std::optional<...>` variables, extract their values from an `httplib::Request`, and apply the present parameters to higher-level message objects (such as `SNDS_INTRA_MESSAGE`) via generated `set_<param>_` setters. This design keeps parameter handling centralized and avoids duplication of parsing logic in individual services. The implementation is available in:

   - `query_params.hpp`

5. `snds_message`: This library defines the intra-service message taxonomy and the lightweight container used to exchange messages between the SeEDS components mentioned above. It uses X-macros to specify all intra-message kinds, HTTP verbs, control/ACK types, and client roles in a single place, automatically generating the corresponding enums and string tables, along with generic `to_string` and `from_string` helpers. The core value type, `SNDS_INTRA_MESSAGE`, encapsulates the primary message body, an optional payload, the message type, sender role, HTTP method, and a set of strongly-typed optional parameters generated from `QUERY_PARAM_LIST`. It provides utilities to infer the message type from the HTTP verb and the presence of specific parameters (for example, distinguishing `GET_BY_ID_REQUEST`,

BY_TYPE_SUBSCRIPTION, or TEMPORAL_QUERY_REQUEST), as well as to pack/unpack parameters into a compact human-readable {"k=v,k2=v2"} representation and to pretty-print messages for debugging. This library serves as the single source of truth for intra-message (between components) semantics across the SeEDS codebase. The implementation is available in:

- snds_message.hpp

6. snds_queue: This library provides the SeEDS Service with a thread-safe, producer–consumer message buffer used internally for inter-thread coordination and batching of SNDS_INTRA_MESSAGE objects. It wraps a std::deque protected by a mutex and condition variable, allowing writers to enqueue messages asynchronously while readers block until data becomes available. A shutdown flag is exposed so that worker threads can safely terminate even when no further messages are pending. Beyond the basic readMessage() and writeMessage() operations, the queue also supports non-blocking reads as well as two batch-drain operations: tryReadAll() and waitAndReadAll(). These atomically extract all queued messages and serialize them into a compact JSON array, embedding each message's name, type, sender role, and payload (parsed as JSON when possible, or wrapped as raw text otherwise). The batch is returned as a synthetic SNDS_INTRA_MESSAGE with a GENERIC type and a textual BATCH envelope, enabling efficient downstream transport or logging without exposing internal queue state. This design provides both low-latency single-message delivery and high-throughput batch extraction under load. The implementation is available in the following files:

   - message_queue.hpp
   - message_queue.cpp

7. snds_resilience: This library implements a lightweight versioned JSON store, together with a follower-side mirror. Its primary goal is to maintain a canonical application state, track its evolution over time, and efficiently synchronize remote replicas using compact RFC 6902 JSON Patch deltas. At the core of the module lies the Store, which holds a canonical JSON object, alongside a monotonically increasing version counter and a deterministic hash derived from a stable serialization. Each state change – whether performed incrementally via mutate() or through full replacement via assign() – automatically bumps the version, recomputes the canonical hash, and stores a bounded snapshot in a rolling history used for patch generation. When a consumer provides a known base version, the store attempts to pro-

duce a minimal diff; if the version is no longer retained, callers can fall back to serving a full snapshot.

The complementary `Mirror` class maintains a local copy of that state on the follower side. It can ingest entire snapshots or apply patches when the base version matches, normalizing hash inconsistencies if they arise. In both components, optional logging via `spdlog` provides visibility into version deltas, patch operations, and integrity checks. Finally, utility routines for canonical JSON dumping and MD5 fingerprinting ensure deterministic behavior across distributed nodes. The implementation is available in the following files:

- `snds_resilience.hpp`
- `snds_resilience.cpp`

The `canonical` utility sub-library provides small, composable helpers for producing deterministic JSON encodings of common C++ container structures. These helpers are used throughout the resilience and registry subsystems to ensure stable serialization, reproducible hashing, and consistent comparison across nodes. The library offers three core templates:

- `sorted_vector`: Converts any set-like container into a lexicographically sorted `std::vector`, guaranteeing deterministic element ordering.
- `map_of_sets`: Converts a `map<string, SetLike>` into a canonical JSON object by sorting both the map's keys and every associated set, producing a fully ordered structure suitable for hashing or replication.
- `set_to_json`: Serializes any set-like container into a sorted JSON array.

These utilities form part of the canonicalization layer used in the resilience module (e.g., when hashing state or producing patches), ensuring that logically equivalent states always serialize to the same byte representation. The implementation is available in:

- `canonical.hpp`

The library also contains the `ResiliencePoller`, which is the active client-side component that drives the resilience protocol over NDN. It attaches to an existing `ndn::Face` and periodically probes a nodes's Producer for both liveness and state evolution. Internally, it runs two threads: an *alive loop*, which sends `GET_RESILIENCE_IS_ALIVE` Interests carrying a random nonce and verifies that the peer echoes it back, and a *meta loop*, which issues `GET_RESILIENCE_META` Interests to retrieve the remote head metadata (version, hash, timestamp). All network operations are posted onto the Face's `io_context`, respecting ndn-cxx's threading model.

Upon receiving metadata that indicates the remote version has advanced, the poller requests a `GET_RESILIENCE_PATCH` from the last locally applied version to the advertised head, and hands the resulting JSON payload to an application-provided `on_apply_patch` callback. If patch application fails, or if the patch is NACK'ed or times out, the poller transparently falls back to fetching a full snapshot by requesting a `GET_RESILIENCE_SNAPSHOT` and passing it to `on_apply_snapshot`. A set of callback hooks (`on_alive`, `on_meta`, `on_error`, `on_patch_applied_ok`, `on_snapshot_applied_ok`) allows the surrounding code to update local mirrors, expose health signals, or log synchronization events. The poller also tracks the last liveness result and last seen remote metadata, providing a compact observability surface over the resilience channel. The implementation is contained in the following files:

- `snds_resilience_poller.hpp`
- `snds_resilience_poller.cpp`

8. `snds_temporal_query`: This library defines the in–memory model and conversion utilities for temporal queries issued against the SeEDS data stores. At its core it introduces the bit–mask enum `Timeframe`, which supports combining timeframe flags such as `NOW`, `BEFORE`, and `AFTER` in a single value, a lightweight `Filter` structure for key–value constraints, and the main `TemporalQuery` aggregate that collects the query id, type, reference date, timeframe, optional result count, arbitrary "extra" key–value pairs and a list of `Filters`. The implementation also provides a symmetric pair of converters between the bit–mask representation and its textual form (e.g., `"now+before+after"`) and helpers for packing and unpacking the filter list to a compact string representation.

   Beyond the in–memory model, the library offers a family of encoding and parsing routines so that the same `TemporalQuery` can be transported over different protocols without losing information. It can be serialized as a canonical query string (`k=v&...`), as a compact JSON object with well–defined fields and inlined extras, or as a path–like form suitable for NDN names (e.g., `/type/id/date=.../timeframe=...`). The corresponding decoding functions (`deserialize_temporal_query`, `params_from_querystring`, `params_from_pathlike`) reverse these encodings and populate a `TemporalQuery` instance while routing unknown keys into the `extras` map. Finally, the header defines stable hashers for both Filter and TemporalQuery (and registers them as std::hash specializations), enabling their use as keys in unordered containers or cache indices based on the full query semantics rather than on a particular wire encoding.

The accompanying header parsed_interest_to_temporal_query.hpp provides a mapping layer that converts parsed NDN Interests into TemporalQuery objects. Each parameter extracted from the Interest – such as date, timeframe, count, and filters – is used to populate the corresponding fields of a TemporalQuery. This enables direct translation between Interest name structures and query semantics within the SeEDS framework. The implementation is available in:

- parsed_interest_to_temporal_query.hpp

4.2 General Data Flow

The HTTP Server acts as an intermediary between the IP-based Internet and the NDN network. Upon receiving a request, it distinguishes between POST and GET methods and parses the corresponding parameters to determine the appropriate NDN operation to execute.

Maintaining this functionality in a scalable and extensible way is essential. A naïve implementation based solely on conditional statements (if-chains) would quickly become unmanageable, as each new parameter or request type would require additional hard-coded logic to identify and process it correctly.

To address this issue, a combined C++ Macro and Template mechanism was developed, the QUERY_PARAMS_LIST as mentioned in Section 4.1. This approach enables new parameters to be integrated seamlessly – by simply declaring them – while automatically generating the necessary parsing and request-handling logic. The implementation details of this scheme are beyond the scope of this document but are available in the SeEDS repository in the following files:

- snds_message.hpp
- query_params.hpp

After parsing an HTTP Request and inspecting its method, the server delegates handling to one of two internal components: the Digital Twin for POST requests, or the Worker for GET requests.

- For HTTP POST requests, the client introduces new content into the system, as discussed in Section 4.3. In response, the Digital Twin constructs the appropriate command–request and forwards it to both the Producer and Consumer components (see Section 4.1). This mechanism ensures that the submitted data is published under the correct name within the network (Publisher) and a new Interest is generated (Consumer). The creation of a new Interest is required in two cases: (i) to notify subscribers that new data has become available, and (ii) during a `BY-TYPE-AND-ID` advertisement, where it is used to identify the SeEDS Node responsible for a particular `@type` to store the new `@id` in the registry of the Node. Both cases are further described in Section 4.3.

- For `HTTP GET` requests, the client is requesting data from the NDN network. In this workflow, the Worker forwards the request to the Consumer, which is responsible for constructing and expressing the appropriate Interests into the NDN network. The Interests are created using `snds_interest` (see Section 4.1). The party receiving the Interest can easily parse the interest components due to the format `prefix/name/command:param1&param2&`. The subsequent sections explain in more detail how these Interests are formed and processed.

The Consumer and Producer, after completing a request, return an ACK or a NACK or even the requested data back to the Worker. The Worker forwards the response to the HTTP Server, which subsequently returns it to the client. For brevity, in the remainder of this document we will simply state that the Worker forwards the response to the client. The `snds_message` library (see Section 4.1) defines several distinct ACK and NACK types, each representing a specific success condition or error case. For example there are ACK messages for successfully creating a new subscription or when a new ID has been successfully advertised.

4.3 Announcing Content

4.3.1 Data Flow

When a node intends to announce content within the NDN network, it does so by advertising specific names. Consumers can then retrieve the corresponding content by expressing Interests for these names. Each node in the network possesses a unique identifier (e.g., Node 1 on the left and Node 2 on the right in Figure 5). As illustrated in messages `(0)` and `(1)`, each node initially announces the base `SeEDS` name concatenated with its unique identifier, forming the name `SeEDS_ID`. In addition, each node announces a `scope` under a name of the form `ID_scopeID`. If a node is responsible for a particular registry or entity type – such as the `CAR_registry` in this example, shown in messages `(2)` and `(3)` – it also advertises the corresponding `TYPE` and `TYPE_REGISTRY` names, enabling it to respond to Interests related to that registry.

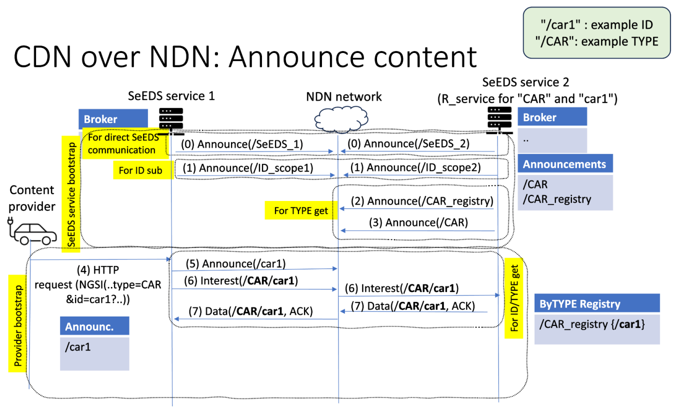


Figure 5: Announcing content inside the NDN network.

Following the initial announcements, assume that a new client intends to provide an entity with `@id = car1` of type `@type = CAR`. When this new `@id` arrives, the receiving SeEDS Service first announces it locally, as illustrated in message `(5)`; this makes the item available by ID. It must then express an Interest, as shown in message `(6)`, for the `@type = CAR` registry so that the node responsible for this type can store the new entry in the registry; this makes the item available by TYPE. The responsible node replies with an `ACK` message, confirming that `@id = car1` has been successfully registered under `@type = CAR`. If the client's HTTP request does not explicitly specify a `@type`, the system omits messages `(6)` and `(7)` and only announces the `@id = car1` within the network.

4.3.2 Implementation Details

When the HTTP Server receives an HTTP `POST` request, it means that a content provider wants to provide content into the NDN network. After receiving the request, the HTTP Server checks whether it has the parameters `@type`, `@id` and `date`. If it contains a `@type` parameter, it means that the `@id` must also be announced through the `@type=TYPE` registry. If it does not contain it, the system ensures the just `@id=ID` is published in the NDN network. If it contains the `@id=ID` and the `date` parameter, then the system must perform a Temporal Query advertisement. We will call the first case , the second case 2 and the third case 3 respectively; we explain how each case is handled below.

1. As described in Section 4.2, incoming messages from HTTP `POST`s are forwarded to the Digital Twin component. This component uses the `snds_message` API (see Section 4.1) to construct the appropriate intra-SeEDS – in this case, a `BY-ID-AND-TYPE-ADVERTISEMENT` – message.

The message is forwarded to the local Consumer to be further processed into an Interest. When the appropriate Producer receives this advertisement Interest, as the SeEDS component (see Section 4.1) responsible for receiving and parsing Interests, it inserts the provided `@id` into the registry associated with its configured `@type`.

Recall that a SeEDS Node declares any `@type` it manages during initialization. For example, let's say a SeEDS Node is responsible for `@type = BUS`, during initialization. If it receives an advertisement request for `@type = BUS` and `@id = bus1`, it must first express an Interest for the corresponding name. This is done through the Consumer component. This Interest is expressed as:

```
SNDS/BUS/byIDAndTypeAdvertisement:bus1&BUS&
```

The Interest above will be eventually processed by the appropriate Node's Producer and it will be inserted into its registry.

After successfully inserting the `@id` in the appropriate `@type` registry, the Producer sends back an Data message as a response. These responses are the ACKs and NACKs, mentioned in Section 4.2. After the initial SeEDS Node that received the HTTP `POST` receives the response, it forwards it back to the Worker and in turn to the Client.

The system also ensures that the `@id's` data are inserted into the persistent storage (database) of the Node's local Broker (see Section 4.1). Since the `@type` is defined, the `@id` is inserted into the `Collection` with the same name as the `@type`. For example if `@type=BUS`, the system ensures that the new `@id=bus1`, along with its data, will get stored inside the `BUS Collection`. If such a `Collection` doesn't exist in the storage, it will automatically create it. The system also assigns an automatic timestamp to the record by using the current time.

Additionally, along with creating the `BY-ID-AND-TYPE-ADVERTISEMENT` intra-SeEDS message, after waiting for a couple of seconds, the system also creates a `BY-ID-AND-TYPE-WITH-SCOPE-ADVERTISEMENT` with the goal of notifying subscribers to a `@type` that new content is available in the system. Recall that all SeEDS Nodes in the network have already announced their `SNDS_ID_scope_(SNDS-NODE-ID)`; we will call this the `local_scope`. The message is forwarded to the Consumer component and it creates the following Interest:

```
SNDS/@type/scopeNotification:@id&@type&local_scope
```

In this context of a `BY-ID-AND-TYPE-ADVERTISEMENT`, the `@id` field is deliberately left empty. This maintains the required placeholder format, ensuring that the SeEDS Node can interpret and parse the Interest. The `local_scope` of the notifying Node is also appended, just for debugging purposes.

The Interest is received by the appropriate SeEDS Node – that is, the node that previously announced the corresponding `@type` to the network. Upon receiving the Interest, the node's Producer parses it, extracting all relevant parameters. Each node contains an in-memory registry of subscribers that have subscribed to the particular `@type`. For each subscriber that has a `subscriber_scope`, like the `local_scope`, the Producer forwards a `SUBSCRIPTION-NOTIFICATION` intra-SeEDS message to the Consumer. The Consumer creates the following Interest:

```
SNDS/subscriber_scope/subscriptionNotification:
@id&@type&local_scope
```

Again in this context of a `BY-ID-AND-TYPE-ADVERTISEMENT`, the `@id` field is deliberately left empty. This maintains the required placeholder format, ensuring that the SeEDS Node can interpret and parse the Interest. The Node receiving this Interest will proceed to perform a *GET by TYPE* to get the new data for the `@type`. The `local_scope` of the subscription holder Node is also appended, just for debugging purposes.

2. Again, as described in Section 4.2, incoming messages from HTTP `POST`s are forwarded to the Digital Twin component. This component uses the `snds_message` API (see Section 4.1) to construct the appropriate ntra-SeEDS message—in this case, a `BY-ID-ADVERTISEMENT` message. Since the message in this case contains only an `@id`, there is no need to discover the SeEDS Node responsible for its type, as in the previous case. Instead, the message is forwarded directly to the Producer component. Upon receiving it, the Producer simply publishes the corresponding name into the NDN network. For example, if `@id = car1` is received, the Producer will announce that it can provide content under the following name:

```
SNDS/car1
```

The system also ensures that the data associated with the `@id` are inserted into the persistent storage (database) of the node's local Broker (see Section 4.1). Because the `@type` is not defined, the entry is stored in the `GENERIC` collection.[2] For example, when a new `@id = car1` is introduced, the system ensures that it is stored – together with its data – in the `GENERIC` collection. If the `GENERIC` collection does not already exist in the storage backend, it is created automatically. The system also assigns an automatic timestamp to the record by using the current time.

Additionally, along with creating the `BY-ID-ADVERTISEMENT` intra-SeEDS message, after waiting for a couple of seconds, the system also creates a `BY-ID-WITH-SCOPE-ADVERTISEMENT` with the goal of notifying subscribers to an `@id` that new content is available in the system. Subscriptions by `@id` are assigned to Nodes based on the hashing function discussed in Section 4.6.2. To find the Node that is responsible for the subscription, the hash function is used to find the Node's `scope`. The `snds_message` is then forwarded to the Consumer component and it creates the following Interest:

```
SNDS/@scope/scopeNotification:@id&@type&local_scope
```

In this context of a `BY-ID-ADVERTISEMENT`, the `@type` field is deliberately left empty. This maintains the required placeholder format, ensuring that the SeEDS Node can interpret and parse the Interest. The `local_scope` of the notifying Node is also appended, just for debugging purposes.

The Interest is received by the appropriate SeEDS Node – that is, the node that previously announced the `scope` to the network. Upon receiving the Interest, the node's Producer parses it and extracts all relevant parameters. Each node contains an in-memory registry of subscribers that have subscribed to the particular `@id`. For each subscriber that has a `subscriber_scope`, like the `local_scope`, the Producer forwards a `SUBSCRIPTION-NOTIFICATION` intra-SeEDS message to the Consumer. The Consumer creates the following Interest:

```
SNDS/subscriber_scope/subscriptionNotification:
@id&@type&local_scope
```

Again in this context of a `BY-ID-ADVERTISEMENT`, the `@type` field is deliberately left empty. This maintains the required placeholder format, ensuring that the SeEDS Node can interpret and parse the Interest. The Node receiving this Interest will proceed to perform a *GET by ID* to get the new data for the `@id`. The `local_scope` of the subscription holder Node is also appended, just for debugging purposes.

[2]The name of this collection is purely a convention and could be anything.

3. Finally, again as described in Section 4.2, incoming messages from HTTP POSTs are forwarded to the Digital Twin component. This component uses the snds_message API (see Section 4.1) to construct the appropriate intra-SeEDS – in this case, a TEMPORAL-QUERY-ADVERTISEMENT – message. The message is forwarded to the local Producer to be announced directly into the network. The name looks like the following:

   ```
   SNDS/@type/@id/date=value
   ```

   If the @type above is empty, it is omitted. The date cannot be omitted, since it is a temporal query. For example, if the @type is missing from the Interest it becomes:

   ```
   SNDS/@id/date=value
   ```

   Using the above scheme, users can retrieve versioned @ids, by expressing Interests and including the date=value parameter.

   The system also ensures that the data associated with the @id are inserted into the persistent storage (database) of the node's local Broker (see Section 4.1). If the @type is not defined, the entry is stored in the GENERIC Collection (see Section 2). If the @type is defined, the @id is inserted into the collection named after that @type (see Section 1). The date=value parameter is stored as a timestamp within the record, enabling timestamp-based queries.

4.4 GET by ID

4.4.1 Data Flow

After an @id is published into the network, users can retrieve its content item. Users create an HTTP GET and specify the @id that they want to retrieve, as shown in message (0) in figures 6 and 7. The SeEDS Service processes the message internally and expresses an Interest involving the @id=car1 inside the name, as shown in message (1) in figures 6 and 7. Note that messages (0) and (1) are identical in both the *Direct* and *Indirect* modes of the *GET by ID* operation; in Direct mode, the actual content items are returned, while in Indirect mode, a metafile containing the IDs of the content items is returned. The choice of modes occurs dynamically,depending on the data received by ProcessQueries(). Moreover, in both Direct and Indirect modes, the processing of filters occurs before the final data is returned to the user via the HTTP interface. Only the fields explicitly specified in the filter conditions are retained in the output.

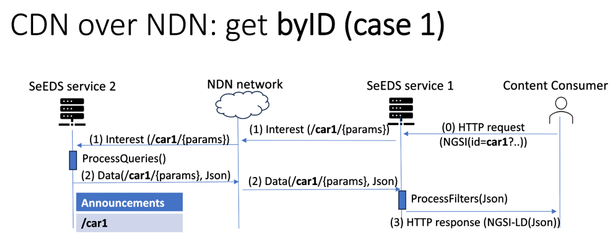

CDN over NDN: get byID (case 1)
SeEDS service 2
NDN network
SeEDS service 1
Content Consumer
(0) HTTP request (NGSI(id=car1?..))
(1) Interest (/car1/{params})
(1) Interest (/car1/{params})
ProcessQueries()
(2) Data(/car1/{params}, Json)
(2) Data(/car1/{params}, Json)
ProcessFilters(Json)
(3) HTTP response (NGSI-LD(Json))
Announcements
/car1

Figure 6: GET by ID direct mode.

4.4.2 Direct Mode

Direct mode occurs when the data processed by `ProcessQueries()` is classified as a *small payload*. In this case, the SeEDS Service responsible for the corresponding `@id` responds *directly* with the JSON payload, as illustrated in message `(2)`. The SeEDS Service that initially received the HTTP `GET` then forwards this data back to the client as the final response.

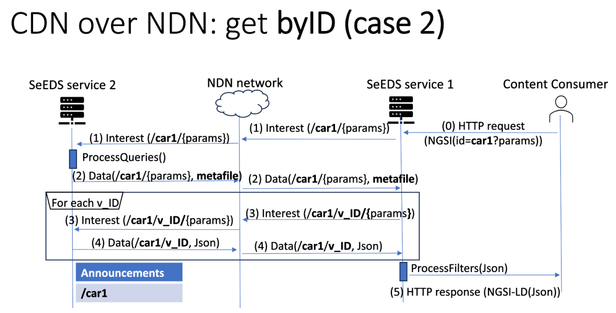

CDN over NDN: get byID (case 2)
SeEDS service 2
NDN network
SeEDS service 1
Content Consumer
(0) HTTP request (NGSI(id=car1?params))
(1) Interest (/car1/{params})
(1) Interest (/car1/{params})
ProcessQueries()
(2) Data(/car1/{params}, metafile)
(2) Data(/car1/{params}, metafile)
For each v_ID
(3) Interest (/car1/v_ID/{params})
(3) Interest (/car1/v_ID/{params})
(4) Data(/car1/v_ID, Json)
(4) Data(/car1/v_ID, Json)
ProcessFilters(Json)
(5) HTTP response (NGSI-LD(Json))
Announcements
/car1

Figure 7: GET by ID indirect mode.

4.4.3 Indirect Mode

Indirect mode occurs when the data processed by `ProcessQueries()` is classified as a *large payload*. In this case, the SeEDS Service responsible for the corresponding `@id` does not transmit the full JSON directly. Instead, it responds with a `metafile`, as illustrated in message `(2)`. The `metafile` contains a list of `@v_ids`, which represent the individual `@ids` along with their respective versions. Upon receiving the `metafile`, the SeEDS Service that originally handled the HTTP `GET` proceeds to express Interests for the specific `@id` and each associated version. This effectively re-initiates the retrieval process, in the same manner as a Direct mode request, just for a specific version. The version can be represented in various forms, such as a timestamp, a numerical identifier, or any other scheme that uniquely distinguishes different instances of the same entity.

4.4.4 Implementation Details

GET by ID is an HTTP `GET` request in which the client specifies an `@id` to retrieve. As described in Section 4.2, all HTTP `GET` requests are forwarded to the Worker component (see Section 4.1). Upon receiving the request, the Worker inserts the `@id` into a `pending_ids` container to prevent issuing duplicate requests for the same identifier. It then uses the `snds_message` API (see Section 4.1) to construct the appropriate intra-SeEDS message – in this case, a `GET-BY-ID-REQUEST`. This message is forwarded to the Consumer, which generates the corresponding NDN Interest using the `snds_interest` library. The Interest is encoded via the `snds_temporal_query` library as a Temporal Query and takes the form:

```
SNDS/@id/temporalQuery:@id
```

The Interest is received by the appropriate SeEDS Node, that is, the node that previously announced the corresponding `@id` to the network. After parsing the Interest, the node's Producer issues a request to the local Broker (see Section 4.1). By convention, the Broker is configured to retrieve the most recent entry associated with the specified `@id` in the local database. For example, given the following available timestamps:

- 1984
- 2020
- 2025

the Broker will return the entry for 2025. The retrieved record – along with any associated payload – is sent back to the Producer, which then returns the data to the Consumer that originated the request. If an error occurs it simply responds back with a NACK.

Upon receiving the Data packet, the Consumer must determine whether the response should be handled in Direct or Indirect Mode 4.4.2. At present, this decision is made by inspecting the number of returned entries: if more than one entry is present, the query is considered to fall under Indirect

Mode. In the case of a simple GET-by-ID request – i.e., a non-temporal query – the response always contains exactly one entry, and therefore it is processed in Direct Mode. In this mode, the Consumer simply returns the retrieved entry and its payload to the Worker, which then forwards the result back to the Client. In the case of a NACK, the Consumer follows the same handling path as it would for a valid entry, but without needing to parse or inspect the number of returned results.

4.5 GET by TYPE

4.5.1 Data Flow

Users can also request data associated with a specific @type. As illustrated in message (0) in Figure 8, the content consumer specifies the desired @type in the request sent to the SeEDS Service. The SeEDS Service that receives the HTTP Request proceeds to express an Interest in the NDN network, as shown in message (1). The SeEDS Service responsible for the corresponding registry then receives this Interest and responds with the list of @ids registered under that @type, as shown in message (2). After obtaining the list of @ids, the SeEDS Service that originally handled the HTTP request performs a separate GET by ID operation for each @id to retrieve the associated data.

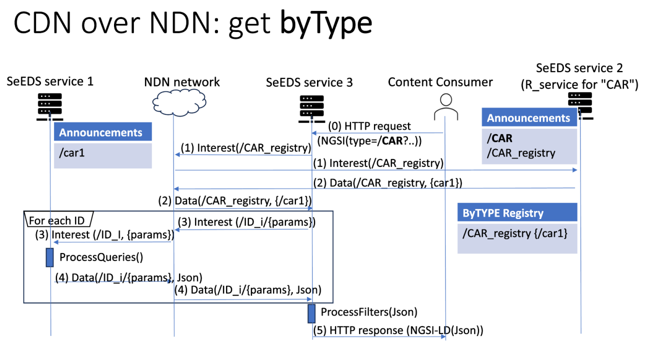


Figure 8: GET by TYPE.

4.5.2 Implementation Details

GET by TYPE is an HTTP GET request in which the client specifies an @type to retrieve. As described in Section 4.2, all HTTP GET requests are forwarded to the Worker component (see Section 4.1). Upon receiving the request, the Worker inserts the @type into a pending_get_by_type_requests container to prevent issuing duplicate requests for the same type. It then uses the `snds_message` API (see Section 4.1) to construct the appropriate intra-SeEDS message – in this case, a `GET-BY-TYPE-REQUEST`. This message is forwarded to the Consumer, which generates the corresponding NDN Interest using the `snds_interest` library. The Interest takes the form:

```
SNDS/@type/getByType:@type
```

The Interest is received by the appropriate SeEDS Node – specifically, the node that previously announced the corresponding `@type` into the network. Recall that each such node maintains an in-memory registry for its `@type`, which is simply a container holding all associated `@ids`. After parsing the Interest, the node's Producer component (see Section 4.1) locates the relevant `@type` in the registry, serializes the stored `@ids` into the agreed-upon byte format, and returns this byte stream as the response to the Interest.

After receiving the Interest response the Consumer component (see Section 4.1) of the Node creates a new intra-SeEDS message, `REGISTRY-PAYLOAD`. This message is then forwarded to the Worker component, which parses the payload (it can do that since the byte format is known) and extracts the `@ids` from it. For each `@id` it stores it in a `pending_ids_to_type_map` and performs a *GET by ID* (see Section 4.4). The only difference is that the Interest looks like this in order to be able to retrieve the data from the corresponding `@type`:

```
SNDS/@type/temporalQuery:@id&@type
```

The Producer component of the SeEDS node will retrieve the `@ids` from its Broker component (see Section 4.1), using the `Collection` with the same name as the `@type` as mentioned in Section 4.3.2, and return them back as a response to the Consumer component. The Consumer aggregates the results and forwards them to the Worker, which ultimately delivers the response back to the Client.

4.6 SUBSCRIPTION by ID

4.6.1 Data Flow

Consumers can also subscribe to a specific `@id`. Once content associated with that `@id` becomes available, the system automatically notifies the subscribed consumers and delivers the corresponding data. Consumers can thus subscribe to `@ids` even before the data has been published.

A consumer that wishes to subscribe to an `@id` must specify the target identifier, as illustrated in message `(0)` in Figure 9. Upon receiving the HTTP `request`, the SeEDS Service proceeds to express an Interest containing both the responsible `ID_SCOPEID` for the given `@id` – determined via a hashing function – and its own `ID_SCOPEID`, as shown in message `(1)`. When the responsible ID_SCOPEID receives this Interest, it records the sender's ID_SCOPE as a subscriber for the specified @id and responds with an ACK, as depicted in message (2).

When new data for the subscribed @id is announced, the system checks for any existing subscribers and notifies them through their respective scopes that new content has become available, as shown in message (3). The notified SeEDS Service then initiates a GET by ID operation to retrieve the newly published data.

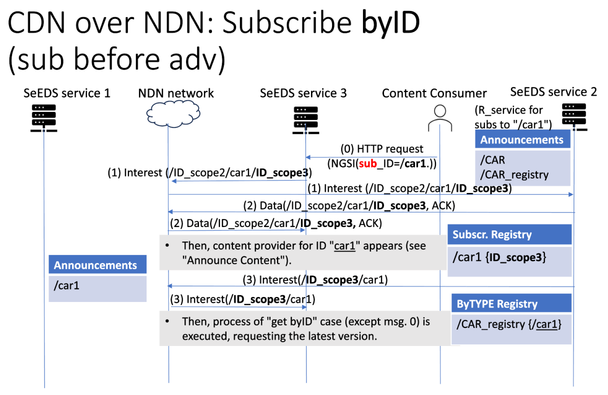

CDN over NDN: Subscribe byID
(sub before adv)
SeEDS service 1
NDN network
SeEDS service 3
Content Consumer
SeEDS service 2
(R_service for subs to "/car1")
Announcements
/CAR
/CAR_registry
(0) HTTP request (NGSI(sub_ID=/car1.))
(1) Interest (/ID_scope2/car1/ID_scope3)
(1) Interest (/ID_scope2/car1/ID_scope3)
(2) Data(/ID_scope2/car1/ID_scope3, ACK)
(2) Data(/ID_scope2/car1/ID_scope3, ACK)
Subscr. Registry
/car1 {ID_scope3}
Then, content provider for ID "car1" appears (see "Announce Content").
Announcements
/car1
(3) Interest(/ID_scope3/car1)
(3) Interest(/ID_scope3/car1)
ByTYPE Registry
/CAR_registry {/car1}
Then, process of "get byID" case (except msg. 0) is executed, requesting the latest version.

Figure 9: Subscribe by ID.

4.6.2 Implementation Details

SUBSCRIPTION by ID is an HTTP GET request in which the client specifies an @id to subscribe to and the isSub=true and suburl=MYURL parameters are included. As described in Section 4.2, all HTTP GET requests are forwarded to the Worker component (see Section 4.1). The HTTP Server component sends a intra-SeEDS message BY-ID-SUBSCRIPTION to the Worker, using the snds_message API (see Section 4.1). Upon receiving the request, the Worker inserts the @id into a subscribed_by_id_users to prevent duplicate subscriptions. To determine which SeEDS Node is responsible for a given @id, the system applies a lightweight deterministic hashing scheme. Specifically, the @id is hashed, and the resulting value is taken modulo the total number of SeEDS Nodes in the network, which produces an integer. All SeEDS Nodes in the network have already announced their scope, in the form of SNDS_ID_scope_(SNDS-NODE-ID). The integer produced gets appended to the SNDS_ID_scope_ and the appropriate node can be notified; let's call this the hashed_scope. There's also the local SeEDS Node's SNDS_ID_SCOPE_(SNDS-NODE-ID), which is going to be used to identify the node as a subscriber, let's call this the local_scope. Then the Worker creates a new intra-SeEDS message BY-ID-SUBSCRIPTION to notify the Consumer. The Consumer then proceeds to create the following Interest:

```
SNDS/hashed_scope/newIdSubscription:@id&local_scope
```

The Interest is forwarded to the appropriate SeEDS Node – specifically, the node that previously announced the corresponding hashed_scope into the network. Upon receiving the Interest, the node's Producer parses it, extracts all relevant parameters, and more importantly, reads the subscriber's local_scope. Each node maintains an in-memory container, which is called subscribed_nodes_for_id, tracking all subscribers that have subscribed to a given @id. The extracted local_scope (essentially, the subscriber) is inserted into this container. After storing the local_scope into the container the Producer sends an ACK back to the SeEDS Node that created the subscription request, whereas in case of an error it sends a NACK.

As a result, when the @id is later announced – along with its associated data – the node holding this subscription information can immediately notify the subscriber that registered interest in that @id.

4.7 SUBSCRIPTION by TYPE

4.7.1 Data Flow

Consumers can also subscribe to a specific @type. In this case (unlike with SUBSCRIBE by ID), the subscription is linked to an *already announced* registry scope, so there is no need to compute a hash for the @type. When the registry receives new data entries for that @type, all subscribed consumers are notified that an update has occurred.

As shown in message (0) in Figure 10, the consumer specifies the target @type in the HTTP Request sent to the SeEDS Service. Upon receiving the HTTP request, the SeEDS Service expresses an Interest toward the corresponding registry scope associated with that @type, as illustrated in message (1). The registry records the SeEDS Service that issued the Interest as a subscriber (the diagram doesn't use ID_SCOPE but SeEDS_ID, which is equivalent) and replies with an ACK, confirming that the subscription has been successfully registered, as shown in message (2).

When new data is added to the registry for the subscribed @type, the system notifies all registered subscribers, as depicted in message (3). Upon receiving the notification, the SeEDS Service proceeds to perform a GET by TYPE operation to retrieve the updated content.

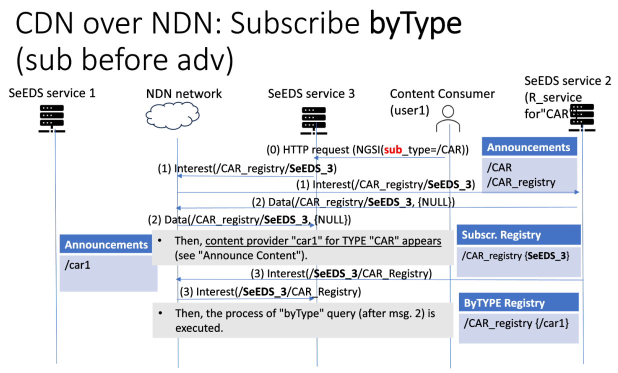

CDN over NDN: Subscribe byType (sub before adv)
SeEDS service 1
NDN network
SeEDS service 3
Content Consumer (user1)
SeEDS service 2 (R_service for"CAR")
(0) HTTP request (NGSI(sub_type=/CAR))
Announcements
/CAR
/CAR_registry
(1) Interest(/CAR_registry/SeEDS_3)
(1) Interest(/CAR_registry/SeEDS_3)
(2) Data(/CAR_registry/SeEDS_3, {NULL})
(2) Data(/CAR_registry/SeEDS_3, {NULL})
Announcements
/car1
Then, content provider "car1" for TYPE "CAR" appears (see "Announce Content").
Subscr. Registry
/CAR_registry {SeEDS_3}
(3) Interest(/SeEDS_3/CAR_Registry)
(3) Interest(/SeEDS_3/CAR_Registry)
ByTYPE Registry
/CAR_registry {/car1}
Then, the process of "byType" query (after msg. 2) is executed.

Figure 10: Subscribe by type.

4.7.2 Implementation Details

SUBSCRIPTION by TYPE is an HTTP `GET` request in which the client specifies a `@type` to subscribe to and the `isSub=true` and `suburl=MYURL` parameters are included. As described in Section 4.2, all HTTP `GET` requests are forwarded to the Worker component (see Section 4.1). The HTTP Server component sends a intra-SeEDS message `BY-TYPE-SUBSCRIPTION` to the Worker, using the `snds_message` API (see Section 4.1). Upon receiving the request, the Worker inserts the `@type` into a `subscribed_by_type_users` container to prevent duplicate subscriptions. Since the `@type` is already an announced scope, there is no need to calculate the `hashed_scope` like the *SUBSCRIPTION by ID* (see Section 4.6) case. Recall that each SeEDS Node has already advertised a scope that looks like this: `SNDS_ID_SCOPE_(SNDS-NODE-ID)`, let's call this `local_scope`. The Worker just creates a new intra-SeEDS message `BY-TYPE-SUBSCRIPTION` to notify the Consumer and the consumer creates the following Interest:

```
SNDS/@type/newTypeSubscription:@type&local_scope
```

The Interest is forwarded to the appropriate SeEDS Node – specifically, the node that previously announced the corresponding `@type` into the network. Upon receiving the Interest, the node's Producer parses it, extracts all relevant parameters, including the subscriber's `local_scope`. Each node maintains an in-memory container, `subscribed_nodes_for_type`, which is used to track all subscribers that have subscribed to a given `@type`. The extracted `local_scope` (essentially the subscriber) is inserted into this container. After storing the `local_scope` into the container the Producer sends an ACK back to the SeEDS Node that created the subscription request, whereas in case of an error it sends a NACK.

As a result, when new data for `@type`, meaning a new `@id`, is inserted into the node's in-memory registry for that `@type`, the SeEDS Node can immediately trigger a notification to the subscriber, informing it that fresh data is available. The notified subscriber then proceeds to perform a *GET by TYPE* for the `@type`.

4.8 Temporal Query

Temporal Queries are HTTP `GET` requests in which the client specifies an `@id` or a `@type`, a `date`, a `timeframe`, `count` and optionally `filter` parameters, with the following meaning:

- `@id` or `@type`: The identifier or type from which the temporal records will be retrieved.
- `date`: The reference timestamp from which the data should be retrieved.
- `timeframe`: Specifies whether to retrieve entries at the given timestamp (now), before it (`before`), or after it (`after`). So the possible values for the timeframe are {now, after, before}.
- `count`: This specifies the upper limit of entries to retrieve.
- `filter`: After the entries are returned to the client, any filters provided are applied to retain only the fields that match the specified filter criteria.

As described in Section 4.2, all HTTP `GET` requests are forwarded to the Worker component (see Section 4.1). The HTTP Server component sends a intra-SeEDS message – `TEMPORAL-QUERY-REQUEST` – to the Worker, using the `snds_message` API (see Section 4.1). Upon receiving the request, the Worker hashes the Temporal Query message and inserts the hash into a `pending_temporal_queries` to prevent duplicate queries. The Temporal Query is first serialized using the `snds_temporal_query` library (see Section 4.1). Serialization converts all query parameters into their corresponding JSON representation. The resulting JSON object is then forwarded to the Consumer as an `snds_message` of type 'TEMPORAL-QUERY-REQUEST'. At the Consumer side, the `snds_temporal_query` library is again used to interpret this message and transform the query into a key–value structure, resulting in an Interest of the following form:

```
SNDS/@type||@id/temporalQuery:@id=value&@type=value
&date=value&count=value&timeframe=value&filters=value
```

Note that `@type` and `@id` are always included in the query structure. However, one of them may not have a value, depending on whether the Temporal Query is executed by ID or by TYPE. The fields `count`, `timeframe`, and `filters` are optional; when not provided, they default to `1`, `now`, and an empty set, respectively.

The Interest is then received by the appropriate SeEDS Node – specifically, the node that previously announced the corresponding `@type` or `@id` into the network. Upon receiving the Interest, the node's Producer component parses it and extracts all relevant parameters. Each node maintains an active Broker (see Section 4.1), which contains versioned entries of `@ids`. Given a query, the Broker uses the supplied parameters to determine which versioned entries should be returned to the Producer. For example, for the following query parameters:

1. `date: 2025`
2. `count: 100`
3. `timeframe: after+before`

The Broker retrieves up to 100 versioned entries from after 2025 and up to 100 entries from before 2025.

The Producer component returns the selected entries to the Consumer component in an Interest response. In the case of an error a NACK is sent back to the Consumer component. Upon receiving this response, the Consumer component determines whether processing should proceed in Direct Mode (see Section 4.4.2) or Indirect Mode (see Section 4.4.3). If the response is handled in Direct Mode, it is forwarded directly to the Worker component, which subsequently delivers it to the Client. In contrast, Indirect Mode requires the Consumer to construct a separate Interest for each version/date contained in the Producer's response. These Interests follow the format:

```
SNDS/@id/version:@id=value&@type=value&date=value&

metafile=True&parentQueryHash=value&lastInterest=True|False
```

The parameters are interpreted as follows:

- `metafile` marks the Interest as one that targets a versioned entry,
- `parentQueryHash` encodes the identifier of the original Temporal Query that prompted these Interests,
- `lastInterest` indicates whether this Interest corresponds to the final versioned '@id' required for the query.

Then, the following operation is a *GET by ID* (see Section 4.4) for the `@id` provided in the Interests. The results are then sent back to the Consumer and are aggregated before being forwarded to the Worker and sent out back to the client.

4.9 Resilience

4.9.1 Data Flow

Figure 11 illustrates the message exchange that keeps a Secondary SeEDS service converged with a remote Primary over the NDN network. The Secondary (left lane) runs two lightweight polling loops in parallel: (i) a *state convergence* loop that tracks the state of the Primary's canonical store, and (ii) a *liveness* loop that verifies that the remote service is responsive. Both loops use standardized Interests and Data packets and carry only minimal material (hashes, patches-snapshots, nonces) to limit overhead.

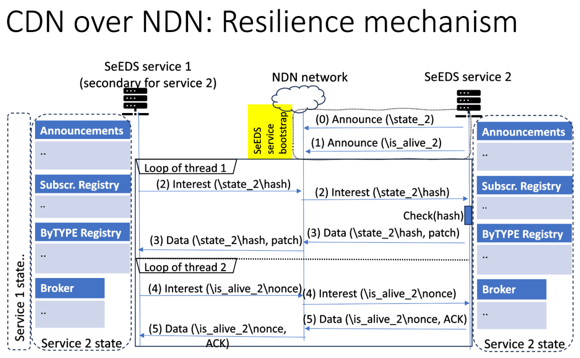

CDN over NDN: Resilience mechanism
SeEDS service 1
(secondary for service 2)
NDN network
SeEDS service 2
SeEDS service bootstrap
(0) Announce (\state_2)
(1) Announce (\is_alive_2)
Announcements
..
Subscr. Registry
..
ByTYPE Registry
..
Broker
..
Service 2 state
Service 1 state..
Loop of thread 1
(2) Interest (\state_2\hash)
(2) Interest (\state_2\hash)
Check(hash)
(3) Data (\state_2\hash, patch)
(3) Data (\state_2\hash, patch)
Loop of thread 2
(4) Interest (\is_alive_2\nonce)
(4) Interest (\is_alive_2\nonce)
(5) Data (\is_alive_2\nonce, ACK)
(5) Data (\is_alive_2\nonce, ACK)
Announcements
..
Subscr. Registry
..
ByTYPE Registry
..
Broker
..
Service 2 state

Figure 11: Resilience message exchange in SeEDS.

As shown in message `(0)`, the Primary has announced its state namespace `{state_2}` in the NDN network, so other Services in the network can express Interests for it. The currently manually configured Secondary initiates its state convergence loop by expressing an Interest for the corresponding state hash, as depicted in message `(2)`. The returned Data packet, shown in message `(3)`, contains the current hash of the store. The Secondary compares this hash with the locally stored version, and when a mismatch is detected, it sends another interest the namespace, specifying that there is a mismatch. The Primary responds with a patch encoded as an RFC 6902 JSON Patch and if the Secondary cannot apply it, it requests an RFC 6902 Snapshot.

In parallel with the state convergence loop, the Secondary executes a liveness verification loop to confirm that the Primary remains reachable. As illustrated in message (1), the Primary advertises its liveness namespace {is_alive_2}. The Secondary periodically sends Interests containing a fresh nonce, as shown in message (4). The Primary responds with a corresponding Data packet that echoes the nonce and includes an acknowledgment, as depicted in message (5).[3]

If the Secondary records consecutive failures in its liveness probes beyond a predefined threshold, it promotes itself to the Primary and terminates the polling loop. This promotion is internally communicated to the SeEDS services through an IPC (inter-process communication) channel. With this mechanism, SeEDS achieves distributed fault tolerance, maintaining service availability, despite network or node failures.

4.9.2 Implementation Details

The Resilience service (see Section 4.1) is executed by the Producer component (see Section 4.1). It is built on the `snds_inter_process_communication` library (see Section 4.1), which enables the Producer to exchange messages locally with the `snds_resilience` main executable (see Section 4.1). These messages include patches, snapshots, and `role` signals. They are based on Interest exchanged between the Primary Node and the Secondary Node, essentially the `snds_resilience` executable, allowing the Secondary to continuously track and mirror the state of the Primary and to recover the state of the Primary through the Secondary in case of a failure. More specifically:

- `Patches`: Changes are applied in the exact order in which the Primary's state was modified. For example, if a new subscription arrived first and a new `@id` was announced afterward, the Secondary will apply these changes in the same sequence, thereby preserving the correct version history.
- `Snapshots`: This mechanism serves as the fallback for patches. If a patch cannot be applied – for example, if the version history is missing, mismatched, or if the Secondary has just started – the system resets the state by loading the snapshot corresponding to a specific version. Suppose the Primary is at version 50, but the Secondary only has state up to version 30 due to a restart. If the Secondary cannot apply patches 31–50 because some of them are missing, a snapshot at version 50 is transmitted instead. The Secondary loads this snapshot and continues processing updates from version 51 onward.
- `Role`: This message indicates whether a terminal error has occurred on the Primary node, requiring the Secondary node to take over and

[3]The nonce is used to avoid getting previous (cached) Data messages as responses.

assume the role of Primary. When the `snds_resilience` executable receives a configurable number of consecutive `is_alive` error signals, it concludes that the remote node has failed. At that point, it issues a `primary` message to the Secondary, instructing it to promote itself to Primary. Conversely, when the original Primary node recovers, the resilience executable sends a `secondary` message, signaling the newly promoted node to revert back to its Secondary role.

The `snds_resilience` executable polls the Primary node at configurable intervals as mentioned above. The interval for the `Patches` and the `Snapshots` is shared, while the interval for the `Role` is independent. The executable uses these intervals to express periodic Interests to two NDN scopes:

- `is_alive`: The result of the Interest that is expressed to the `is_alive` scope, is transmitted through the `Role` message.
- `state`: The result of the Interest that is expressed to the `state` scope, is transmitted through the `meta` message. Based on the `meta` message, the system determines whether a `patch` or a `snapshot` needs to be generated. The `meta` mechanism retrieves the current version of the Primary node's state and compares it with the local version on the Secondary. This comparison allows the system to decide whether the Secondary can and should advance using patches or whether it must fall back to a snapshot.

5 Conclusion

This thesis described the design and implementation of the SeEDS system, an NDN-based Data Space that supports the NGSI-LD data model, temporal operations, distributed registries, and subscription mechanisms. The work builds on the earlier SNDS architecture and extends it with a more complete API, clearer internal abstractions, and a resilient state-replication mechanism.

A significant part of the effort focused on defining a modular internal structure for the service. Components such as the HTTP Server, Worker, Consumer, Producer, and Digital Twin were modularized and redesigned. Using the idea of the previous implementation, they are able to communicate through a unified intra-message format, reducing coupling and improving traceability of data flow. This structure made it possible to support the full range of NGSI-LD-style operations – GET by ID, GET by TYPE, subscriptions, and temporal queries – while ensuring that requests could be consistently translated into NDN Interests.

The implementation also introduced distributed subscription handling for both IDs and types. These mechanisms rely on deterministic scope selection and lightweight in-memory registries, allowing nodes in the network to notify subscribers whenever new relevant data is published. While simple, these mechanisms demonstrate how event-driven behaviour can be layered on top of NDN's naming and routing concepts.

Mechanisms such as metafile-based indirect retrieval and direct mode responses allow the system to be robust and efficient, responding to requests based on their workload and bandwidth usage.

Temporal querying and interaction with the Broker were implemented to provide access to historical records and versioned data. These additions allow the system to support use cases where past states of entities need to be accessed, not only their current representation. Although the Broker is an external dependency using MongoDB, the integration required defining stable serialization formats, Interest encodings, and proper integration of resilience mechanisms.

The resilience subsystem provides another important extension. Its purpose is to maintain a consistent state between a Primary and Secondary instance through periodic polling using patches, and snapshots. The design offers a functional basis for recovering from node failures without relying on central coordination.

Overall, the SeEDS implementation shows that a Data Space with NGSI-LD semantics can be realised on top of NDN using a combination of structured naming, modular components, and lightweight inter-process and inter-component communication. The system currently represents a well rounded working prototype rather than a finalized platform. Several areas would benefit from further work, including: refining error handling, evaluating the performance of distributed registries at scale, optimising request execution through the NDN caching mechanism, and conducting more extensive experiments on the global NDN testbed.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Professor George Xylomenos, for his guidance throughout my academic studies and for the continuous support that enabled me to pursue and complete this work. His direction has shaped much of my academic development.

I also wish to thank Dr. Yannis Thomas for his valuable insights into Named Data Networking paradigm and for setting up the NDN testbed used in this thesis in the appendix. His earlier work on the SNDS implementation provided an essential foundation for the system developed here.

Finally, I would like to thank Dr. Iakovos Pittaras for his assistance in setting up and running the experimental environment. His help was instrumental in carrying out the evaluations presented in this thesis in the appendix.

Appendices

A Experiments and KPIs

A.1 MiniNDN evalution

Mini-NDN is a lightweight emulation framework designed to facilitate experimentation with Named Data Networking (NDN) in a controlled, reproducible environment. It is built on top of Mininet, a popular network emulator that allows users to create virtual network topologies composed of multiple nodes, links, and switches within a single physical or virtual machine. Mini-NDN extends this capability by integrating the NDN Forwarding Daemon (NFD) and NDN tools into each emulated node, effectively transforming the Mininet network into a fully functional NDN deployment.

Each Mini-NDN node runs its own instance of NFD and can host NDN applications, such as producers and consumers, enabling realistic Interest/Data exchange over emulated network topologies. Mini-NDN supports custom configurations through topology and configuration files, making it straightforward to define arbitrary network graphs and specify per-node parameters such as faces, prefixes, and routing protocols.

Mini-NDN was employed to test and validate the functionality of the SeEDS prototype prior to its deployment in the global NDN testbed, ensuring correct operation before deployment to the actual testbed. Below, for brevity's sake, we are going to present the basic *GET by ID* and *GET by TYPE* requests. Everything else has been tested to ensure its correct operation and can be found on the official github repository.

A.1.1 Topology

The Mini-NDN emulation was deployed using a simple four-node topology defined in the configuration file shown in Listing 2. The nodes are labeled `a`, `b`, `c`, and `d`, each running its own instance of the NDN Forwarding Daemon (NFD) with the log level set to `DEBUG`.

The connectivity between nodes is symmetric, with bidirectional links defined for each pair. Nodes `a`, `b`, and `c` form a triangular core interconnected with a delay of `10ms` per link, ensuring low-latency communication suitable for multi-hop experiments. Node `d` connects only to node `a`, also with a `10ms` delay, representing an edge node that communicates with the core through a single gateway.

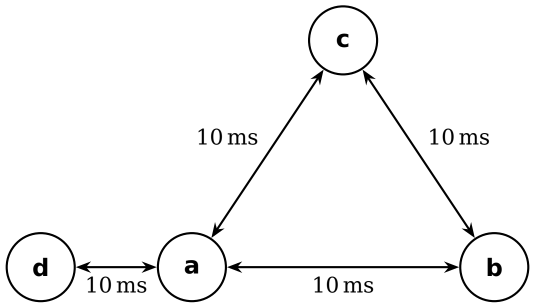


Figure 12: Mini-NDN topology: triangular core (a–b–c) with 10 ms links; edge node d connected to a.

Listing 2: Mini-NDN topology .conf file

```
[nodes]
a: _nfd-log-level=DEBUG
b: _nfd-log-level=DEBUG
c: _nfd-log-level=DEBUG
d: _nfd-log-level=DEBUG

[links]
a:b delay=10ms
b:a delay=10ms
b:c delay=10ms
c:b delay=10ms
c:a delay=10ms
a:c delay=10ms
d:a delay=10ms
a:d delay=10ms
```

A.1.2 GET by ID

We have already published the `@ids=car1,car2,car3` in the NDN network, through node A (the producer). Let's try to retrieve the item with `@id=car1` through node B:

Listing 3: Node's B log after performing a GET by ID for car1

```
[2025-09-25 00:33:40.583] [debug][Consumer]Extracted content from
    message: /SNDS/car1/getByID:car1
[2025-09-25 00:33:40.583] [debug][Consumer]Extracted content length
    from message: 23
[2025-09-25 00:33:40.583] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_REQUEST
[2025-09-25 00:33:40.583] [debug][Consumer]Query.type:
```

https://doi.org/10.26219/heal.aueb.9822

```
[2025-09-25 00:33:40.583] [debug][Consumer]Query.id: car1
[2025-09-25 00:33:40.583] [info][Consumer]Created name /SNDS/car1/
    temporalQuery:id%3Dcar1&type%3D&date%3D&count%3D&timeframe%3D&
    filters%3D
[2025-09-25 00:33:40.583] [info][Consumer]Sending actual Interest /
    SNDS/car1/temporalQuery%3Aid%3Dcar1%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime=6000
[2025-09-25 00:33:40.585] [info][Consumer]Received Data with name: /
    SNDS/car1/temporalQuery%3Aid%3Dcar1%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D in function onData
[2025-09-25 00:33:40.585] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car1",
      "id": "car1",
      "timestamp": {
        "$date": 1758749528574
      },
      "type": "GENERIC"
    }
  }
] in function onData
[2025-09-25 00:33:40.585] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: car1, Command: temporalQuery, Params: [id=car1, type
    =, date=, count=, timeframe=, filters=] in onData
[2025-09-25 00:33:40.585] [info][Consumer]Temporal Query Reponse...
```

Let's try to retrieve the `@id=car2` through node C:

Listing 4: Node's C log after performing a GET by ID for car2

```
[2025-09-25 00:42:10.775] [debug][Consumer]Extracted content from
    message: /SNDS/car2/getByID:car2
[2025-09-25 00:42:10.775] [debug][Consumer]Extracted content length
    from message: 23
[2025-09-25 00:42:10.775] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_REQUEST
[2025-09-25 00:42:10.775] [debug][Consumer]Query.type:
[2025-09-25 00:42:10.775] [debug][Consumer]Query.id: car2
[2025-09-25 00:42:10.775] [info][Consumer]Created name /SNDS/car2/
    temporalQuery:id%3Dcar2&type%3D&date%3D&count%3D&timeframe%3D&
    filters%3D
[2025-09-25 00:42:10.775] [info][Consumer]Sending actual Interest /
    SNDS/car2/temporalQuery%3Aid%3Dcar2%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime=6000
[2025-09-25 00:42:10.777] [info][Consumer]Received Data with name: /
    SNDS/car2/temporalQuery%3Aid%3Dcar2%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D in function onData
[2025-09-25 00:42:10.777] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car2",
      "id": "car2",
      "timestamp": {
        "$date": 1758749532578
      },
      "type": "GENERIC"
    }
  }
] in function onData
[2025-09-25 00:42:10.777] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: car2, Command: temporalQuery, Params: [id=car2, type
    =, date=, count=, timeframe=, filters=] in onData
```

Let’s try to retrieve the `@id=car3` through node D:

Listing 5: Node’s D log after performing a GET by ID for car3

```
[2025-09-25 00:38:19.399] [debug][Consumer]Extracted content from
    message: /SNDS/car3/getByID:car3
[2025-09-25 00:38:19.399] [debug][Consumer]Extracted content length
    from message: 23
[2025-09-25 00:38:19.399] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_REQUEST
[2025-09-25 00:38:19.399] [debug][Consumer]Query.type:
[2025-09-25 00:38:19.399] [debug][Consumer]Query.id: car3
[2025-09-25 00:38:19.399] [info][Consumer]Created name /SNDS/car3/
    temporalQuery:id%3Dcar3&type%3D&date%3D&count%3D&timeframe%3D&
    filters%3D
[2025-09-25 00:38:19.399] [info][Consumer]Sending actual Interest /
    SNDS/car3/temporalQuery%3Aid%3Dcar3%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime=6000
[2025-09-25 00:38:19.400] [info][Consumer]Received Data with name: /
    SNDS/car3/temporalQuery%3Aid%3Dcar3%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D in function onData
[2025-09-25 00:38:19.400] [info][Consumer]Data packet payload: [
 {
  "": {
   "_id": "car3",
   "id": "car3",
   "timestamp": {
    "$date": 1758749534580
   },
   "type": "GENERIC"
  }
 }
] in function onData
[2025-09-25 00:38:19.400] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: car3, Command: temporalQuery, Params: [id=car3, type
    =, date=, count=, timeframe=, filters=] in onData
```

https://doi.org/10.26219/heal.aueb.9822

Here are the logs of the Producer, receiving requests from the other Nodes that requested the @ids:

Listing 6: Producer Log File

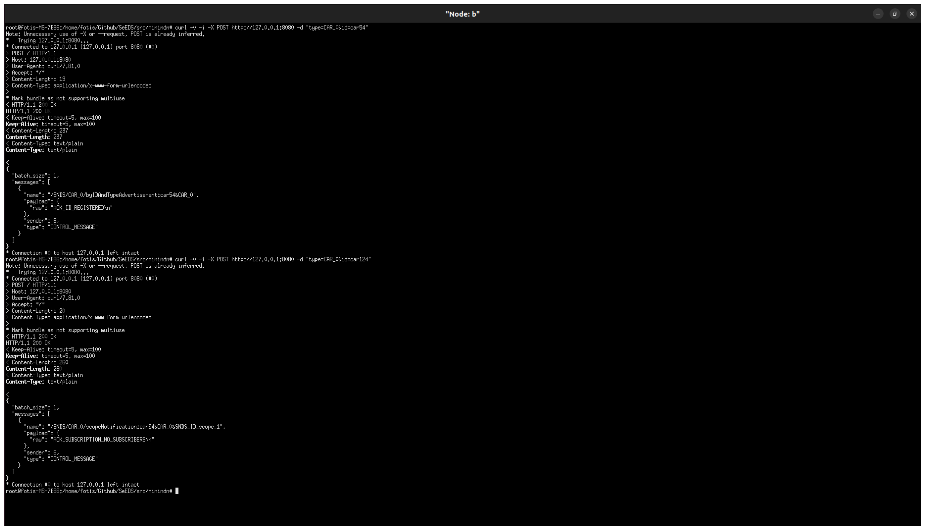
```
[2025-09-25 00:33:40.583] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/car1/temporalQuery%3Aid%3Dcar1
    %26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters%3D
[2025-09-25 00:33:40.583] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: car1, Command: temporalQuery, Params: [id=car1, type
    =, date=, count=, timeframe=, filters=]
[2025-09-25 00:33:40.583] [debug][Producer]Created interest response:
     /SNDS/car1/temporalQuery%3Aid%3Dcar1%26type%3D%26date%3D%26
    count%3D%26timeframe%3D%26filters%3D in:
    process_temporal_query_interest
[2025-09-25 00:33:40.583] [debug][Producer]TemporalQuery -> id: car1,
     date: , type: , timeframe: , count: , filters:
[2025-09-25 00:33:40.583] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car1&type=&count=&date=&timeframe=&filters=
[2025-09-25 00:33:40.585] [info][Producer]Successfully queried /
    temporal-query
[2025-09-25 00:33:40.585] [debug][Producer]Sending data to the NDN...
[2025-09-25 00:35:26.584] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/SNDS_ID_scope_0/
    scopeNotification%3Acar2%26%26SNDS_ID_scope_2
[2025-09-25 00:35:26.584] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: SNDS_ID_scope_0, Command: scopeNotification, Params:
     [car2, , SNDS_ID_scope_2]
[2025-09-25 00:35:26.584] [info][Producer][BY_ID] Subscription
    notification for ID name car2 and/or TYPE: .
[2025-09-25 00:35:26.584] [info][Producer]No subscribers found for
    ID: car2 and/or TYPE: .
[2025-09-25 00:35:26.584] [info][Producer]Responding to interest: /
    SNDS/SNDS_ID_scope_0/scopeNotification%3Acar2%26%26
    SNDS_ID_scope_2
[2025-09-25 00:35:26.584] [debug][Producer]Sending data to the NDN...
[2025-09-25 00:35:41.239] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/car3/temporalQuery%3Aid%3Dcar3
    %26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters%3D
[2025-09-25 00:35:41.239] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: car3, Command: temporalQuery, Params: [id=car3, type
    =, date=, count=, timeframe=, filters=]
[2025-09-25 00:35:41.239] [debug][Producer]Created interest response:
     /SNDS/car3/temporalQuery%3Aid%3Dcar3%26type%3D%26date%3D%26
    count%3D%26timeframe%3D%26filters%3D in:
    process_temporal_query_interest
[2025-09-25 00:35:41.239] [debug][Producer]TemporalQuery -> id: car3,
     date: , type: , timeframe: , count: , filters:
```

```
[2025-09-25 00:35:41.239] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car3&type=&count=&date=&timeframe=&filters=
[2025-09-25 00:35:41.241] [info][Producer]Successfully queried /
    temporal-query
[2025-09-25 00:35:41.241] [debug][Producer]Sending data to the NDN...
```

We can see that within the described Mini-NDN topology, node A acts as the producer responsible for serving data associated with specific *@ids*. Because all nodes are interconnected, any node in the network can retrieve content produced by node A. Nodes B and C communicate with node A via direct paths within the triangular core, enabling efficient Interest forwarding and Data return. Node D, which is connected only to node A, retrieves data through its single 10 ms link without possibly requiring any intermediate forwarding. This configuration ensures that every node in the network has a reachable route to the producer, allowing all consumers to successfully issue Interests and receive corresponding Data packets for a given `@id`, thus validating correct NDN forwarding and content retrieval in the SeEDS prototype.

A.1.3 GET by TYPE

We also publish the `@ids` to the `@type=CAR_0` registry announced by node A. Although in this setup the `@ids` originate from other nodes, they could equally be published directly through node A itself. This procedure ensures that the corresponding registry scopes (in this case, `/CAR_0`) are discoverable throughout the network.

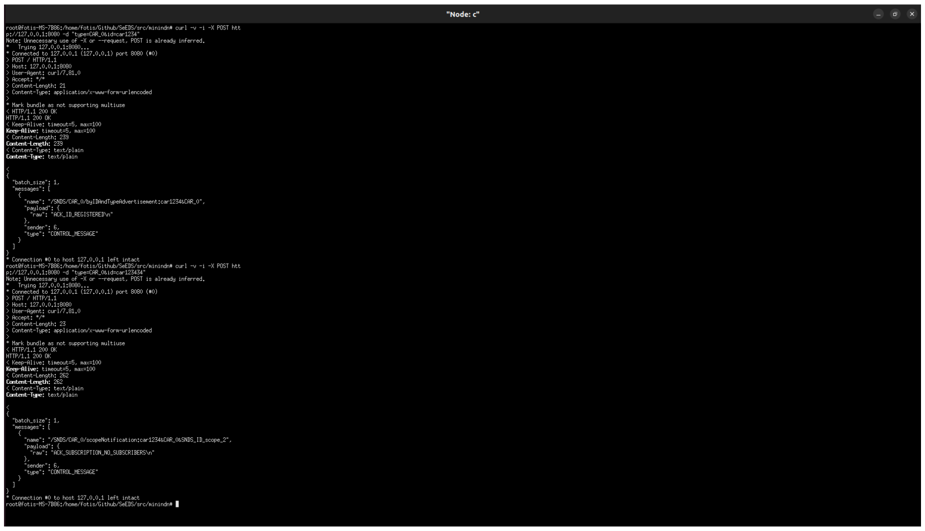


```
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# curl -v -i -X POST http://127.0.0.1:8080 -d "type=CAR_0&id=car54"
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Length: 19
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Keep-Alive: timeout=5, max=100
Keep-Alive: timeout=5, max=100
< Content-Length: 237
Content-Length: 237
< Content-Type: text/plain
Content-Type: text/plain

<
{
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_0/byIDAndTypeAdvertisement:car54&CAR_0",
      "payload": {
        "raw": "ACK_ID_REGISTERED\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
* Connection #0 to host 127.0.0.1 left intact
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# curl -v -i -X POST http://127.0.0.1:8080 -d "type=CAR_0&id=car124"
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Length: 20
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Keep-Alive: timeout=5, max=100
Keep-Alive: timeout=5, max=100
< Content-Length: 260
Content-Length: 260
< Content-Type: text/plain
Content-Type: text/plain

<
{
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_0/scopeNotification:car54&CAR_0&SNDS_ID_scope_1",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NO_SUBSCRIBERS\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
* Connection #0 to host 127.0.0.1 left intact
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# █
```

Figure 13: Posting the `@ids=car54,car124` to the `@type=CAR_0` registry of A.



```
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# curl -v -i -X POST htt
p://127.0.0.1:8080 -d "type=CAR_0&id=car1234"
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Length: 21
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Keep-Alive: timeout=5, max=100
Keep-Alive: timeout=5, max=100
< Content-Length: 239
Content-Length: 239
< Content-Type: text/plain
Content-Type: text/plain

<
{
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_0/byIDAndTypeAdvertisement:car1234&CAR_0",
      "payload": {
        "raw": "ACK_ID_REGISTERED\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
* Connection #0 to host 127.0.0.1 left intact
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# curl -v -i -X POST htt
p://127.0.0.1:8080 -d "type=CAR_0&id=car123434"
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> POST / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.81.0
> Accept: */*
> Content-Length: 23
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
HTTP/1.1 200 OK
< Keep-Alive: timeout=5, max=100
Keep-Alive: timeout=5, max=100
< Content-Length: 262
Content-Length: 262
< Content-Type: text/plain
Content-Type: text/plain

<
{
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_0/scopeNotification:car1234&CAR_0&SNDS_ID_scope_2",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NO_SUBSCRIBERS\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
* Connection #0 to host 127.0.0.1 left intact
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# █
```

Figure 14: Posting the `@ids=car123434,car1234` to the `@type=CAR_0` registry of A.

Let's try to get A's registry through D:

Listing 7: D after performing a GET by TYPE for CAR_0

https://doi.org/10.26219/heal.aueb.9822

```
[2025-09-25 00:16:25.868] [info][Consumer]Sending actual Interest /
    SNDS/CAR_0_registry/getByType%3ACAR_0?CanBePrefix&Lifetime=6000
[2025-09-25 00:16:25.869] [info][Consumer]Received Data with name: /
    SNDS/CAR_0_registry/getByType%3ACAR_0 in function onData
[2025-09-25 00:16:25.869] [info][Consumer]Data packet payload: R
    car123434|car1234|car124|car54 in function onData
[2025-09-25 00:16:25.869] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: CAR_0_registry, Command: getByType, Params: [CAR_0]
    in onData
[2025-09-25 00:16:25.869] [info][Consumer]Received Registry File
[2025-09-25 00:16:25.869] [info][Consumer]In Function: onData sending
     name CAR_0 to worker R car123434|car1234|car124|car54
[2025-09-25 00:16:25.869] [info][Worker]Received content: CAR_0
[2025-09-25 00:16:25.869] [info][Consumer]Reading packet...
[2025-09-25 00:16:25.869] [info][Worker]Received payload: R car123434
    |car1234|car124|car54
[2025-09-25 00:16:25.869] [info][Worker]Received type:
    REGISTRY_PAYLOAD
[2025-09-25 00:16:25.869] [debug][Worker]Consumer intra message...
[2025-09-25 00:16:25.869] [debug][Worker]Number of IDs in payload: 4
[2025-09-25 00:16:25.869] [debug][Worker]Reading ID #1 with size 9
[2025-09-25 00:16:25.869] [info][Worker]Parsed ID #1: car123434
[2025-09-25 00:16:25.869] [debug][Worker]Reading ID #2 with size 7
[2025-09-25 00:16:25.869] [info][Worker]Parsed ID #2: car1234
[2025-09-25 00:16:25.869] [debug][Worker]Reading ID #3 with size 6
[2025-09-25 00:16:25.869] [info][Worker]Parsed ID #3: car124
[2025-09-25 00:16:25.869] [debug][Worker]Reading ID #4 with size 5
[2025-09-25 00:16:25.869] [info][Worker]Parsed ID #4: car54
[2025-09-25 00:16:25.869] [info][Worker]Size of ids in registry file:
     4
[2025-09-25 00:16:25.869] [debug][Worker]Processing registry id:
    car123434
[2025-09-25 00:16:25.869] [info][Worker]Pushing request to the
    consumer for id car123434
[2025-09-25 00:16:25.869] [debug][Worker]Processing registry id:
    car1234
[2025-09-25 00:16:25.869] [info][Worker]Pushing request to the
    consumer for id car1234
[2025-09-25 00:16:25.869] [debug][Worker]Processing registry id:
    car124
[2025-09-25 00:16:25.869] [info][Worker]Pushing request to the
    consumer for id car124
[2025-09-25 00:16:25.869] [debug][Worker]Processing registry id:
    car54
[2025-09-25 00:16:25.869] [info][Worker]Pushing request to the
    consumer for id car54
[2025-09-25 00:16:25.869] [debug][Consumer]Extracted content from
```

```
    message: /SNDS/CAR_0/getByIDForType:car123434&CAR_0
[2025-09-25 00:16:25.869] [debug][Consumer]Extracted content length
    from message: 42
[2025-09-25 00:16:25.869] [info][Worker]Reading packet...
```

Recall that each GET by TYPE request, becomes a GET by ID for each `@id` of the registry. D has successfully requested each `@id` of the registry:

Listing 8: D's log after receiving the registry payload, which are the @ids, and performing a GET by ID for each

```
[2025-09-25 00:16:25.869] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
[2025-09-25 00:16:25.869] [debug][Consumer]Query.type: CAR_0
[2025-09-25 00:16:25.869] [debug][Consumer]Query.id: car123434
[2025-09-25 00:16:25.869] [info][Consumer]Created name /SNDS/CAR_0/
    temporalQuery:id%3Dcar123434&type%3DCAR_0&date%3D&count%3D&
    timeframe%3D&filters%3D
[2025-09-25 00:16:25.869] [info][Consumer]Sending actual Interest /
    SNDS/CAR_0/temporalQuery%3Aid%3Dcar123434%26type%3DCAR_0%26date
    %3D%26count%3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime
    =6000
[2025-09-25 00:16:25.871] [info][Consumer]Received Data with name: /
    SNDS/CAR_0/temporalQuery%3Aid%3Dcar123434%26type%3DCAR_0%26date
    %3D%26count%3D%26timeframe%3D%26filters%3D in function onData
[2025-09-25 00:16:25.871] [info][Consumer]Data packet payload: null
    in function onData
[2025-09-25 00:16:25.871] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: CAR_0, Command: temporalQuery, Params: [id=
    car123434, type=CAR_0, date=, count=, timeframe=, filters=] in
    onData
[2025-09-25 00:16:25.871] [info][Consumer]Temporal Query Reponse...
[2025-09-25 00:16:25.871] [error][Consumer]No JSON start detected.
    First 32 bytes: 6e 75 6c 6c
[2025-09-25 00:16:25.871] [info][Consumer]Reading packet...
[2025-09-25 00:16:25.871] [debug][Consumer]Extracted content from
    message: /SNDS/CAR_0/getByIDForType:car1234&CAR_0
[2025-09-25 00:16:25.871] [debug][Consumer]Extracted content length
    from message: 40
[2025-09-25 00:16:25.871] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
[2025-09-25 00:16:25.871] [debug][Consumer]Query.type: CAR_0
[2025-09-25 00:16:25.871] [debug][Consumer]Query.id: car1234
[2025-09-25 00:16:25.871] [info][Consumer]Created name /SNDS/CAR_0/
    temporalQuery:id%3Dcar1234&type%3DCAR_0&date%3D&count%3D&
    timeframe%3D&filters%3D
[2025-09-25 00:16:25.871] [info][Consumer]Sending actual Interest /
    SNDS/CAR_0/temporalQuery%3Aid%3Dcar1234%26type%3DCAR_0%26date%3D
```

https://doi.org/10.26219/heal.aueb.9822

```
    %26count%3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime
    =6000
[2025-09-25 00:16:25.872] [info][Consumer]Received Data with name: /
    SNDS/CAR_0/temporalQuery%3Aid%3Dcar1234%26type%3DCAR_0%26date%3D
    %26count%3D%26timeframe%3D%26filters%3D in function onData
[2025-09-25 00:16:25.872] [info][Consumer]Data packet payload: null
    in function onData
[2025-09-25 00:16:25.872] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: CAR_0, Command: temporalQuery, Params: [id=car1234,
    type=CAR_0, date=, count=, timeframe=, filters=] in onData
[2025-09-25 00:16:25.872] [info][Consumer]Temporal Query Reponse...
[2025-09-25 00:16:25.872] [error][Consumer]No JSON start detected.
    First 32 bytes: 6e 75 6c 6c
[2025-09-25 00:16:25.872] [info][Consumer]Reading packet...
[2025-09-25 00:16:25.872] [debug][Consumer]Extracted content from
    message: /SNDS/CAR_0/getByIDForType:car124&CAR_0
[2025-09-25 00:16:25.872] [debug][Consumer]Extracted content length
    from message: 39
[2025-09-25 00:16:25.872] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
[2025-09-25 00:16:25.872] [debug][Consumer]Query.type: CAR_0
[2025-09-25 00:16:25.872] [debug][Consumer]Query.id: car124
[2025-09-25 00:16:25.872] [info][Consumer]Created name /SNDS/CAR_0/
    temporalQuery:id%3Dcar124&type%3DCAR_0&date%3D&count%3D&
    timeframe%3D&filters%3D
[2025-09-25 00:16:25.872] [info][Consumer]Sending actual Interest /
    SNDS/CAR_0/temporalQuery%3Aid%3Dcar124%26type%3DCAR_0%26date%3D
    %26count%3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime
    =6000
[2025-09-25 00:16:25.874] [info][Consumer]Received Data with name: /
    SNDS/CAR_0/temporalQuery%3Aid%3Dcar124%26type%3DCAR_0%26date%3D
    %26count%3D%26timeframe%3D%26filters%3D in function onData
[2025-09-25 00:16:25.874] [info][Consumer]Data packet payload: null
    in function onData
[2025-09-25 00:16:25.874] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: CAR_0, Command: temporalQuery, Params: [id=car124,
    type=CAR_0, date=, count=, timeframe=, filters=] in onData
[2025-09-25 00:16:25.874] [info][Consumer]Temporal Query Reponse...
[2025-09-25 00:16:25.874] [error][Consumer]No JSON start detected.
    First 32 bytes: 6e 75 6c 6c
[2025-09-25 00:16:25.874] [info][Consumer]Reading packet...
[2025-09-25 00:16:25.874] [debug][Consumer]Extracted content from
    message: /SNDS/CAR_0/getByIDForType:car54&CAR_0
[2025-09-25 00:16:25.874] [debug][Consumer]Extracted content length
    from message: 38
[2025-09-25 00:16:25.874] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
```

```
[2025-09-25 00:16:25.874] [debug][Consumer]Query.type: CAR_0
[2025-09-25 00:16:25.874] [debug][Consumer]Query.id: car54
[2025-09-25 00:16:25.874] [info][Consumer]Created name /SNDS/CAR_0/
    temporalQuery:id%3Dcar54&type%3DCAR_0&date%3D&count%3D&timeframe
    %3D&filters%3D
[2025-09-25 00:16:25.874] [info][Consumer]Sending actual Interest /
    SNDS/CAR_0/temporalQuery%3Aid%3Dcar54%26type%3DCAR_0%26date%3D
    %26count%3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime
    =6000
[2025-09-25 00:16:25.875] [info][Consumer]Received Data with name: /
    SNDS/CAR_0/temporalQuery%3Aid%3Dcar54%26type%3DCAR_0%26date%3D
    %26count%3D%26timeframe%3D%26filters%3D in function onData
[2025-09-25 00:16:25.875] [info][Consumer]Data packet payload: null
    in function onData
[2025-09-25 00:16:25.875] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: CAR_0, Command: temporalQuery, Params: [id=car54,
    type=CAR_0, date=, count=, timeframe=, filters=] in onData
[2025-09-25 00:16:25.875] [info][Consumer]Temporal Query Reponse...
[2025-09-25 00:16:25.875] [error][Consumer]No JSON start detected.
    First 32 bytes: 6e 75 6c 6c
[2025-09-25 00:16:25.875] [info][Consumer]Reading packet...
```

The *GET by TYPE* evaluation demonstrated the ability of SeEDS to retrieve data objects associated with a specific `@type` across the distributed NDN network. By publishing multiple `@ids` under the shared `@type=CAR_0` registry at node A, other nodes, in this case node D, were able to discover and access the corresponding data through Interest packets constructed at the type level. This confirmed that the registry mechanism correctly advertises type scopes (e.g., `/CAR_0`) and propagates them throughout the network, enabling type-based content discovery.

The experiment verified that when a consumer issues a *GET by TYPE* request, the SeEDS Service translates it into an NDN Interest targeting the appropriate registry scope. Node A, acting as the producer, responds with the set of matching entries, allowing the consumer, in this case D, to perform a *GET by ID* for each `@id` (entry).

A.2 Resilience

The primary objective of this experiment was to measure system resilience under partial failure conditions, with the specific target of demonstrating that recovery latency does not exceed 200 ms when 50% of the data intermediary nodes fail.

A.2.1 Topology

The experimental evaluation was carried out on a seven-node star topology, shown in Figure 15. In this configuration, node D functions as the central hub responsible for interconnecting all peripheral nodes through the Named Data Networking Forwarding Daemon (NFD). The peripheral nodes do not maintain direct links with one another; instead, every communication event must traverse the central hub. Consequently, all data exchanges and request forwarding operations are mediated by node D.

The nodes that will fail here are A,B,C which will be the Primary Nodes and the nodes that will be set up to be their Secondaries, and will remain alive, will be E,F,G.

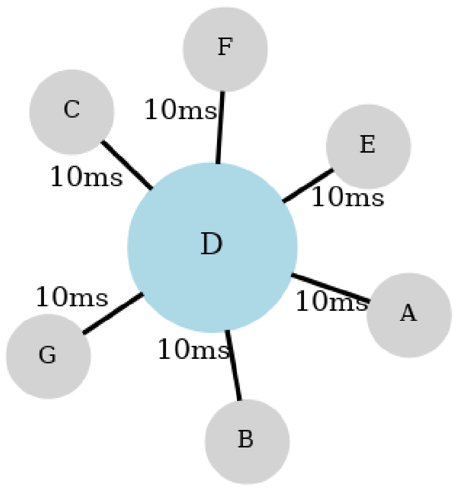


Figure 15: Experimental topology for resilience experiments.

Here's the config file inserted in MiniNDN:

Listing 9: The MiniNDN config for creating a star topology

```
[nodes]
a: _nfd-log-level=DEBUG
b: _nfd-log-level=DEBUG
c: _nfd-log-level=DEBUG
d: _nfd-log-level=DEBUG
e: _nfd-log-level=DEBUG
f: _nfd-log-level=DEBUG
g: _nfd-log-level=DEBUG
[links]
d:a delay=10ms
a:d delay=10ms
```

```
d:b delay=10ms
b:d delay=10ms
c:d delay=10ms
d:c delay=10ms
d:e delay=10ms
e:d delay=10ms
d:f delay=10ms
f:d delay=10ms
d:g delay=10ms
g:d delay=10ms
```

A.2.2 Failure Injection

To emulate node failures, we deliberately terminated half of the non-hub nodes during the experiment. This was achieved by applying a configurable timeout mechanism, after which the designated Primary nodes were forced to stop their daemons and thus become unavailable. The remaining nodes (Secondary) continued to operate, monitoring their paired failing counterparts through the resilience poller.

A.2.3 Data Replication

Prior to node failures, data was actively published into the NDN network by all participating nodes. Each node contributed multiple data objects (e.g., key-value pairs of IDs and associated types) into the system, ensuring that the global dataset was replicated across the distributed instances running locally on each host. The publication mechanism guaranteed redundancy, such that information was not confined to a single point of failure.

A.2.4 Failure Recovery and System Recovery Time

Following the induced failures, retrieval operations were performed exclusively from the surviving Secondary nodes. By design, these nodes were configured to subscribe to content from their peers, ensuring that they could continue to respond to queries despite the absence of their Primary counterparts. This allowed us to verify that replicated data remained accessible through alternative nodes.

Recovery was evaluated by analyzing the log files produced during the experiment. Each node continuously recorded system events with millisecond granularity. The recovery time metric was defined as the difference between two logged timestamps:

- The timestamp recorded at the moment a Primary node was declared failed, which corresponds to the daemon termination triggered by the timeout mechanism.

- The timestamp of the subsequent event when the paired Secondary node successfully published the final replicated data entry back into the NDN network.

This difference represents the effective duration required for the system to detect the failure, promote the Secondary to Primary, and ensure that data availability was restored.

A.2.5 Results of Failure Recovery

Recall that each node uses it's unique ID to make the following announcements:

Listing 10: Local announcements for each Node

```
[2025-09-26 07:53:42.456] [info][Producer]Announcing ID Scope name:
    SNDS/SNDS_ID_scope_id
[2025-09-26 07:53:42.456] [info][Producer]Announcing name: SNDS/
    is_alive_id
[2025-09-26 07:53:42.456] [info][Producer]Announcing name: SNDS/
    state_id
[2025-09-26 07:53:42.457] [info][Producer]Announcing name: /SNDS/
    CAR_id
[2025-09-26 07:53:42.457] [info][Producer]Announcing name: /SNDS/
    CAR_id_registry
[2025-09-26 07:53:42.458] [info][Producer]Successfully registered
    prefix: /SNDS_id
```

The `ID` serves as a placeholder for each node's unique identifier. The *unique IDs* of each Node follow the alphabetical order and are zero-indexed. The nodes are configured to publish data to the same node's `@type` and `Subscriber` registries, as well as to publish the corresponding `@ids` locally. This operation relies on each node's unique node ID. For instance, when publishing to node A's registry, which has a unique node ID of 0, the data is published under the `@type=CAR_0` registry. Another example would be, that the `@id=car0` should be published inside Node A which has a unique node ID of 0 or that the `@id=car1` should be published inside Node B which has a unique node ID of 1. This procedure is repeated for all Nodes, ensuring that each node maintains and announces its own unique data within the network. Recall that the Primary Nodes are A,B,C so we should expect the Secondary Nodes E,F,G to contain the data of the Primaries once they fail.

Inside node A we must see `@id=car0` get announced successfully:

Listing 11: Publish @id=car0

```
[2025-09-26 10:01:07.876] [info][HTTPServer] Received POST request
    for / with body id=car0
```

```
[2025-09-26 10:01:07.876] [info][HTTPServer] Received POST request
    from 127.0.0.1
Message: {id=car0}
Type: BY_ID_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car0
[2025-09-26 10:01:07.876] [debug][HTTPServer] Writing message from
    POST request to service queue...
Message: {id=car0}
Type: BY_ID_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car0

[2025-09-26 10:01:07.876] [debug][DigitalTwin]Extracted content from
    message {id=car0}
[2025-09-26 10:01:07.876] [debug][DigitalTwin]Extracted content
    length from message 9
[2025-09-26 10:01:07.876] [debug][DigitalTwin]Extracted type from
    message: BY_ID_ADVERTISEMENT
[2025-09-26 10:01:07.876] [info][DigitalTwin]Received by ID get
    request...
[2025-09-26 10:01:07.876] [debug][DigitalTwin]Pushed ID car0 to the
    announced IDs vector...
[2025-09-26 10:01:07.876] [debug][DigitalTwin]Extracted id Scope from
     message SNDS_ID_scope_0
Message: /SNDS/car0/postID:car0
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:
[2025-09-26 10:01:07.876] [info][DigitalTwin]Pushing byID
    advertisement to the Producer...
Message: /SNDS/car0/postID:car0
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:

[2025-09-26 10:01:07.876] [debug][DigitalTwin]Sleeping 2000 ms before
```

```
     sending scope advertisement...
[2025-09-26 10:01:07.880] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/state_0/getResilienceMeta%3A
[2025-09-26 10:01:07.880] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: state_0, Command: getResilienceMeta, Params: []
[2025-09-26 10:01:07.881] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/is_alive_0/
    getResilienceIsAlive%3A17432869098159683528
[2025-09-26 10:01:07.882] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: is_alive_0, Command: getResilienceIsAlive, Params:
    [17432869098159683528]
[2025-09-26 10:01:07.882] [debug][Producer]Sending data to the NDN...
[2025-09-26 10:01:07.902] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/is_alive_0/
    getResilienceIsAlive%3A11341762269107256235
[2025-09-26 10:01:07.902] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: is_alive_0, Command: getResilienceIsAlive, Params:
    [11341762269107256235]
[2025-09-26 10:01:07.902] [debug][Producer]Sending data to the NDN...
[2025-09-26 10:01:07.920] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/state_0/getResilienceMeta%3A
[2025-09-26 10:01:07.920] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: state_0, Command: getResilienceMeta, Params: []
[2025-09-26 10:01:07.922] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/is_alive_0/
    getResilienceIsAlive%3A7264770695334591861
[2025-09-26 10:01:07.922] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: is_alive_0, Command: getResilienceIsAlive, Params:
    [7264770695334591861]
[2025-09-26 10:01:07.922] [debug][Producer]Sending data to the NDN...
[2025-09-26 10:01:07.942] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/is_alive_0/
    getResilienceIsAlive%3A10537152589682238699
[2025-09-26 10:01:07.942] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: is_alive_0, Command: getResilienceIsAlive, Params:
    [10537152589682238699]
[2025-09-26 10:01:07.942] [debug][Producer]Sending data to the NDN...
[2025-09-26 10:01:07.960] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/state_0/getResilienceMeta%3A
[2025-09-26 10:01:07.960] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: state_0, Command: getResilienceMeta, Params: []
[2025-09-26 10:01:07.962] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/is_alive_0/
    getResilienceIsAlive%3A12718435922522655948
[2025-09-26 10:01:07.962] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: is_alive_0, Command: getResilienceIsAlive, Params:
    [12718435922522655948]
```

```
[2025-09-26 10:01:07.962] [debug][Producer]Sending data to the NDN...
Message: /SNDS/car0/postID:car0
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:
[2025-09-26 10:01:07.971] [info][Producer]Got new ID advertisement
    request...
Message: /SNDS/car0/postID:car0
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:

[2025-09-26 10:01:07.971] [debug][Producer]Parsed interest Prefix:
    SNDS, Name: car0, Command: postID, Params: [car0] from string
[2025-09-26 10:01:07.971] [info][Producer]Announcing name: /SNDS/car0
```

We can see in the above that Node A successfully published `@id=car0` to the NDN network. Similarly, inside Node B we should see `@id=car1` get announced successfully:

Listing 12: Publish @id=car1

```
[2025-09-26 10:01:19.920] [info][HTTPServer] Received POST request
    for / with body id=car1
[2025-09-26 10:01:19.920] [info][HTTPServer] Received POST request
    from 127.0.0.1
Message: {id=car1}
Type: BY_ID_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car1
[2025-09-26 10:01:19.920] [debug][HTTPServer] Writing message from
    POST request to service queue...
Message: {id=car1}
Type: BY_ID_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car1

[2025-09-26 10:01:19.920] [debug][DigitalTwin]Extracted content from
```

```
    message {id=car1}
[2025-09-26 10:01:19.920] [debug][DigitalTwin]Extracted content
    length from message 9
[2025-09-26 10:01:19.920] [debug][DigitalTwin]Extracted type from
    message: BY_ID_ADVERTISEMENT
[2025-09-26 10:01:19.920] [info][DigitalTwin]Received by ID get
    request...
[2025-09-26 10:01:19.920] [debug][DigitalTwin]Pushed ID car1 to the
    announced IDs vector...
[2025-09-26 10:01:19.920] [debug][DigitalTwin]Extracted id Scope from
     message SNDS_ID_scope_5
Message: /SNDS/car1/postID:car1
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:
[2025-09-26 10:01:19.920] [info][DigitalTwin]Pushing byID
    advertisement to the Producer...
Message: /SNDS/car1/postID:car1
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:

[2025-09-26 10:01:19.920] [debug][DigitalTwin]Sleeping 2000 ms before
     sending scope advertisement...
Message: /SNDS/car1/postID:car1
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:
[2025-09-26 10:01:19.921] [info][Producer]Got new ID advertisement
    request...
Message: /SNDS/car1/postID:car1
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:

[2025-09-26 10:01:19.921] [debug][Producer]Parsed interest Prefix:
    SNDS, Name: car1, Command: postID, Params: [car1] from string
[2025-09-26 10:01:19.921] [info][Producer]Announcing name: /SNDS/car1
```

Now let's check whether the Secondary Nodes can actually recover.

https://doi.org/10.26219/heal.aueb.9822

Here is the timestamp when Node A shuts down:

Listing 13: A's timeout timestamp

```
[TIMEOUT] 2025-09-26 10:03:08.446 a: daemon exceeded timeout
```

And here is the log of Node E, which is set up to be the Secondary of Node A:

Listing 14: E's recovery logs

```
[2025-09-26 10:03:08.456] [warning][Producer][resilience-ipc]
    promotion to PRIMARY (reason=alive-missed, misses=1)
[2025-09-26 10:03:08.456] [info][Producer][resilience] mutate -> ver
    =6 hash=73f0555c38af9c19ece369fcb898abc0
[2025-09-26 10:03:08.456] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(c76727ce) 2(04168ec0) 3(b4d35eeb) 4(
    e9df653a) 5(0ef09630) 6(73f0555c)
[2025-09-26 10:03:08.456] [debug][Producer][resilience] ops from v5
    -> v6:
[
  {
    "op": "replace",
    "path": "/resilience/reason",
    "value": "alive-missed"
  },
  {
    "op": "replace",
    "path": "/resilience/role",
    "value": "primary"
  },
  {
    "op": "replace",
    "path": "/resilience/since",
    "value": "2025-09-26T07:03:08Z"
  },
  {
    "op": "add",
    "path": "/resilience/misses",
    "value": 1
  }
]
[2025-09-26 10:03:08.456] [warning][Producer][resilience-ipc] stop
    requested; will be joined from main loop
[2025-09-26 10:03:08.522] [warning][Producer][resilience] applying
    promotion side-effects on main loop
[2025-09-26 10:03:08.522] [debug][Producer][overlay] merge: applying
    json_patch_against_base (ops=7)
[2025-09-26 10:03:08.522] [debug][Producer][resilience] assign -> ver
    =7 hash=19f8dbf49001bae4f664a330312d42a9
```

```
[2025-09-26 10:03:08.522] [debug][Producer][overlay] SQUASH -> base
    version 6->7 hash 73f0555c38af9c19ece369fcb898abc0->19
    f8dbf49001bae4f664a330312d42a9 overlay ver=16 hash=994121709
    bfc030ff093864add91033d
[2025-09-26 10:03:08.522] [debug][Producer][overlay] CLEARED
[2025-09-26 10:03:08.523] [debug][Producer][promotion] rebuilt
    containers from store:
[2025-09-26 10:03:08.523] [debug][Producer]type_registry[CAR_0] =
    [car10 ]
[2025-09-26 10:03:08.523] [debug][Producer]type_registry[CAR_4] =
    [car14 ]
[2025-09-26 10:03:08.523] [debug][Producer]
    subscribed_nodes_for_type[CAR_0] = [SNDS_ID_scope_6 ]
[2025-09-26 10:03:08.523] [debug][Producer]
    subscribed_nodes_for_type[CAR_4] = [SNDS_ID_scope_3 ]
[2025-09-26 10:03:08.523] [debug][Producer]announcements = [/SNDS/
    SNDS_ID_scope_4 /SNDS/is_alive_4 /SNDS/state_4 /SNDS/
    CAR_4_registry /SNDS/CAR_4 /SNDS/car4 /SNDS_0 /SNDS/car0 /SNDS/
    CAR_0 /SNDS/CAR_0_registry /SNDS/is_alive_0 /SNDS/
    SNDS_ID_scope_0 /SNDS_4 /SNDS/state_0 ]
[2025-09-26 10:03:08.523] [info][Producer][promotion] summary: types
    =2 ids=2 subs(byId)=0 subs(byType)=2 announced=14
[2025-09-26 10:03:08.523] [info][Producer][promotion] (re)announcing
    /SNDS/SNDS_ID_scope_4
[2025-09-26 10:03:08.523] [info][Producer][promotion] (re)announcing
    /SNDS/is_alive_4
[2025-09-26 10:03:08.523] [info][Producer][promotion] (re)announcing
    /SNDS/state_4
[2025-09-26 10:03:08.523] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_4_registry
[2025-09-26 10:03:08.523] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_4
[2025-09-26 10:03:08.523] [info][Producer][promotion] (re)announcing
    /SNDS/car4
[2025-09-26 10:03:08.524] [info][Producer][promotion] (re)announcing
    /SNDS_0
[2025-09-26 10:03:08.524] [info][Producer][promotion] (re)announcing
    /SNDS/car0
[2025-09-26 10:03:08.524] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_0
[2025-09-26 10:03:08.524] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_0_registry
[2025-09-26 10:03:08.524] [info][Producer][promotion] (re)announcing
    /SNDS/is_alive_0
[2025-09-26 10:03:08.524] [info][Producer][promotion] (re)announcing
    /SNDS/SNDS_ID_scope_0
[2025-09-26 10:03:08.524] [info][Producer][promotion] (re)announcing
```

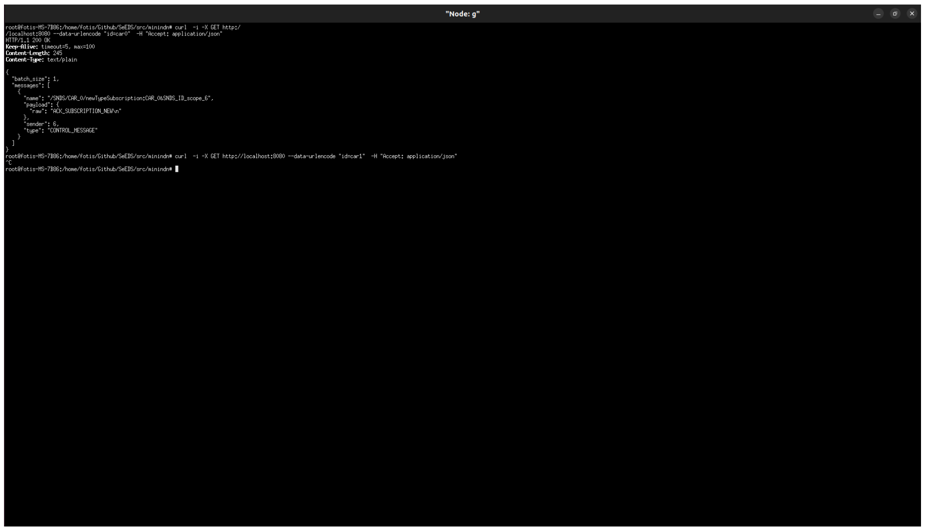
https://doi.org/10.26219/heal.aueb.9822

```
    /SNDS_4
[2025-09-26 10:03:08.525] [info][Producer][promotion] (re)announcing
    /SNDS/state_0
[2025-09-26 10:03:08.525] [info][Producer]Getting from IP: 10.0.0.26
    and port: 10000
[2025-09-26 10:03:08.525] [debug][Producer]HTTP GET /state?type=CAR_0
[2025-09-26 10:03:08.527] [info][Producer][resilience] mutate -> ver
    =8 hash=85a5cbdd3d895ce5212470a8dfe17f55
[2025-09-26 10:03:08.527] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(c76727ce) 2(04168ec0) 3(b4d35eeb) 4(
    e9df653a) 5(0ef09630) 6(73f0555c) 7(19f8dbf4) 8(85a5cbdd)
[2025-09-26 10:03:08.527] [debug][Producer][resilience] ops from v7
    -> v8:
[
  {
    "op": "replace",
    "path": "/broker/collections/CAR_0/lastSeen",
    "value": "2025-09-26T07:03:08Z"
  }
]
[2025-09-26 10:03:08.527] [info][Producer]Getting from IP: 10.0.0.26
    and port: 10000
[2025-09-26 10:03:08.527] [debug][Producer]HTTP GET /state?type=CAR_4
[2025-09-26 10:03:08.528] [info][Producer][resilience] mutate -> ver
    =9 hash=71624b465ef5120729971953ad41ec36
[2025-09-26 10:03:08.529] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(c76727ce) 2(04168ec0) 3(b4d35eeb) 4(
    e9df653a) 5(0ef09630) 6(73f0555c) 7(19f8dbf4) 8(85a5cbdd)
    9(71624b46)
[2025-09-26 10:03:08.529] [debug][Producer][resilience] ops from v8
    -> v9:
[
  {
    "op": "add",
    "path": "/broker/collections/CAR_4",
    "value": {
      "algo": "SHA-256",
      "count": 0,
      "hash": "
          e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
          ",
      "lastSeen": "2025-09-26T07:03:08Z"
    }
  }
]
[2025-09-26 10:03:08.530] [info][Producer]Successfully registered
    prefix: /SNDS/SNDS_ID_scope_4
```

```
[2025-09-26 10:03:08.530] [info][Producer]Successfully registered
    prefix: /SNDS/is_alive_4
[2025-09-26 10:03:08.530] [info][Producer]Successfully registered
    prefix: /SNDS/state_4
[2025-09-26 10:03:08.530] [info][Producer]Successfully registered
    prefix: /SNDS/CAR_4_registry
[2025-09-26 10:03:08.532] [info][Producer]Successfully registered
    prefix: /SNDS/CAR_4
[2025-09-26 10:03:08.532] [info][Producer]Successfully registered
    prefix: /SNDS/car4
[2025-09-26 10:03:08.533] [info][Producer]Successfully registered
    prefix: /SNDS_0
[2025-09-26 10:03:08.533] [info][Producer]Successfully registered
    prefix: /SNDS/car0
[2025-09-26 10:03:08.533] [info][Producer]Successfully registered
    prefix: /SNDS/CAR_0
[2025-09-26 10:03:08.533] [info][Producer]Successfully registered
    prefix: /SNDS/CAR_0_registry
[2025-09-26 10:03:08.533] [info][Producer]Successfully registered
    prefix: /SNDS/is_alive_0
[2025-09-26 10:03:08.533] [info][Producer]Successfully registered
    prefix: /SNDS/SNDS_ID_scope_0
[2025-09-26 10:03:08.533] [info][Producer]Successfully registered
    prefix: /SNDS_4
[2025-09-26 10:03:08.533] [info][Producer]Successfully registered
    prefix: /SNDS/state_0
```

We see that the failure of Node A happened at **2025-09-26 10:03:08.446** and its Secondary (E) published the last data at **2025-09-26 10:03:08.533**. This means that the system recovered in 87 ms. This is to be expected, because the data is continuously replicated in our system and it is essentially ready to be published once the Primary Node fails.

Let's see now if the data of A, which now are essentially the data of E, are retrievable through the network. Let's ask for data through G, specifically for the `@id=car0`:



```
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# curl  -i -X GET http:/
/localhost:8080 --data-urlencode "id=car0"  -H "Accept: application/json"
HTTP/1.1 200 OK
Keep-Alive: timeout=5, max=100
Content-Length: 245
Content-Type: text/plain

{
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_0/newTypeSubscription:CAR_0&SNDS_ID_scope_6",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NEW\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# curl  -i -X GET http://localhost:8080 --data-urlencode "id=car1"  -H "Accept: application/json"
^C
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# 
```

Figure 16: Request `@id=car0` and `@id=car1` through G.

Let's check E's logs, which now provides the data of A:

Listing 15: E's logs after becoming the provider of A's data

```
[2025-09-26 10:22:14.393] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/car0/temporalQuery%3Aid%3Dcar0
    %26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters%3D
[2025-09-26 10:22:14.394] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: car0, Command: temporalQuery, Params: [id=car0, type
    =, date=, count=, timeframe=, filters=]
[2025-09-26 10:22:14.394] [debug][Producer]Created interest response:
     /SNDS/car0/temporalQuery%3Aid%3Dcar0%26type%3D%26date%3D%26
    count%3D%26timeframe%3D%26filters%3D in:
    process_temporal_query_interest
[2025-09-26 10:22:14.394] [debug][Producer]TemporalQuery -> id: car0,
     date: , type: , timeframe: , count: , filters:
[2025-09-26 10:22:14.394] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car0&type=&count=&date=&timeframe=&filters=
[2025-09-26 10:22:14.395] [info][Producer]Successfully queried /
    temporal-query
[2025-09-26 10:22:14.395] [debug][Producer]Sending data to the NDN...
```

Let's check G's logs, which requests the data of A:

Listing 16: G performs the request to retrieve ex A's data, which are now provided through E

```
[2025-09-26 10:22:14.393] [info][HTTPServer] Received GET request for
     / with body id=car0
```

```
[2025-09-26 10:22:14.393] [info][HTTPServer] Received GET request for
      / with target /
[2025-09-26 10:22:14.393] [info][HTTPServer] Received GET request
     from 127.0.0.1
Message: {id=car0}
Type: GET_BY_ID_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
id = car0
[2025-09-26 10:22:14.393] [debug][HTTPServer] Writing message from
     GET request to service queue...
Message: {id=car0}
Type: GET_BY_ID_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
id = car0

[2025-09-26 10:22:14.393] [info][Worker]Received content: {id=car0}
[2025-09-26 10:22:14.393] [info][Worker]Received payload:
[2025-09-26 10:22:14.393] [info][Worker]Received type:
     GET_BY_ID_REQUEST
[2025-09-26 10:22:14.393] [debug][Worker]Server intra message...
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_0/newTypeSubscription:CAR_0&SNDS_ID_scope_6",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NEW\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:
[2025-09-26 10:22:14.393] [info][HTTPServer] Read Message from queue:
Message: BATCH
```

```
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_0/newTypeSubscription:CAR_0&SNDS_ID_scope_6",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NEW\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:

[2025-09-26 10:22:14.393] [info][Worker]Inserted ID: car0 into set.
    Pushing regular getByID request for ID: car0
[2025-09-26 10:22:14.393] [info][Worker]Reading packet...
[2025-09-26 10:22:14.393] [debug][Consumer]Extracted content from
    message: /SNDS/car0/getByID:car0
[2025-09-26 10:22:14.393] [debug][Consumer]Extracted content length
    from message: 23
[2025-09-26 10:22:14.393] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_REQUEST
[2025-09-26 10:22:14.393] [debug][Consumer]Query.type:
[2025-09-26 10:22:14.393] [debug][Consumer]Query.id: car0
[2025-09-26 10:22:14.393] [info][Consumer]Created name /SNDS/car0/
    temporalQuery:id%3Dcar0&type%3D&date%3D&count%3D&timeframe%3D&
    filters%3D
[2025-09-26 10:22:14.393] [info][Consumer]Sending actual Interest /
    SNDS/car0/temporalQuery%3Aid%3Dcar0%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime=6000
[2025-09-26 10:22:14.396] [info][Consumer]Received Data with name: /
    SNDS/car0/temporalQuery%3Aid%3Dcar0%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D in function onData
[2025-09-26 10:22:14.396] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car0",
      "id": "car0",
      "timestamp": {
        "$date": 1758870068072
      },
      "type": "GENERIC"
    }
  }
] in function onData
[2025-09-26 10:22:14.396] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: car0, Command: temporalQuery, Params: [id=car0, type
    =, date=, count=, timeframe=, filters=] in onData
[2025-09-26 10:22:14.396] [info][Consumer]Temporal Query Reponse...
[2025-09-26 10:22:14.396] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-09-26 10:22:14.396] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car0",
      "id": "car0",
      "timestamp": {
        "$date": 1758870068072
      },
      "type": "GENERIC"
    }
  }
]
[2025-09-26 10:22:14.396] [debug][Consumer]Sanitized payload to:
[
  {
    "": {
      "_id": "car0",
      "id": "car0",
      "timestamp": {
        "$date": 1758870068072
      },
      "type": "GENERIC"
    }
  }
]
```

We can see that G got a response back, after requesting it from E. Let's also check B's logs, which is another Primary Node:

Listing 17: Timeout timestamp of Node B

```
[TIMEOUT] 2025-09-26 10:03:11.493 b: daemon exceeded timeout
```

Let's check F's logs, which is the Secondary Node of B:

Listing 18: Logs of Node F after providing the data of Node B

```
[2025-09-26 10:03:11.496] [warning][Producer][resilience-ipc]
    promotion to PRIMARY (reason=alive-missed, misses=1)
[2025-09-26 10:03:11.496] [info][Producer][resilience] mutate -> ver
    =6 hash=ac41eea00ae8385b88550b5d064cee98
[2025-09-26 10:03:11.496] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(3b6ede70) 2(b4b2de78) 3(050d6435) 4(20
    c5c4af) 5(7ee82c13) 6(ac41eea0)
[2025-09-26 10:03:11.496] [debug][Producer][resilience] ops from v5
    -> v6:
[
  {
    "op": "replace",
    "path": "/resilience/reason",
    "value": "alive-missed"
  },
  {
    "op": "replace",
    "path": "/resilience/role",
    "value": "primary"
  },
  {
    "op": "replace",
    "path": "/resilience/since",
    "value": "2025-09-26T07:03:11Z"
  },
  {
    "op": "add",
    "path": "/resilience/misses",
    "value": 1
  }
]
[2025-09-26 10:03:11.496] [warning][Producer][resilience-ipc] stop
    requested; will be joined from main loop
[2025-09-26 10:03:11.530] [warning][Producer][resilience] applying
    promotion side-effects on main loop
[2025-09-26 10:03:11.530] [debug][Producer][overlay] merge: applying
    json_patch_against_base (ops=7)
[2025-09-26 10:03:11.530] [debug][Producer][resilience] assign -> ver
    =7 hash=9b408c51324d2bb669672017a28994e2
[2025-09-26 10:03:11.530] [debug][Producer][overlay] SQUASH -> base
    version 6->7 hash ac41eea00ae8385b88550b5d064cee98->9
    b408c51324d2bb669672017a28994e2 overlay ver=16 hash=59
    d0966e24c8873e2ec151dfd7a8b395
[2025-09-26 10:03:11.530] [debug][Producer][overlay] CLEARED
[2025-09-26 10:03:11.530] [debug][Producer][promotion] rebuilt
    containers from store:
[2025-09-26 10:03:11.530] [debug][Producer]type_registry[CAR_1] =
```

https://doi.org/10.26219/heal.aueb.9822

```
    [car11 ]
[2025-09-26 10:03:11.531] [debug][Producer]type_registry[CAR_5] =
    [car15 ]
[2025-09-26 10:03:11.531] [debug][Producer]
    subscribed_nodes_for_type[CAR_1] = [SNDS_ID_scope_0 ]
[2025-09-26 10:03:11.531] [debug][Producer]
    subscribed_nodes_for_type[CAR_5] = [SNDS_ID_scope_4 ]
[2025-09-26 10:03:11.531] [debug][Producer]announcements = [/SNDS/
    SNDS_ID_scope_5 /SNDS/is_alive_5 /SNDS/CAR_1 /SNDS/CAR_5 /SNDS/
    CAR_5_registry /SNDS_1 /SNDS_5 /SNDS/is_alive_1 /SNDS/car5 /SNDS
    /CAR_1_registry /SNDS/SNDS_ID_scope_1 /SNDS/car1 /SNDS/state_5 /
    SNDS/state_1 ]
[2025-09-26 10:03:11.531] [info][Producer][promotion] summary: types
    =2 ids=2 subs(byId)=0 subs(byType)=2 announced=14
[2025-09-26 10:03:11.531] [info][Producer][promotion] (re)announcing
    /SNDS/SNDS_ID_scope_5
[2025-09-26 10:03:11.531] [info][Producer][promotion] (re)announcing
    /SNDS/is_alive_5
[2025-09-26 10:03:11.531] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_1
[2025-09-26 10:03:11.531] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_5
[2025-09-26 10:03:11.531] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_5_registry
[2025-09-26 10:03:11.531] [info][Producer][promotion] (re)announcing
    /SNDS_1
[2025-09-26 10:03:11.532] [info][Producer][promotion] (re)announcing
    /SNDS_5
[2025-09-26 10:03:11.532] [info][Producer][promotion] (re)announcing
    /SNDS/is_alive_1
[2025-09-26 10:03:11.532] [info][Producer][promotion] (re)announcing
    /SNDS/car5
[2025-09-26 10:03:11.532] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_1_registry
[2025-09-26 10:03:11.532] [info][Producer][promotion] (re)announcing
    /SNDS/SNDS_ID_scope_1
[2025-09-26 10:03:11.532] [info][Producer][promotion] (re)announcing
    /SNDS/car1
[2025-09-26 10:03:11.532] [info][Producer][promotion] (re)announcing
    /SNDS/state_5
[2025-09-26 10:03:11.532] [info][Producer][promotion] (re)announcing
    /SNDS/state_1
[2025-09-26 10:03:11.533] [info][Producer]Getting from IP: 10.0.0.34
    and port: 10000
[2025-09-26 10:03:11.533] [debug][Producer]HTTP GET /state?type=CAR_1
[2025-09-26 10:03:11.535] [info][Producer][resilience] mutate -> ver
    =8 hash=d13592ce859b022333f677c5267aa2b9
```

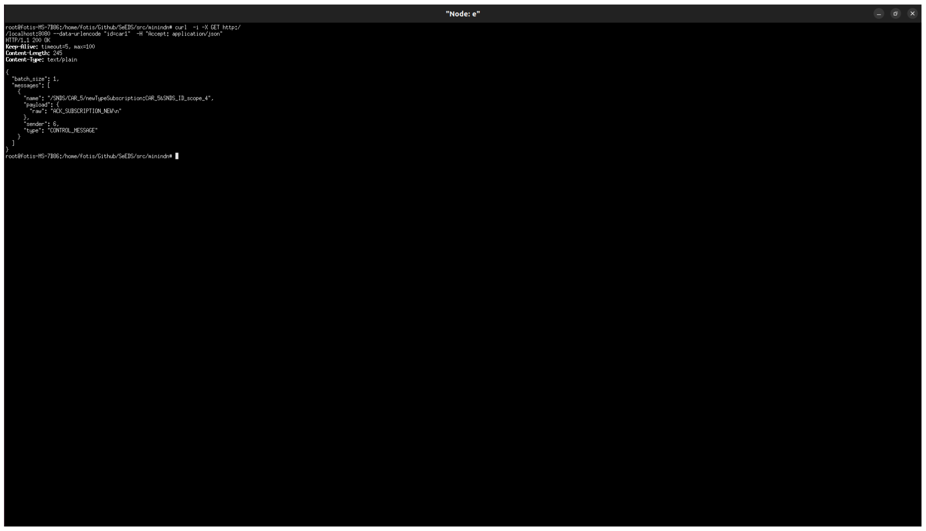
https://doi.org/10.26219/heal.aueb.9822

```
[2025-09-26 10:03:11.535] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(3b6ede70) 2(b4b2de78) 3(050d6435) 4(20
    c5c4af) 5(7ee82c13) 6(ac41eea0) 7(9b408c51) 8(d13592ce)
[2025-09-26 10:03:11.535] [debug][Producer][resilience] ops from v7
    -> v8:
[
  {
    "op": "replace",
    "path": "/broker/collections/CAR_1/lastSeen",
    "value": "2025-09-26T07:03:11Z"
  }
]
[2025-09-26 10:03:11.535] [info][Producer]Getting from IP: 10.0.0.34
    and port: 10000
[2025-09-26 10:03:11.535] [debug][Producer]HTTP GET /state?type=CAR_5
[2025-09-26 10:03:11.537] [info][Producer][resilience] mutate -> ver
    =9 hash=1d452cd075217771307c5fa18c634be8
[2025-09-26 10:03:11.537] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(3b6ede70) 2(b4b2de78) 3(050d6435) 4(20
    c5c4af) 5(7ee82c13) 6(ac41eea0) 7(9b408c51) 8(d13592ce) 9(1
    d452cd0)
[2025-09-26 10:03:11.537] [debug][Producer][resilience] ops from v8
    -> v9:
[
  {
    "op": "add",
    "path": "/broker/collections/CAR_5",
    "value": {
      "algo": "SHA-256",
      "count": 0,
      "hash": "
          e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
          ",
      "lastSeen": "2025-09-26T07:03:11Z"
    }
  }
]
[2025-09-26 10:03:11.539] [info][Producer]Successfully registered
    prefix: /SNDS/SNDS_ID_scope_5
[2025-09-26 10:03:11.540] [info][Producer]Successfully registered
    prefix: /SNDS/is_alive_5
[2025-09-26 10:03:11.542] [info][Producer]Successfully registered
    prefix: /SNDS/CAR_1
[2025-09-26 10:03:11.542] [info][Producer]Successfully registered
    prefix: /SNDS/CAR_5
[2025-09-26 10:03:11.542] [info][Producer]Successfully registered
    prefix: /SNDS/CAR_5_registry
```

```
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS_1
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS_5
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS/is_alive_1
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS/car5
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS/CAR_1_registry
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS/SNDS_ID_scope_1
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS/car1
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS/state_5
[2025-09-26 10:03:11.543] [info][Producer]Successfully registered
    prefix: /SNDS/state_1
```

We see that the timeout timestamp for Node B **2025-09-26 10:03:11.493** and the full recovery timestamp is at **2025-09-26 10:03:11.543** which is a 50 ms difference.

Let's see if the data of B, which are now F's, are retrievable through the NDN network. Let's request data from E, the `@id=car1` which was originally published to B:



```
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# curl  -i -X GET http:/
/localhost:8080 --data-urlencode "id=car1"  -H "Accept: application/json"
HTTP/1.1 200 OK
Keep-Alive: timeout=5, max=100
Content-Length: 245
Content-Type: text/plain

{
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_5/newTypeSubscription:CAR_5&SNDS_ID_scope_4",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NEW\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
root@fotis-MS-7B86:/home/fotis/Github/SeEDS/src/minindn# 
```

Figure 17: Get the data of B, which are now F's, through E

Let's check F's logs, to see whether it actually provides the data:

Listing 19: F logs should now provide the data originally provided through B

```
[2025-09-26 10:22:24.170] [info][Producer]Received interest in
    function: onInterest, name: /SNDS/car1/temporalQuery%3Aid%3Dcar1
    %26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters%3D
[2025-09-26 10:22:24.170] [info][Producer]Parsed interest: Prefix:
    SNDS, Name: car1, Command: temporalQuery, Params: [id=car1, type
    =, date=, count=, timeframe=, filters=]
[2025-09-26 10:22:24.170] [debug][Producer]Created interest response:
     /SNDS/car1/temporalQuery%3Aid%3Dcar1%26type%3D%26date%3D%26
    count%3D%26timeframe%3D%26filters%3D in:
    process_temporal_query_interest
[2025-09-26 10:22:24.170] [debug][Producer]TemporalQuery -> id: car1,
     date: , type: , timeframe: , count: , filters:
[2025-09-26 10:22:24.170] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car1&type=&count=&date=&timeframe=&filters=
[2025-09-26 10:22:24.171] [info][Producer]Successfully queried /
    temporal-query
[2025-09-26 10:22:24.171] [debug][Producer]Sending data to the NDN...
```

Let's check E's logs, to see whether it actually retrieved the data:

```
[2025-09-26 10:26:55.977] [info][HTTPServer] Received GET request for
     / with body id=car1
[2025-09-26 10:26:55.977] [info][HTTPServer] Received GET request for
     / with target /
[2025-09-26 10:26:55.977] [info][HTTPServer] Received GET request
    from 127.0.0.1
Message: {id=car1}
Type: GET_BY_ID_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
id = car1
[2025-09-26 10:26:55.977] [debug][HTTPServer] Writing message from
    GET request to service queue...
Message: {id=car1}
Type: GET_BY_ID_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
id = car1

Message: BATCH
Type: GENERIC
Sender: WORKER
```

```
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_5/newTypeSubscription:CAR_5&SNDS_ID_scope_4",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NEW\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:
[2025-09-26 10:26:55.977] [info][HTTPServer] Read Message from queue:
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "/SNDS/CAR_5/newTypeSubscription:CAR_5&SNDS_ID_scope_4",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NEW\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:

[2025-09-26 10:26:55.977] [info][Worker]Received content: {id=car1}
[2025-09-26 10:26:55.977] [info][Worker]Received payload:
[2025-09-26 10:26:55.977] [info][Worker]Received type:
    GET_BY_ID_REQUEST
[2025-09-26 10:26:55.977] [debug][Worker]Server intra message...
[2025-09-26 10:26:55.977] [info][Worker]Inserted ID: car1 into set.
    Pushing regular getByID request for ID: car1
[2025-09-26 10:26:55.977] [info][Worker]Reading packet...
[2025-09-26 10:26:55.977] [debug][Consumer]Extracted content from
    message: /SNDS/car1/getByID:car1
```

https://doi.org/10.26219/heal.aueb.9822

```
[2025-09-26 10:26:55.977] [debug][Consumer]Extracted content length
    from message: 23
[2025-09-26 10:26:55.977] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_REQUEST
[2025-09-26 10:26:55.977] [debug][Consumer]Query.type:
[2025-09-26 10:26:55.977] [debug][Consumer]Query.id: car1
[2025-09-26 10:26:55.977] [info][Consumer]Created name /SNDS/car1/
    temporalQuery:id%3Dcar1&type%3D&date%3D&count%3D&timeframe%3D&
    filters%3D
[2025-09-26 10:26:55.977] [info][Consumer]Sending actual Interest /
    SNDS/car1/temporalQuery%3Aid%3Dcar1%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D?CanBePrefix&Lifetime=6000
[2025-09-26 10:26:55.977] [info][Consumer]Received Data with name: /
    SNDS/car1/temporalQuery%3Aid%3Dcar1%26type%3D%26date%3D%26count
    %3D%26timeframe%3D%26filters%3D in function onData
[2025-09-26 10:26:55.977] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car1",
      "id": "car1",
      "timestamp": {
        "$date": 1758870079983
      },
      "type": "GENERIC"
    }
  }
] in function onData
[2025-09-26 10:26:55.977] [info][Consumer]Parsed interest: Prefix:
    SNDS, Name: car1, Command: temporalQuery, Params: [id=car1, type
    =, date=, count=, timeframe=, filters=] in onData
[2025-09-26 10:26:55.977] [info][Consumer]Temporal Query Reponse...
[2025-09-26 10:26:55.977] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-09-26 10:26:55.977] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car1",
      "id": "car1",
      "timestamp": {
        "$date": 1758870079983
      },
      "type": "GENERIC"
    }
  }
]
[2025-09-26 10:26:55.977] [debug][Consumer]Sanitized payload to:
[
  {
    "": {
      "_id": "car1",
      "id": "car1",
      "timestamp": {
        "$date": 1758870079983
      },
      "type": "GENERIC"
    }
  }
]
```

Finally let’s also check the time of failure of Primary Node C and the recovery of Secondary Node G:

Listing 20: Timeout of Node C

```
[TIMEOUT] 2025-09-26 10:03:14.265 c: daemon exceeded timeout
```

Listing 21: Recovery of Node C through Node G

```
[2025-09-26 10:03:14.281] [warning][Producer][resilience-ipc]
    promotion to PRIMARY (reason=alive-missed, misses=1)
[2025-09-26 10:03:14.281] [info][Producer][resilience] mutate -> ver
    =6 hash=be7008804c441ae38b5c667e74638083
[2025-09-26 10:03:14.281] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(e9af7cdf) 2(721c5a31) 3(1743154a) 4(8
    a357a28) 5(a3baea0e) 6(be700880)
[2025-09-26 10:03:14.281] [debug][Producer][resilience] ops from v5
    -> v6:
[
  {
    "op": "replace",
    "path": "/resilience/reason",
    "value": "alive-missed"
  },
  {
    "op": "replace",
    "path": "/resilience/role",
    "value": "primary"
  },
  {
    "op": "replace",
    "path": "/resilience/since",
    "value": "2025-09-26T07:03:14Z"
```

```
 },
 {
   "op": "add",
   "path": "/resilience/misses",
   "value": 1
 }
]
[2025-09-26 10:03:14.281] [warning][Producer][resilience-ipc] stop
    requested; will be joined from main loop
[2025-09-26 10:03:14.320] [warning][Producer][resilience] applying
    promotion side-effects on main loop
[2025-09-26 10:03:14.320] [debug][Producer][overlay] merge: applying
    json_patch_against_base (ops=7)
[2025-09-26 10:03:14.320] [debug][Producer][resilience] assign -> ver
    =7 hash=75a517298fe7436e7751682d14c27144
[2025-09-26 10:03:14.320] [debug][Producer][overlay] SQUASH -> base
    version 6->7 hash be7008804c441ae38b5c667e74638083->75
    a517298fe7436e7751682d14c27144 overlay ver=16 hash=6
    f373a7c7c0c7f801932225586b6fa4f
[2025-09-26 10:03:14.320] [debug][Producer][overlay] CLEARED
[2025-09-26 10:03:14.320] [debug][Producer][promotion] rebuilt
    containers from store:
[2025-09-26 10:03:14.320] [debug][Producer]type_registry[CAR_2] =
    [car12 ]
[2025-09-26 10:03:14.320] [debug][Producer]type_registry[CAR_6] =
    [car16 ]
[2025-09-26 10:03:14.320] [debug][Producer]
    subscribed_nodes_for_type[CAR_2] = [SNDS_ID_scope_1 ]
[2025-09-26 10:03:14.320] [debug][Producer]
    subscribed_nodes_for_type[CAR_6] = [SNDS_ID_scope_5 ]
[2025-09-26 10:03:14.320] [debug][Producer]announcements = [/SNDS_2 /
    SNDS/CAR_6 /SNDS_6 /SNDS/car6 /SNDS/car2 /SNDS/SNDS_ID_scope_6 /
    SNDS/is_alive_6 /SNDS/CAR_6_registry /SNDS/state_6 /SNDS/CAR_2 /
    SNDS/state_2 /SNDS/CAR_2_registry /SNDS/SNDS_ID_scope_2 /SNDS/
    is_alive_2 ]
[2025-09-26 10:03:14.320] [info][Producer][promotion] summary: types
    =2 ids=2 subs(byId)=0 subs(byType)=2 announced=14
[2025-09-26 10:03:14.320] [info][Producer][promotion] (re)announcing
    /SNDS_2
[2025-09-26 10:03:14.320] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_6
[2025-09-26 10:03:14.320] [info][Producer][promotion] (re)announcing
    /SNDS_6
[2025-09-26 10:03:14.320] [info][Producer][promotion] (re)announcing
    /SNDS/car6
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/car2
```

```
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/SNDS_ID_scope_6
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/is_alive_6
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_6_registry
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/state_6
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_2
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/state_2
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/CAR_2_registry
[2025-09-26 10:03:14.321] [info][Producer][promotion] (re)announcing
    /SNDS/SNDS_ID_scope_2
[2025-09-26 10:03:14.322] [info][Producer][promotion] (re)announcing
    /SNDS/is_alive_2
[2025-09-26 10:03:14.322] [info][Producer]Getting from IP: 10.0.0.42
    and port: 10000
[2025-09-26 10:03:14.322] [debug][Producer]HTTP GET /state?type=CAR_2
[2025-09-26 10:03:14.324] [info][Producer][resilience] mutate -> ver
    =8 hash=8e464c8782cf6591d70ef4f5885d1acb
[2025-09-26 10:03:14.324] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(e9af7cdf) 2(721c5a31) 3(1743154a) 4(8
    a357a28) 5(a3baea0e) 6(be700880) 7(75a51729) 8(8e464c87)
[2025-09-26 10:03:14.324] [debug][Producer][resilience] ops from v7
    -> v8:
[
  {
   "op": "replace",
   "path": "/broker/collections/CAR_2/lastSeen",
   "value": "2025-09-26T07:03:14Z"
  }
]
[2025-09-26 10:03:14.324] [info][Producer]Getting from IP: 10.0.0.42
    and port: 10000
[2025-09-26 10:03:14.324] [debug][Producer]HTTP GET /state?type=CAR_6
[2025-09-26 10:03:14.325] [info][Producer][resilience] mutate -> ver
    =9 hash=8d00fd7121cc9637c59c2b2698b4f4fb
[2025-09-26 10:03:14.325] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(e9af7cdf) 2(721c5a31) 3(1743154a) 4(8
    a357a28) 5(a3baea0e) 6(be700880) 7(75a51729) 8(8e464c87) 9(8
    d00fd71)
[2025-09-26 10:03:14.325] [debug][Producer][resilience] ops from v8
    -> v9:
[
  {
    "op": "add",
    "path": "/broker/collections/CAR_6",
    "value": {
      "algo": "SHA-256",
      "count": 0,
      "hash": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
      "lastSeen": "2025-09-26T07:03:14Z"
    }
  }
]
[2025-09-26 10:03:14.327] [info][Producer]Successfully registered prefix: /SNDS_2
[2025-09-26 10:03:14.327] [info][Producer]Successfully registered prefix: /SNDS/CAR_6
[2025-09-26 10:03:14.327] [info][Producer]Successfully registered prefix: /SNDS_6
[2025-09-26 10:03:14.327] [info][Producer]Successfully registered prefix: /SNDS/car6
[2025-09-26 10:03:14.327] [info][Producer]Successfully registered prefix: /SNDS/car2
[2025-09-26 10:03:14.328] [info][Producer]Successfully registered prefix: /SNDS/SNDS_ID_scope_6
[2025-09-26 10:03:14.329] [info][Producer]Successfully registered prefix: /SNDS/is_alive_6
[2025-09-26 10:03:14.329] [info][Producer]Successfully registered prefix: /SNDS/CAR_6_registry
[2025-09-26 10:03:14.329] [info][Producer]Successfully registered prefix: /SNDS/state_6
[2025-09-26 10:03:14.329] [info][Producer]Successfully registered prefix: /SNDS/CAR_2
[2025-09-26 10:03:14.330] [info][Producer]Successfully registered prefix: /SNDS/state_2
[2025-09-26 10:03:14.330] [info][Producer]Successfully registered prefix: /SNDS/CAR_2_registry
[2025-09-26 10:03:14.330] [info][Producer]Successfully registered prefix: /SNDS/SNDS_ID_scope_2
[2025-09-26 10:03:14.330] [info][Producer]Successfully registered prefix: /SNDS/is_alive_2
```

C’s failure log is at: **2025-09-26 10:03:14.265** and G’s last data recovered is at: **2025-09-26 10:03:14.330**. This gives us a difference of 65 ms.

The resilience evaluation demonstrated that the SeEDS system effectively recovers from transient disruptions under 200 ms. Following recovery, the synchronization mechanism between the Primary and Secondary nodes is restored, ensuring that the latest state of the data maintained by the Primary nodes becomes accessible through their corresponding Secondary replicas. This confirms that SeEDS maintains data availability and consistency even in the presence of short-lived network or service interruptions, thereby validating its robustness and reliability in distributed environments.

A.3 NDN-testbed

We also deployed and tested the SeEDS prototype in the official NDN testbed. The experiment involved a distributed setup with three participating nodes that interconnect through the NDN across three different continents, including the University of Memphis (UMemphis) testbed node.

The prototype we tested included the *GET by TYPE* and *GET by ID* requests. The other requests have also been tested locally in MiniNDN and work as expected. Future work will include testing the other request types in the testbed network.

The experiment demonstrated end-to-end communication between the nodes, ensuring that Interests and Data packets could be exchanged reliably across wide-area NDN infrastructure. This deployment validated:

- The ability of SeEDS to operate in a multi-continental environment.
- Correct operation of the system in the global NDN testbed, beyond controlled lab conditions.
- Successful integration of the UMemphis node as a critical participant in the experiment.

A.3.1 Topology

The experiment was deployed on the NDN testbed using a three-node topology spanning different continents in NDN hops. The participating nodes were:

- `MMLab1` – NDN testbed node located in Athens University of Economics and was configured as the Producer node, responsible for publishing and advertising data into the NDN network.
- `MMLab2` – NDN testbed node located again in Athens University of Economics and Business, issuing Interests to retrieve data from the producer.
- `UMemphis` – NDN testbed node located in North America (University of Memphis), was configured as a Consumer node, retrieving the same data across transcontinental paths.

The full topology can be found in: https://play.ndn.today/?testbed=1. The topology shown in Figure 18 has been simplified to contain a max of 3 hops distance from the MMLab NDN nodes to the UMemphis NDN node. We can also include all distances, but these are the most likely paths a NDN packet is going to take. Note we also keep the consumer node MMLab2 connections with consumer UMemphis, because the Memphis node might be able to retrieve data from the cache of MMLab2.

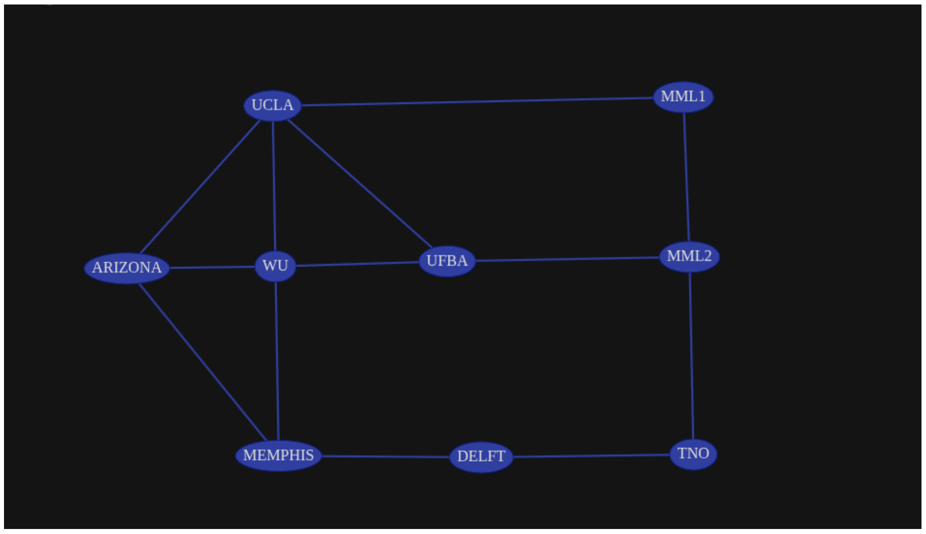


Figure 18: Simplified 3-hop NDN testbed topology showing the main paths between the producer (MMLab1) and the consumers (MMLab2, UMemphis), with potential intermediate relay/caching nodes.

The nodes shown in the figure are:

- UFBA: Federal University of Bahia in Brazil - South America.
- UCLA: University of California in Los Angeles - United States - North America.
- WU: Washington University in St.Louis - United States - North America.
- TNO: Netherlands Organisation for Applied Scientific Research - Europe
- DELFT: Delft University of Technology in Netherlands - Europe.
- ARIZONA: University of Arizona - North America.

Based on the graph and the routing tables at least two continents are involved in the communication between Memphis and the MMLab nodes, with

https://doi.org/10.26219/heal.aueb.9822

the possibility that a third continent (South America) is going to be involved in the communication. This topology ensured intercontinental communication paths, with each node positioned in a different geographic region. If the data between the nodes can be retrieved, the intercontinental communication is possible. Note that the MiniNDN links are set up like this:

Listing 22: MiniNDN config file for the testbed nodes

```
[nodes]
UCLA: _network=/world router=/UCLA.Router/ position=-40.5,-28.7,0
TNO: _network=/world router=/TNO.Router/ position=-0.1,4.8,0
MEMPHIS: _network=/world router=/MEMPHIS.Router/ position=-39.9,5,0
UFBA: _network=/world router=/UFBA.Router/ position=-23.7,-13.7,0
MML2: _network=/world router=/MML2.Router/ position=-0.5,-14.2,0
MML1: _network=/world router=/MML1.Router/ position=-1,-29.5,0
ARIZONA: _network=/world router=/ARIZONA.Router/ position=-54.5,-13,0
DELFT: _network=/world router=/DELFT.Router/ position=-20.5,5.1,0
WU: _network=/world router=/WU.Router/ position=-40.2,-13.2,0
[switches]
[links]
UCLA:MML1 delay=10ms
UCLA:ARIZONA delay=10ms
UCLA:WU delay=10ms
UCLA:UFBA delay=10ms
TNO:DELFT delay=10ms
TNO:MML2 delay=10ms
MEMPHIS:ARIZONA delay=10ms
MEMPHIS:WU delay=10ms
MEMPHIS:DELFT delay=10ms
UFBA:WU delay=10ms
UFBA:MML2 delay=10ms
MML2:MML1 delay=10ms
ARIZONA:WU delay=10ms
```

A.3.2 GET by ID

In this experiment, the entity `@id=car001` is published directly by the Producer node (`MMLab1`), thereby registering it within the Producer's local namespace and enabling retrieval by other nodes across the network.

Let's publish `@id=car001` to the network:

Listing 23: Publish `@id=car001` to the NDN network through the Producer Node

```
scn4ndn@seeds8:~$ curl -v -i -X POST http://127.0.0.1:8080 -d "id=car
    001"
Note: Unnecessary use of -X or --request, POST is already inferred.
*Trying 127.0.0.1:8080...
```

https://doi.org/10.26219/heal.aueb.9822

```
*Connected to 127.0.0.1 (127.0.0.1) port 8080
>POST /HTTP/1.1
>Host: 127.0.0.1:8080
>User-Agent: curl/8.5.0
>Accept: */*
>Content-Length: 9
>Content-Type: application/x-www-form-urlencoded
>
<HTTP/1.1 200OK
HTTP/1.1 200OK
<Keep-Alive: timeout=5, max=100
Keep-Alive: timeout=5, max=100
<Content-Length: 189
Content-Length: 189
<Content-Type: text/plain
Content-Type: text/plain

<
{
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "raw": "ACK_ID_ADVERTISED\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
*Connection #0 to host 127.0.0.1 left intact
```

Let's check the logs from within the SeEDS Service of the Producer:

Listing 24: SeEDS Service logs of the Producer after publishing `@id=car001`

```
[2025-10-01 13:49:06.864] [info][HTTPServer] Received POST request
    for / with body id=car001
[2025-10-01 13:49:06.864] [info][HTTPServer] Received POST request
    from 127.0.0.1
Message: {id=car001}
Type: BY_ID_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car001
```

```
[2025-10-01 13:49:06.864] [debug][HTTPServer] Writing message from
    POST request to service queue...
Message: {id=car001}
Type: BY_ID_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car001

[2025-10-01 13:49:06.864] [debug][DigitalTwin]Extracted content from
    message {id=car001}
[2025-10-01 13:49:06.864] [debug][DigitalTwin]Extracted content
    length from message 11
[2025-10-01 13:49:06.864] [debug][DigitalTwin]Extracted type from
    message: BY_ID_ADVERTISEMENT
[2025-10-01 13:49:06.864] [info][DigitalTwin]Received by ID get
    request...
[2025-10-01 13:49:06.864] [debug][DigitalTwin]Pushed ID car001 to the
     announced IDs vector...
[2025-10-01 13:49:06.864] [debug][DigitalTwin]Extracted id Scope from
     message SNDS_ID_scope_2
Message: /ndn/gr/edu/mmlab1/aueb/thomasi/car001/postID:car001
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:49:06.864] [info][DigitalTwin]Pushing byID
    advertisement to the Producer...
Message: /ndn/gr/edu/mmlab1/aueb/thomasi/car001/postID:car001
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:49:06.864] [debug][DigitalTwin]Sleeping 2000 ms before
     sending scope advertisement...
Message: /ndn/gr/edu/mmlab1/aueb/thomasi/car001/postID:car001
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:49:06.893] [info][Producer]Got new ID advertisement
    request...
```

```
Message: /ndn/gr/edu/mmlab1/aueb/thomasi/car001/postID:car001
Type: BY_ID_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:49:06.893] [debug][Producer]Parsed interest Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: car001, Command: postID,
    Params: [car001] from string
[2025-10-01 13:49:06.893] [info][Producer]Announcing name: /ndn/gr/
    edu/mmlab1/aueb/thomasi/car001
[2025-10-01 13:49:06.894] [info][Producer]Publishing to broker...
[2025-10-01 13:49:06.894] [warning][Producer]_publish_to_broker: '
    type' (string) not provided...
[2025-10-01 13:49:06.894] [info][Producer]POST /add -> {"id":"car001"
    }
[2025-10-01 13:49:07.082] [info][Producer]Published to broker OK: {"
    status": "success", "inserted_count_versioned": 1, "
    upserted_or_modified_count": 1}

[2025-10-01 13:49:07.082] [debug][Producer]Appending body to broker
    journal: {"id":"car001"}
[2025-10-01 13:49:07.082] [info][Producer][resilience] mutate -> ver
    =8 hash=a96596e77b8ca3a9df6378a6ef9a4362
[2025-10-01 13:49:07.082] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(5b784e73) 2(851bb2ce) 3(f2dc8068) 4(1
    dbec4b9) 5(92323da8) 6(a8c74cc9) 7(11cc7dd5) 8(a96596e7)
[2025-10-01 13:49:07.082] [debug][Producer][resilience] ops from v7
    -> v8:
[
  {
    "op": "add",
    "path": "/broker/journal",
    "value": [
      {
        "id": "car001",
        "payload": {
          "id": "car001"
        },
        "ts": "2025-10-01T13:49:07Z",
        "type": "GENERIC"
      }
    ]
  }
]
[2025-10-01 13:49:07.083] [info][Worker]Received content:
```

```
[2025-10-01 13:49:07.083] [info][Worker]Received payload:
    ACK_ID_ADVERTISED
[2025-10-01 13:49:07.083] [info][Worker]Received type:
    BY_ID_ADVERTISEMENT
[2025-10-01 13:49:07.083] [info][Worker]Received ACK Message with
    name: payload: ACK_ID_ADVERTISED...
Message:
Type: CONTROL_MESSAGE
Sender: WORKER
Payload: ACK_ID_ADVERTISED

HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:49:07.083] [debug][Worker]Writing message to service
    queue:
Message:
Type: CONTROL_MESSAGE
Sender: WORKER
Payload: ACK_ID_ADVERTISED

HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:49:07.083] [info][Worker]Reading packet...
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "raw": "ACK_ID_ADVERTISED\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:49:07.083] [info][HTTPServer] Read Message from queue:
Message: BATCH
Type: GENERIC
Sender: WORKER
```

```
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "raw": "ACK_ID_ADVERTISED\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:49:07.087] [info][Producer]Successfully registered
    prefix: /ndn/gr/edu/mmlab1/aueb/thomasi/car001
```

Let's try to GET the `@id=car001` from the SeEDS Service of Consumer `MMLab2`:

Listing 25: GET by ID for `@id=car001` through Node `MMlab2`

```
scn4ndn@seeds7:~$ curl -i -X GET http://localhost:8080/?"id=car001"
HTTP/1.1 200OK
Keep-Alive: timeout=5, max=100
Content-Length: 332
Content-Type: text/plain

{
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1759326546992,
            "id": "car001",
            "type": "GENERIC"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}
```

We got the ID, but let's make sure we actually got it from the Producer Node. Let's check the logs of the Producer:

Listing 26: Producer logs after receiving GET by ID from MMLab2

```
[2025-10-01 13:50:41.971] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/
    car001/temporalQuery%3Aid%3Dcar001%26type%3D%26date%3D%26count%3
    D%26timeframe%3D%26filters%3D
[2025-10-01 13:50:41.971] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: car001, Command:
    temporalQuery, Params: [id=car001, type=, date=, count=,
    timeframe=, filters=]
[2025-10-01 13:50:41.971] [debug][Producer]Created interest response:
     /ndn/gr/edu/mmlab1/aueb/thomasi/car001/temporalQuery%3Aid%3
    Dcar001%26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters
    %3D in: process_temporal_query_interest
[2025-10-01 13:50:41.971] [debug][Producer]TemporalQuery -> id:
    car001, date: , type: , timeframe: , count: , filters:
[2025-10-01 13:50:41.971] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car001&type=&count=&date=&timeframe=&filters=
[2025-10-01 13:50:41.972] [info][Producer]Successfully queried /
    temporal-query
[2025-10-01 13:50:41.972] [debug][Producer]Sending data to the NDN...
```

Let's also check the logs of the MMLab2 SeEDS Service:

Listing 27: After GET by ID of @id=car001 and response from MMLab1-Producer

```
[2025-10-01 13:50:41.969] [info][HTTPServer] Received GET request for
     / with body
[2025-10-01 13:50:41.969] [info][HTTPServer] Received GET request for
     / with target /?id=car001
[2025-10-01 13:50:41.969] [info][HTTPServer] Received GET request
    from 127.0.0.1
Message: {id=car001}
Type: GET_BY_ID_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
id = car001
[2025-10-01 13:50:41.969] [debug][HTTPServer] Writing message from
    GET request to service queue...
Message: {id=car001}
```

```
Type: GET_BY_ID_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
id = car001

[2025-10-01 13:50:41.969] [info][Worker]Received content: {id=car001}
[2025-10-01 13:50:41.969] [info][Worker]Received payload:
[2025-10-01 13:50:41.969] [info][Worker]Received type:
    GET_BY_ID_REQUEST
[2025-10-01 13:50:41.969] [debug][Worker]Server intra message...
[2025-10-01 13:50:41.969] [info][Worker]Inserted ID: car001 into set.
     Pushing regular getByID request for ID: car001
[2025-10-01 13:50:41.969] [info][Worker]Reading packet...
[2025-10-01 13:50:41.969] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/car001/getByID:car001
[2025-10-01 13:50:41.969] [debug][Consumer]Extracted content length
    from message: 53
[2025-10-01 13:50:41.969] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_REQUEST
[2025-10-01 13:50:41.969] [debug][Consumer]Query.type:
[2025-10-01 13:50:41.969] [debug][Consumer]Query.id: car001
[2025-10-01 13:50:41.969] [info][Consumer]Created name /ndn/gr/edu/
    mmlab1/aueb/thomasi/car001/temporalQuery:id%3Dcar001&type%3D&
    date%3D&count%3D&timeframe%3D&filters%3D
[2025-10-01 13:50:41.969] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/car001/temporalQuery%3Aid%3
    Dcar001%26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters
    %3D?CanBePrefix&MustBeFresh&Lifetime=200
[2025-10-01 13:50:41.977] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/car001/temporalQuery%3Aid%3
    Dcar001%26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters
    %3D in function onData
[2025-10-01 13:50:41.977] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car001",
      "id": "car001",
      "timestamp": {
        "$date": 1759326546992
      },
      "type": "GENERIC"
    }
  }
] in function onData
[2025-10-01 13:50:41.977] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: car001, Command:
    temporalQuery, Params: [id=car001, type=, date=, count=,
    timeframe=, filters=] in onData
[2025-10-01 13:50:41.977] [info][Consumer]Temporal Query Reponse...
[2025-10-01 13:50:41.977] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-10-01 13:50:41.977] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car001",
      "id": "car001",
      "timestamp": {
        "$date": 1759326546992
      },
      "type": "GENERIC"
    }
  }
]
[2025-10-01 13:50:41.977] [debug][Consumer]Sanitized payload to:
[
  {
    "": {
      "_id": "car001",
      "id": "car001",
      "timestamp": {
        "$date": 1759326546992
      },
      "type": "GENERIC"
    }
  }
]
[2025-10-01 13:50:41.977] [debug][Consumer]Direct mode (single id in
    bucket).
[2025-10-01 13:50:41.977] [debug][Consumer]Got ID and type in direct:
     id=car001,type=
[2025-10-01 13:50:41.978] [info][Worker]Received content: /?id=car001
    &type=&date=&metafile=false&parentQueryHash=&lastInterest=true&
    mode=direct
[2025-10-01 13:50:41.978] [info][Worker]Received payload: [
  {
    "": {
      "_id": "car001",
      "id": "car001",
      "timestamp": {
```

https://doi.org/10.26219/heal.aueb.9822

```
        "$date": 1759326546992
      },
      "type": "GENERIC"
    }
  }
]
[2025-10-01 13:50:41.978] [info][Worker]Received type:
    VERSION_RESPONSE
[2025-10-01 13:50:41.978] [debug][Worker]Consumer intra message...
[2025-10-01 13:50:41.978] [info][Worker]Direct mode...
[2025-10-01 13:50:41.978] [info][Consumer]Reading packet...
[2025-10-01 13:50:41.978] [debug][Worker]Cleared
    pending_get_by_id_requests for 'car001': erased=1
[2025-10-01 13:50:41.978] [debug][Worker]Cleared
    pending_get_by_type_requests for 'GENERIC': erased=0
[2025-10-01 13:50:41.978] [debug][Worker]Direct mode: no filters
    discovered; returning full item
[2025-10-01 13:50:41.978] [info][Worker]Reading packet...
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1759326546992,
            "id": "car001",
            "type": "GENERIC"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:50:41.978] [info][HTTPServer] Read Message from queue:
Message: BATCH
Type: GENERIC
```

```
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1759326546992,
            "id": "car001",
            "type": "GENERIC"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:
```

Let's also try to get it from the UMemphis Node:

Listing 28: HTTP Request for @id=car001

```
scn4ndn@seeds23:~$ curl -i -X GET http://localhost:8080/?"id=car001"
HTTP/1.1 200OK
Keep-Alive: timeout=5, max=100
Content-Length: 332
Content-Type: text/plain

{
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1759326546992,
            "id": "car001",
            "type": "GENERIC"
          }
        ],
        "mode": "direct"
```

https://doi.org/10.26219/heal.aueb.9822

```
    },
    "sender": 6,
    "type": "TEMPORAL_QUERY_RESPONSE"
  }
 ]
}
scn4ndn@seeds23:~$
```

Let's check the Producer's SeEDS Service logs:

Listing 29: Logs after receiving GET by ID request from node UMemphis

```
[2025-10-01 13:53:00.664] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/
    car001/temporalQuery%3Aid%3Dcar001%26type%3D%26date%3D%26count%3
    D%26timeframe%3D%26filters%3D
[2025-10-01 13:53:00.664] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: car001, Command:
    temporalQuery, Params: [id=car001, type=, date=, count=,
    timeframe=, filters=]
[2025-10-01 13:53:00.664] [debug][Producer]Created interest response:
     /ndn/gr/edu/mmlab1/aueb/thomasi/car001/temporalQuery%3Aid%3
    Dcar001%26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters
    %3D in: process_temporal_query_interest
[2025-10-01 13:53:00.664] [debug][Producer]TemporalQuery -> id:
    car001, date: , type: , timeframe: , count: , filters:
[2025-10-01 13:53:00.664] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car001&type=&count=&date=&timeframe=&filters=
[2025-10-01 13:53:00.668] [info][Producer]Successfully queried /
    temporal-query
[2025-10-01 13:53:00.669] [debug][Producer]Sending data to the NDN...
```

Let's also check the SeEDS Service logs of the UMemphis Node:

Listing 30: GET by ID of @id=car001 through the UMemphis

```
[2025-10-01 13:53:00.501] [info][HTTPServer] Received GET request for
     / with body
[2025-10-01 13:53:00.501] [info][HTTPServer] Received GET request for
     / with target /?id=car001
[2025-10-01 13:53:00.501] [info][HTTPServer] Received GET request
    from 127.0.0.1
Message: {id=car001}
Type: GET_BY_ID_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
id = car001
```

```
[2025-10-01 13:53:00.501] [debug][HTTPServer] Writing message from
    GET request to service queue...
Message: {id=car001}
Type: GET_BY_ID_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
id = car001

[2025-10-01 13:53:00.501] [info][Worker]Received content: {id=car001}
[2025-10-01 13:53:00.501] [info][Worker]Received payload:
[2025-10-01 13:53:00.501] [info][Worker]Received type:
    GET_BY_ID_REQUEST
[2025-10-01 13:53:00.501] [debug][Worker]Server intra message...
[2025-10-01 13:53:00.501] [info][Worker]Inserted ID: car001 into set.
     Pushing regular getByID request for ID: car001
[2025-10-01 13:53:00.501] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/car001/getByID:car001
[2025-10-01 13:53:00.501] [info][Worker]Reading packet...
[2025-10-01 13:53:00.501] [debug][Consumer]Extracted content length
    from message: 53
[2025-10-01 13:53:00.501] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_REQUEST
[2025-10-01 13:53:00.501] [debug][Consumer]Query.type:
[2025-10-01 13:53:00.501] [debug][Consumer]Query.id: car001
[2025-10-01 13:53:00.501] [info][Consumer]Created name /ndn/gr/edu/
    mmlab1/aueb/thomasi/car001/temporalQuery:id%3Dcar001&type%3D&
    date%3D&count%3D&timeframe%3D&filters%3D
[2025-10-01 13:53:00.501] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/car001/temporalQuery%3Aid%3
    Dcar001%26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters
    %3D?CanBePrefix&MustBeFresh&Lifetime=6000
[2025-10-01 13:53:00.833] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/car001/temporalQuery%3Aid%3
    Dcar001%26type%3D%26date%3D%26count%3D%26timeframe%3D%26filters
    %3D in function onData
[2025-10-01 13:53:00.833] [info][Consumer]Data packet payload: [
 {
  "": {
   "_id": "car001",
   "id": "car001",
   "timestamp": {
     "$date": 1759326546992
   },
   "type": "GENERIC"
  }
 }
] in function onData
[2025-10-01 13:53:00.834] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: car001, Command:
    temporalQuery, Params: [id=car001, type=, date=, count=,
    timeframe=, filters=] in onData
[2025-10-01 13:53:00.834] [info][Consumer]Temporal Query Reponse...
[2025-10-01 13:53:00.834] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-10-01 13:53:00.834] [debug][Consumer]Candidate JSON slice:
[
 {
  "": {
   "_id": "car001",
   "id": "car001",
   "timestamp": {
    "$date": 1759326546992
   },
   "type": "GENERIC"
  }
 }
]
[2025-10-01 13:53:00.834] [debug][Consumer]Sanitized payload to:
[
 {
  "": {
   "_id": "car001",
   "id": "car001",
   "timestamp": {
    "$date": 1759326546992
   },
   "type": "GENERIC"
  }
 }
]
[2025-10-01 13:53:00.834] [debug][Consumer]Direct mode (single id in
    bucket).
[2025-10-01 13:53:00.834] [debug][Consumer]Got ID and type in direct:
     id=car001,type=
[2025-10-01 13:53:00.834] [info][Consumer]Reading packet...
[2025-10-01 13:53:00.834] [info][Worker]Received content: /?id=car001
    &type=&date=&metafile=false&parentQueryHash=&lastInterest=true&
    mode=direct
[2025-10-01 13:53:00.834] [info][Worker]Received payload: [
 {
    "": {
      "_id": "car001",
      "id": "car001",
      "timestamp": {
        "$date": 1759326546992
      },
      "type": "GENERIC"
    }
  }
]
[2025-10-01 13:53:00.834] [info][Worker]Received type:
    VERSION_RESPONSE
[2025-10-01 13:53:00.834] [debug][Worker]Consumer intra message...
[2025-10-01 13:53:00.834] [info][Worker]Direct mode...
[2025-10-01 13:53:00.834] [debug][Worker]Cleared
    pending_get_by_id_requests for 'car001': erased=1
[2025-10-01 13:53:00.835] [debug][Worker]Cleared
    pending_get_by_type_requests for 'GENERIC': erased=0
[2025-10-01 13:53:00.835] [debug][Worker]Direct mode: no filters
    discovered; returning full item
[2025-10-01 13:53:00.835] [info][Worker]Reading packet...
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1759326546992,
            "id": "car001",
            "type": "GENERIC"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:
```

```
[2025-10-01 13:53:00.835] [info][HTTPServer] Read Message from queue:
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1759326546992,
            "id": "car001",
            "type": "GENERIC"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:
```

A.3.3 GET by TYPE

In this experiment, the Producer node MMLab1 populates its internal @type=CAR registry by inserting both @id=car001 and @id=car002 entries to it. This publications ensure that the corresponding @type=CAR scope is properly announced within the NDN network, allowing type-based discovery and retrieval through SeEDS.

Furthermore, to validate remote publication, the Consumer node MMLab2 publishes @id=car003 to the Producer's CAR registry. This action demonstrates that external nodes can contribute data to the Producer's registry using the same interface, enabling a distributed and collaborative data publication process. As a result, all three entities – @id=car001, @id=car002, and @id=car003 – become accessible within the NDN network through the shared @type=CAR registry scope.

Let's publish @ids=car001,car002 to the CAR registry of the Producer:

Listing 31: Publish @ids=car001,car002 from within the Producer, to the

https://doi.org/10.26219/heal.aueb.9822

Producer's registry

```
scn4ndn@seeds8:~$ curl -v -i -X POST http://127.0.0.1:8080 -d "id=car
    002&type=CAR"
Note: Unnecessary use of -X or --request, POST is already inferred.
*Trying 127.0.0.1:8080...
*Connected to 127.0.0.1 (127.0.0.1) port 8080
>POST /HTTP/1.1
>Host: 127.0.0.1:8080
>User-Agent: curl/8.5.0
>Accept: */*
>Content-Length: 18
>Content-Type: application/x-www-form-urlencoded
>
<HTTP/1.1 200OK
HTTP/1.1 200OK
<Keep-Alive: timeout=5, max=100
Keep-Alive: timeout=5, max=100
<Content-Length: 260
Content-Length: 260
<Content-Type: text/plain
Content-Type: text/plain

<
{
  "batch_size": 1,
  "messages": [
    {
      "name": "/ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
          byIDAndTypeAdvertisement:car002&CAR",
      "payload": {
        "raw": "ACK_ID_REGISTERED\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
*Connection #0 to host 127.0.0.1 left intact
scn4ndn@seeds8:~$ curl -v -i -X POST http://127.0.0.1:8080 -d "id=car
    001&type=CAR"
Note: Unnecessary use of -X or --request, POST is already inferred.
*Trying 127.0.0.1:8080...
*Connected to 127.0.0.1 (127.0.0.1) port 8080
>POST /HTTP/1.1
>Host: 127.0.0.1:8080
>User-Agent: curl/8.5.0
>Accept: */*
```

https://doi.org/10.26219/heal.aueb.9822

```
>Content-Length: 18
>Content-Type: application/x-www-form-urlencoded
>
<HTTP/1.1 200OK
HTTP/1.1 200OK
<Keep-Alive: timeout=5, max=100
Keep-Alive: timeout=5, max=100
<Content-Length: 283
Content-Length: 283
<Content-Type: text/plain
Content-Type: text/plain

<
{
  "batch_size": 1,
  "messages": [
    {
      "name": "/ndn/gr/edu/mmlab1/aueb/thomasi/CAR/scopeNotification:
          car002&CAR&SNDS_ID_scope_0",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NO_SUBSCRIBERS\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
*Connection #0 to host 127.0.0.1 left intact
scn4ndn@seeds8:~$
```

Let's check the Producer's SeEDS Service logs for `@ids=car001,car002`:

Listing 32: Producer SeEDS Service logs after publishing `@ids=car001,car002` to the `@type=CAR` registry

```
[2025-10-01 13:37:32.755] [info][HTTPServer] Received POST request
    for / with body id=car001&type=CAR
[2025-10-01 13:37:32.755] [info][HTTPServer] Received POST request
    from 127.0.0.1
Message: {id=car001,type=CAR}
Type: BY_ID_AND_TYPE_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car001
type = CAR
[2025-10-01 13:37:32.755] [debug][HTTPServer] Writing message from
    POST request to service queue...
```

```
Message: {id=car001,type=CAR}
Type: BY_ID_AND_TYPE_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car001
type = CAR

[2025-10-01 13:37:32.755] [debug][DigitalTwin]Extracted content from
    message {id=car001,type=CAR}
[2025-10-01 13:37:32.755] [debug][DigitalTwin]Extracted content
    length from message 20
[2025-10-01 13:37:32.755] [debug][DigitalTwin]Extracted type from
    message: BY_ID_AND_TYPE_ADVERTISEMENT
[2025-10-01 13:37:32.755] [info][DigitalTwin]Received by Type get
    request...
[2025-10-01 13:37:32.755] [debug][DigitalTwin]The Type is the
    following: CAR
[2025-10-01 13:37:32.755] [debug][DigitalTwin]The ID is the
    following: car001
Message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    byIDAndTypeAdvertisement:car001&CAR
Type: BY_ID_AND_TYPE_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:37:32.755] [info][DigitalTwin]Pushing byType
    advertisement to the Consumer...
Message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    byIDAndTypeAdvertisement:car001&CAR
Type: BY_ID_AND_TYPE_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:37:32.755] [debug][DigitalTwin]Sleeping 2000 ms before
     sending scope advertisement...
[2025-10-01 13:37:32.755] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    byIDAndTypeAdvertisement:car001&CAR
[2025-10-01 13:37:32.755] [debug][Consumer]Extracted content length
    from message: 71
[2025-10-01 13:37:32.755] [debug][Consumer]Extracted message type
    from message: BY_ID_AND_TYPE_ADVERTISEMENT
```

https://doi.org/10.26219/heal.aueb.9822

```
[2025-10-01 13:37:32.755] [debug][Consumer]Created name: /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR/byIDAndTypeAdvertisement:car001&CAR in
    func: express_by_id_and_type_advertisement_interest
[2025-10-01 13:37:32.755] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/byIDAndTypeAdvertisement%3
    Acar001%26CAR?CanBePrefix&MustBeFresh&Lifetime=6000
[2025-10-01 13:37:32.756] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    byIDAndTypeAdvertisement%3Acar001%26CAR
[2025-10-01 13:37:32.756] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    byIDAndTypeAdvertisement, Params: [car001, CAR]
[2025-10-01 13:37:32.756] [info][Producer]Processing Type
    Advertisement
[2025-10-01 13:37:32.756] [info][Producer]Type advertisement: name(
    registry) = CAR, ID = car001
[2025-10-01 13:37:32.756] [info][Producer][event] type_add: CAR ->
    car001
[2025-10-01 13:37:32.756] [info][Producer][resilience] mutate -> ver
    =8 hash=0c1d2fdbfecac4ed4a5374ed83f8a7d8
[2025-10-01 13:37:32.756] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(630cd269) 2(2bcd1ae9) 3(c9dc1836) 4(9
    a5a300d) 5(c872b4e2) 6(f1eb6fbc) 7(b5972495) 8(0c1d2fdb)
[2025-10-01 13:37:32.756] [debug][Producer][resilience] ops from v7
    -> v8:
[
  {
    "op": "add",
    "path": "/types/CAR",
    "value": {
      "ids": {
        "car001": true
      }
    }
  }
]
[2025-10-01 13:37:32.756] [info][Producer]Getting from IP: 127.0.0.1
    and port: 10000
[2025-10-01 13:37:32.756] [debug][Producer]HTTP GET /state?type=CAR
[2025-10-01 13:37:32.760] [info][Producer][resilience] mutate -> ver
    =9 hash=231555f1734d0c307e5320479fcc0c9f
[2025-10-01 13:37:32.760] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(630cd269) 2(2bcd1ae9) 3(c9dc1836) 4(9
    a5a300d) 5(c872b4e2) 6(f1eb6fbc) 7(b5972495) 8(0c1d2fdb)
    9(231555f1)
[2025-10-01 13:37:32.760] [debug][Producer][resilience] ops from v8
    -> v9:
```

```
[
  {
    "op": "add",
    "path": "/broker/collections/CAR",
    "value": {
      "algo": "SHA-256",
      "count": 3,
      "hash": "
          b66cd183738b81929b9632a47631dace78dd98715418434d4f92be99a6ec75c1
          ",
      "lastSeen": "2025-10-01T13:37:32Z"
    }
  }
]
[2025-10-01 13:37:32.761] [info][Producer][debug][event:type_add CAR
     ->car001] summary -> types=1 ids=1 subs(byId)=0 subs(byType)=0
     announced=6
[2025-10-01 13:37:32.761] [debug][Producer][debug][event:type_add CAR
     ->car001] type_registry:
[2025-10-01 13:37:32.761] [debug][Producer][CAR] car001
[2025-10-01 13:37:32.761] [debug][Producer][debug][event:type_add CAR
     ->car001] subscribed_nodes_for_id:
[2025-10-01 13:37:32.761] [debug][Producer][debug][event:type_add CAR
     ->car001] subscribed_nodes_for_type:
[2025-10-01 13:37:32.761] [debug][Producer][debug][event:type_add CAR
     ->car001] announcements: /ndn/gr/edu/mmlab1/aueb/thomasi/
     CAR_registry /ndn/gr/edu/mmlab1/aueb/thomasi/CAR /ndn/gr/edu/
     mmlab1/aueb/thomasi/state_0 /ndn/gr/edu/mmlab1/aueb/thomasi/
     is_alive_0 /ndn/gr/edu/mmlab1/aueb/thomasi/SNDS_ID_scope_0 /ndn/
     gr/edu/mmlab1/aueb/thomasi_0
[2025-10-01 13:37:32.761] [info][Producer][debug][event:type_add CAR
     ->car001] broker.journal: entries=0 cursor=0 last_fp=
[2025-10-01 13:37:32.761] [debug][Producer][overlay][event:type_add
     CAR->car001] ver=0 hash=d41d8cd98f00b204e9800998ecf8427e keys=0
[2025-10-01 13:37:32.761] [info][Producer]Registered new ID car001
     for type CAR. Sending CONTROL.
[2025-10-01 13:37:32.761] [debug][Producer]Sending data to the NDN...
[2025-10-01 13:37:32.764] [info][Consumer]Received Data with name: /
     ndn/gr/edu/mmlab1/aueb/thomasi/CAR/byIDAndTypeAdvertisement%3
     Acar001%26CAR in function onData
[2025-10-01 13:37:32.764] [info][Consumer]Data packet payload:
     ACK_ID_REGISTERED in function onData
[2025-10-01 13:37:32.764] [info][Consumer]Parsed interest: Prefix:
     ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
     byIDAndTypeAdvertisement, Params: [car001, CAR] in onData
[2025-10-01 13:37:32.764] [info][Consumer]Got ACK ACK_ID_REGISTERED.
     Skipping...
```

https://doi.org/10.26219/heal.aueb.9822

```
[2025-10-01 13:37:32.764] [info][Worker]Received content: /ndn/gr/edu
    /mmlab1/aueb/thomasi/CAR/byIDAndTypeAdvertisement:car001&CAR
[2025-10-01 13:37:32.764] [info][Worker]Received payload:
    ACK_ID_REGISTERED
[2025-10-01 13:37:32.765] [info][Worker]Received type:
    GET_BY_ID_FOR_TYPE_RESPONSE

[2025-10-01 13:38:23.395] [info][HTTPServer] Received POST request
    for / with body id=car002&type=CAR
[2025-10-01 13:38:23.395] [info][HTTPServer] Received POST request
    from 127.0.0.1
Message: {id=car002,type=CAR}
Type: BY_ID_AND_TYPE_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car002
type = CAR
[2025-10-01 13:38:23.395] [debug][HTTPServer] Writing message from
    POST request to service queue...
Message: {id=car002,type=CAR}
Type: BY_ID_AND_TYPE_ADVERTISEMENT
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_POST
Params:
id = car002
type = CAR

Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "/ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
          scopeNotification:car001&CAR&SNDS_ID_scope_0",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NO_SUBSCRIBERS\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
```

```

HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:38:23.395] [debug][DigitalTwin]Extracted content from
    message {id=car002,type=CAR}
[2025-10-01 13:38:23.395] [debug][DigitalTwin]Extracted content
    length from message 20
[2025-10-01 13:38:23.395] [debug][DigitalTwin]Extracted type from
    message: BY_ID_AND_TYPE_ADVERTISEMENT
[2025-10-01 13:38:23.395] [info][DigitalTwin]Received by Type get
    request...
[2025-10-01 13:38:23.395] [info][HTTPServer] Read Message from queue:
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "/ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
          scopeNotification:car001&CAR&SNDS_ID_scope_0",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NO_SUBSCRIBERS\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:38:23.395] [debug][DigitalTwin]The Type is the
    following: CAR
[2025-10-01 13:38:23.395] [debug][DigitalTwin]The ID is the
    following: car002
Message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    byIDAndTypeAdvertisement:car002&CAR
Type: BY_ID_AND_TYPE_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:38:23.395] [info][DigitalTwin]Pushing byType
    advertisement to the Consumer...
Message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
```

https://doi.org/10.26219/heal.aueb.9822

```
    byIDAndTypeAdvertisement:car002&CAR
Type: BY_ID_AND_TYPE_ADVERTISEMENT
Sender: DIGITAL_TWIN
Payload:
HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:38:23.396] [debug][DigitalTwin]Sleeping 2000 ms before
     sending scope advertisement...
[2025-10-01 13:38:23.396] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    byIDAndTypeAdvertisement:car002&CAR
[2025-10-01 13:38:23.396] [debug][Consumer]Extracted content length
    from message: 71
[2025-10-01 13:38:23.396] [debug][Consumer]Extracted message type
    from message: BY_ID_AND_TYPE_ADVERTISEMENT
[2025-10-01 13:38:23.396] [debug][Consumer]Created name: /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR/byIDAndTypeAdvertisement:car002&CAR in
    func: express_by_id_and_type_advertisement_interest
[2025-10-01 13:38:23.396] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/byIDAndTypeAdvertisement%3
    Acar002%26CAR?CanBePrefix&MustBeFresh&Lifetime=6000
[2025-10-01 13:38:23.396] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    byIDAndTypeAdvertisement%3Acar002%26CAR
[2025-10-01 13:38:23.396] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    byIDAndTypeAdvertisement, Params: [car002, CAR]
[2025-10-01 13:38:23.396] [info][Producer]Processing Type
    Advertisement
[2025-10-01 13:38:23.396] [info][Producer]Type advertisement: name(
    registry) = CAR, ID = car002
[2025-10-01 13:38:23.396] [info][Producer][event] type_add: CAR ->
    car002
[2025-10-01 13:38:23.397] [info][Producer][resilience] mutate -> ver
    =10 hash=5de9c8ab762e109fd4b0a985de005a49
[2025-10-01 13:38:23.397] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(630cd269) 2(2bcd1ae9) 3(c9dc1836) 4(9
    a5a300d) 5(c872b4e2) 6(f1eb6fbc) 7(b5972495) 8(0c1d2fdb)
    9(231555f1) 10(5de9c8ab)
[2025-10-01 13:38:23.397] [debug][Producer][resilience] ops from v9
    -> v10:
[
  {
    "op": "add",
    "path": "/types/CAR/ids/car002",
    "value": true
```

```
 }
]
[2025-10-01 13:38:23.397] [info][Producer]Getting from IP: 127.0.0.1
    and port: 10000
[2025-10-01 13:38:23.397] [debug][Producer]HTTP GET /state?type=CAR
[2025-10-01 13:38:23.399] [info][Producer][resilience] mutate -> ver
    =11 hash=c84debb2b5768c8bb334fa056dd1a216
[2025-10-01 13:38:23.400] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(630cd269) 2(2bcd1ae9) 3(c9dc1836) 4(9
    a5a300d) 5(c872b4e2) 6(f1eb6fbc) 7(b5972495) 8(0c1d2fdb)
    9(231555f1) 10(5de9c8ab) 11(c84debb2)
[2025-10-01 13:38:23.400] [debug][Producer][resilience] ops from v10
    -> v11:
[
 {
  "op": "replace",
  "path": "/broker/collections/CAR/lastSeen",
  "value": "2025-10-01T13:38:23Z"
 }
]
[2025-10-01 13:38:23.400] [info][Producer][debug][event:type_add CAR
    ->car002] summary -> types=1 ids=2 subs(byId)=0 subs(byType)=0
    announced=6
[2025-10-01 13:38:23.400] [debug][Producer][debug][event:type_add CAR
    ->car002] type_registry:
[2025-10-01 13:38:23.400] [debug][Producer][CAR] car002 car001
[2025-10-01 13:38:23.400] [debug][Producer][debug][event:type_add CAR
    ->car002] subscribed_nodes_for_id:
[2025-10-01 13:38:23.400] [debug][Producer][debug][event:type_add CAR
    ->car002] subscribed_nodes_for_type:
[2025-10-01 13:38:23.400] [debug][Producer][debug][event:type_add CAR
    ->car002] announcements: /ndn/gr/edu/mmlab1/aueb/thomasi/
    CAR_registry /ndn/gr/edu/mmlab1/aueb/thomasi/CAR /ndn/gr/edu/
    mmlab1/aueb/thomasi/state_0 /ndn/gr/edu/mmlab1/aueb/thomasi/
    is_alive_0 /ndn/gr/edu/mmlab1/aueb/thomasi/SNDS_ID_scope_0 /ndn/
    gr/edu/mmlab1/aueb/thomasi_0
[2025-10-01 13:38:23.400] [info][Producer][debug][event:type_add CAR
    ->car002] broker.journal: entries=0 cursor=0 last_fp=
[2025-10-01 13:38:23.400] [debug][Producer][overlay][event:type_add
    CAR->car002] ver=0 hash=d41d8cd98f00b204e9800998ecf8427e keys=0
[2025-10-01 13:38:23.400] [info][Producer]Registered new ID car002
    for type CAR. Sending CONTROL.
[2025-10-01 13:38:23.400] [debug][Producer]Sending data to the NDN...
[2025-10-01 13:38:23.401] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/byIDAndTypeAdvertisement%3
    Acar002%26CAR in function onData
[2025-10-01 13:38:23.401] [info][Consumer]Data packet payload:
    ACK_ID_REGISTERED in function onData
[2025-10-01 13:38:23.401] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    byIDAndTypeAdvertisement, Params: [car002, CAR] in onData
[2025-10-01 13:38:23.401] [info][Consumer]Got ACK ACK_ID_REGISTERED.
    Skipping...
[2025-10-01 13:38:23.401] [info][Consumer]Reading packet...
[2025-10-01 13:38:23.401] [info][Worker]Received content: /ndn/gr/edu
    /mmlab1/aueb/thomasi/CAR/byIDAndTypeAdvertisement:car002&CAR
[2025-10-01 13:38:23.401] [info][Worker]Received payload:
    ACK_ID_REGISTERED
[2025-10-01 13:38:23.401] [info][Worker]Received type:
    GET_BY_ID_FOR_TYPE_RESPONSE
```

Let’s now publish to Producer’s @type=CAR registry through MMLab2:

Listing 33: Publish to Producer’s registry through MMLab2

```
scn4ndn@seeds7:~$ curl -v -i -X POST http://127.0.0.1:8080 -d "id=car
    003&type=CAR"
Note: Unnecessary use of -X or --request, POST is already inferred.
*Trying 127.0.0.1:8080...
*Connected to 127.0.0.1 (127.0.0.1) port 8080
>POST /HTTP/1.1
>Host: 127.0.0.1:8080
>User-Agent: curl/8.5.0
>Accept: */*
>Content-Length: 18
>Content-Type: application/x-www-form-urlencoded
>
<HTTP/1.1 200OK
HTTP/1.1 200OK
<Keep-Alive: timeout=5, max=100
Keep-Alive: timeout=5, max=100
<Content-Length: 260
Content-Length: 260
<Content-Type: text/plain
Content-Type: text/plain

<
{
  "batch_size": 1,
  "messages": [
    {
      "name": "/ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
          byIDAndTypeAdvertisement:car003&CAR",
      "payload": {
        "raw": "ACK_ID_REGISTERED\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
*Connection #0 to host 127.0.0.1 left intact
scn4ndn@seeds7:~$
```

Let's check the SeEDS Service logs of the Producer:

Listing 34: SeEDS Service logs after publishing through the MMLab2 node to @type=CAR registry

```

[2025-10-01 13:41:27.555] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    byIDAndTypeAdvertisement%3Acar003%26CAR
[2025-10-01 13:41:27.555] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    byIDAndTypeAdvertisement, Params: [car003, CAR]
[2025-10-01 13:41:27.555] [info][Producer]Processing Type
    Advertisement
[2025-10-01 13:41:27.555] [info][Producer]Type advertisement: name(
    registry) = CAR, ID = car003
[2025-10-01 13:41:27.555] [info][Producer][event] type_add: CAR ->
    car003
[2025-10-01 13:41:27.555] [info][Producer][resilience] mutate -> ver
    =12 hash=a029fd95434064b0a2deb814c82214c0
[2025-10-01 13:41:27.555] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(630cd269) 2(2bcd1ae9) 3(c9dc1836) 4(9
    a5a300d) 5(c872b4e2) 6(f1eb6fbc) 7(b5972495) 8(0c1d2fdb)
    9(231555f1) 10(5de9c8ab) 11(c84debb2) 12(a029fd95)
[2025-10-01 13:41:27.555] [debug][Producer][resilience] ops from v11
    -> v12:
[
  {
    "op": "add",
    "path": "/types/CAR/ids/car003",
    "value": true
  }
]
[2025-10-01 13:41:27.555] [info][Producer]Getting from IP: 127.0.0.1
    and port: 10000
[2025-10-01 13:41:27.555] [debug][Producer]HTTP GET /state?type=CAR
[2025-10-01 13:41:27.558] [info][Producer][resilience] mutate -> ver
    =13 hash=e6602199b9cb6dd9a1fb37adcffa2c1d
[2025-10-01 13:41:27.558] [debug][Producer][resilience] history
    versions: 0(37a6259c) 1(630cd269) 2(2bcd1ae9) 3(c9dc1836) 4(9
    a5a300d) 5(c872b4e2) 6(f1eb6fbc) 7(b5972495) 8(0c1d2fdb)
    9(231555f1) 10(5de9c8ab) 11(c84debb2) 12(a029fd95) 13(e6602199)
[2025-10-01 13:41:27.558] [debug][Producer][resilience] ops from v12
    -> v13:
[
  {
   "op": "replace",
   "path": "/broker/collections/CAR/lastSeen",
   "value": "2025-10-01T13:41:27Z"
  }
]
[2025-10-01 13:41:27.559] [info][Producer][debug][event:type_add CAR
    ->car003] summary -> types=1 ids=3 subs(byId)=0 subs(byType)=0
    announced=6
[2025-10-01 13:41:27.559] [debug][Producer][debug][event:type_add CAR
    ->car003] type_registry:
[2025-10-01 13:41:27.559] [debug][Producer][CAR] car003 car002 car001
[2025-10-01 13:41:27.559] [debug][Producer][debug][event:type_add CAR
    ->car003] subscribed_nodes_for_id:
[2025-10-01 13:41:27.559] [debug][Producer][debug][event:type_add CAR
    ->car003] subscribed_nodes_for_type:
[2025-10-01 13:41:27.559] [debug][Producer][debug][event:type_add CAR
    ->car003] announcements: /ndn/gr/edu/mmlab1/aueb/thomasi/
    CAR_registry /ndn/gr/edu/mmlab1/aueb/thomasi/CAR /ndn/gr/edu/
    mmlab1/aueb/thomasi/state_0 /ndn/gr/edu/mmlab1/aueb/thomasi/
    is_alive_0 /ndn/gr/edu/mmlab1/aueb/thomasi/SNDS_ID_scope_0 /ndn/
    gr/edu/mmlab1/aueb/thomasi_0
[2025-10-01 13:41:27.559] [info][Producer][debug][event:type_add CAR
    ->car003] broker.journal: entries=0 cursor=0 last_fp=
[2025-10-01 13:41:27.559] [debug][Producer][overlay][event:type_add
    CAR->car003] ver=0 hash=d41d8cd98f00b204e9800998ecf8427e keys=0
[2025-10-01 13:41:27.559] [info][Producer]Registered new ID car003
    for type CAR. Sending CONTROL.
[2025-10-01 13:41:27.559] [debug][Producer]Sending data to the NDN...
```

Let's try a *GET by TYPE* request from MMLab2:

Listing 35: *GET by TYPE* request for @type=CAR registry from MMLab2

```
scn4ndn@seeds7:~$ curl -i -X GET http://localhost:8080/?"type=CAR"
HTTP/1.1 200OK
Keep-Alive: timeout=5, max=100
Content-Length: 283
Content-Type: text/plain

{
  "batch_size": 1,
  "messages": [
   {
      "name": "/ndn/gr/edu/mmlab1/aueb/thomasi/CAR/scopeNotification:
          car003&CAR&SNDS_ID_scope_1",
      "payload": {
        "raw": "ACK_SUBSCRIPTION_NO_SUBSCRIBERS\n"
      },
      "sender": 6,
      "type": "CONTROL_MESSAGE"
    }
  ]
}
```

Let's check the logs of the Producer:

Listing 36: Producer logs after receiving a *GET by TYPE* request from `MMLab2`

```
[2025-10-01 13:43:29.560] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/
    CAR_registry/getByType%3ACAR
[2025-10-01 13:43:29.560] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR_registry, Command:
    getByType, Params: [CAR]
[2025-10-01 13:43:29.560] [info][Producer]Sending registry file
    payload for type CAR.
[2025-10-01 13:43:29.560] [debug][Producer]Sending data to the NDN...
[2025-10-01 13:43:29.567] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    temporalQuery%3Aid%3Dcar003%26type%3DCAR%26date%3D%26count%3D%26
    timeframe%3D%26filters%3D
[2025-10-01 13:43:29.567] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car003, type=CAR, date=, count=,
    timeframe=, filters=]
[2025-10-01 13:43:29.567] [debug][Producer]Created interest response:
     /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar003
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in: process_temporal_query_interest
[2025-10-01 13:43:29.567] [debug][Producer]TemporalQuery -> id:
    car003, date: , type: CAR, timeframe: , count: , filters:
[2025-10-01 13:43:29.567] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car003&type=CAR&count=&date=&timeframe=&
    filters=
[2025-10-01 13:43:29.571] [info][Producer]Successfully queried /
    temporal-query
[2025-10-01 13:43:29.571] [debug][Producer]Sending data to the NDN...
[2025-10-01 13:43:29.579] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    temporalQuery%3Aid%3Dcar002%26type%3DCAR%26date%3D%26count%3D%26
    timeframe%3D%26filters%3D
```

```
[2025-10-01 13:43:29.579] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car002, type=CAR, date=, count=,
    timeframe=, filters=]
[2025-10-01 13:43:29.579] [debug][Producer]Created interest response:
     /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar002
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in: process_temporal_query_interest
[2025-10-01 13:43:29.579] [debug][Producer]TemporalQuery -> id:
    car002, date: , type: CAR, timeframe: , count: , filters:
[2025-10-01 13:43:29.579] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car002&type=CAR&count=&date=&timeframe=&
    filters=
[2025-10-01 13:43:29.580] [info][Producer]Successfully queried /
    temporal-query
[2025-10-01 13:43:29.580] [debug][Producer]Sending data to the NDN...
[2025-10-01 13:43:29.584] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    temporalQuery%3Aid%3Dcar001%26type%3DCAR%26date%3D%26count%3D%26
    timeframe%3D%26filters%3D
[2025-10-01 13:43:29.584] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car001, type=CAR, date=, count=,
    timeframe=, filters=]
[2025-10-01 13:43:29.584] [debug][Producer]Created interest response:
     /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar001
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in: process_temporal_query_interest
[2025-10-01 13:43:29.584] [debug][Producer]TemporalQuery -> id:
    car001, date: , type: CAR, timeframe: , count: , filters:
[2025-10-01 13:43:29.584] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car001&type=CAR&count=&date=&timeframe=&
    filters=
[2025-10-01 13:43:29.585] [info][Producer]Successfully queried /
    temporal-query
[2025-10-01 13:43:29.585] [debug][Producer]Sending data to the NDN...
```

Let's check the logs of the MMLab2:

Listing 37: MMLab2 after receive a response for its *GET by TYPE* request for @type=CAR

```
[2025-10-01 13:43:29.557] [info][HTTPServer] Received GET request for
     / with target /?type=CAR
[2025-10-01 13:43:29.557] [info][HTTPServer] Received GET request
    from 127.0.0.1
Message: {type=CAR}
Type: GET_BY_TYPE_REQUEST
Sender: HTTP_SRV
```

https://doi.org/10.26219/heal.aueb.9822

```
Payload:
HTTP Type: HTTP_GET
Params:
type = CAR
[2025-10-01 13:43:29.557] [debug][HTTPServer] Writing message from
    GET request to service queue...
Message: {type=CAR}
Type: GET_BY_TYPE_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
type = CAR

[2025-10-01 13:43:29.557] [info][Worker]Received content: {type=CAR}
[2025-10-01 13:43:29.557] [info][Worker]Received payload:
[2025-10-01 13:43:29.557] [info][Worker]Received type:
    GET_BY_TYPE_REQUEST
[2025-10-01 13:43:29.557] [debug][Worker]Server intra message...
[2025-10-01 13:43:29.557] [info][Worker]Pushing getByType request for
     type: CAR_registry
[2025-10-01 13:43:29.557] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR_registry/
    getByType:CAR
[2025-10-01 13:43:29.557] [info][Worker]Reading packet...
[2025-10-01 13:43:29.557] [debug][Consumer]Extracted content length
    from message: 58
[2025-10-01 13:43:29.557] [debug][Consumer]Extracted message type
    from message: GET_BY_TYPE_REQUEST
[2025-10-01 13:43:29.557] [debug][Consumer]Created name: /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR_registry/getByType:CAR in func:
    express_get_by_type_interest
[2025-10-01 13:43:29.557] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR_registry/getByType%3ACAR?
    CanBePrefix&MustBeFresh&Lifetime=200
[2025-10-01 13:43:29.563] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR_registry/getByType%3ACAR in
    function onData
[2025-10-01 13:43:29.563] [info][Consumer]Data packet payload:
    Rcar003car002car001 in function onData
[2025-10-01 13:43:29.563] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR_registry, Command:
    getByType, Params: [CAR] in onData
[2025-10-01 13:43:29.563] [info][Consumer]Received Registry File
[2025-10-01 13:43:29.564] [info][Consumer]In Function: onData sending
     name CAR to worker Rcar003car002car001
[2025-10-01 13:43:29.564] [info][Worker]Received content: CAR
```

```
[2025-10-01 13:43:29.564] [info][Worker]Received payload:
    Rcar003car002car001
[2025-10-01 13:43:29.564] [info][Worker]Received type:
    REGISTRY_PAYLOAD
[2025-10-01 13:43:29.564] [info][Consumer]Reading packet...
[2025-10-01 13:43:29.564] [debug][Worker]Consumer intra message...
[2025-10-01 13:43:29.564] [debug][Worker]Number of IDs in payload: 3
[2025-10-01 13:43:29.564] [debug][Worker]Reading ID #1 with size 6
[2025-10-01 13:43:29.564] [info][Worker]Parsed ID #1: car003
[2025-10-01 13:43:29.564] [debug][Worker]Reading ID #2 with size 6
[2025-10-01 13:43:29.564] [info][Worker]Parsed ID #2: car002
[2025-10-01 13:43:29.564] [debug][Worker]Reading ID #3 with size 6
[2025-10-01 13:43:29.564] [info][Worker]Parsed ID #3: car001
[2025-10-01 13:43:29.564] [info][Worker]Size of ids in registry file:
     3
[2025-10-01 13:43:29.564] [debug][Worker]Processing registry id:
    car003
[2025-10-01 13:43:29.564] [info][Worker]Pushing request to the
    consumer for id car003
[2025-10-01 13:43:29.564] [debug][Worker]Processing registry id:
    car002
[2025-10-01 13:43:29.564] [info][Worker]Pushing request to the
    consumer for id car002
[2025-10-01 13:43:29.564] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    getByIDForType:car003&CAR
[2025-10-01 13:43:29.564] [debug][Consumer]Extracted content length
    from message: 61
[2025-10-01 13:43:29.564] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
[2025-10-01 13:43:29.564] [debug][Consumer]Query.type: CAR
[2025-10-01 13:43:29.564] [debug][Consumer]Query.id: car003
[2025-10-01 13:43:29.564] [info][Consumer]Created name /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR/temporalQuery:id%3Dcar003&type%3DCAR&
    date%3D&count%3D&timeframe%3D&filters%3D
[2025-10-01 13:43:29.564] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar003
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D?
    CanBePrefix&MustBeFresh&Lifetime=200
[2025-10-01 13:43:29.564] [debug][Worker]Processing registry id:
    car001
[2025-10-01 13:43:29.565] [info][Worker]Pushing request to the
    consumer for id car001
[2025-10-01 13:43:29.565] [info][Worker]Reading packet...
[2025-10-01 13:43:29.576] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar003
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in function onData
[2025-10-01 13:43:29.576] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car003",
      "battery": 95,
      "engine_temp": 79.4,
      "id": "car003",
      "speed": 55,
      "timestamp": {
        "$date": 1758793800000
      },
      "type": "CAR"
    }
  }
] in function onData
[2025-10-01 13:43:29.576] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car003, type=CAR, date=, count=,
    timeframe=, filters=] in onData
[2025-10-01 13:43:29.576] [info][Consumer]Temporal Query Reponse...
[2025-10-01 13:43:29.576] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-10-01 13:43:29.576] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car003",
      "battery": 95,
      "engine_temp": 79.4,
      "id": "car003",
      "speed": 55,
      "timestamp": {
        "$date": 1758793800000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:43:29.576] [debug][Consumer]Sanitized payload to:
[
  {
    "": {
      "_id": "car003",
      "battery": 95,
```

https://doi.org/10.26219/heal.aueb.9822

```
      "engine_temp": 79.4,
      "id": "car003",
      "speed": 55,
      "timestamp": {
        "$date": 1758793800000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:43:29.576] [debug][Consumer]Direct mode (single id in
    bucket).
[2025-10-01 13:43:29.576] [debug][Consumer]Got ID and type in direct:
     id=car003,type=CAR
[2025-10-01 13:43:29.576] [info][Worker]Received content: /?id=car003
    &type=CAR&date=&metafile=false&parentQueryHash=&lastInterest=
    true&mode=direct
[2025-10-01 13:43:29.576] [info][Worker]Received payload: [
  {
    "": {
      "_id": "car003",
      "battery": 95,
      "engine_temp": 79.4,
      "id": "car003",
      "speed": 55,
      "timestamp": {
        "$date": 1758793800000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:43:29.577] [info][Worker]Received type:
    VERSION_RESPONSE
[2025-10-01 13:43:29.577] [debug][Worker]Consumer intra message...
[2025-10-01 13:43:29.577] [info][Worker]Direct mode...
[2025-10-01 13:43:29.577] [debug][Worker]Cleared
    pending_get_by_id_requests for 'car003': erased=0
[2025-10-01 13:43:29.577] [info][Consumer]Reading packet...
[2025-10-01 13:43:29.577] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    getByIDForType:car002&CAR
[2025-10-01 13:43:29.577] [debug][Worker]Cleared
    pending_get_by_type_requests for 'CAR': erased=1
[2025-10-01 13:43:29.577] [debug][Worker]Direct mode: no filters
    discovered; returning full item
[2025-10-01 13:43:29.577] [info][Worker]Reading packet...
```

```
[2025-10-01 13:43:29.577] [debug][Consumer]Extracted content length
    from message: 61
[2025-10-01 13:43:29.577] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1758793800000,
            "id": "car003",
            "type": "CAR"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:43:29.577] [debug][Consumer]Query.type: CAR
[2025-10-01 13:43:29.577] [info][HTTPServer] Read Message from queue:
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1758793800000,
            "id": "car003",
            "type": "CAR"
          }
```

https://doi.org/10.26219/heal.aueb.9822

```
      ],
      "mode": "direct"
    },
    "sender": 6,
    "type": "TEMPORAL_QUERY_RESPONSE"
  }
 ]
}

HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:43:29.577] [debug][Consumer]Query.id: car002
[2025-10-01 13:43:29.577] [info][Consumer]Created name /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR/temporalQuery:id%3Dcar002&type%3DCAR&
    date%3D&count%3D&timeframe%3D&filters%3D
[2025-10-01 13:43:29.577] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar002
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D?
    CanBePrefix&MustBeFresh&Lifetime=200
[2025-10-01 13:43:29.583] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar002
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in function onData
[2025-10-01 13:43:29.583] [info][Consumer]Data packet payload: [
 {
  "": {
   "_id": "car002",
   "battery": 78,
   "engine_temp": 90.1,
   "id": "car002",
   "speed": 70,
   "timestamp": {
    "$date": 1758793200000
   },
   "type": "CAR"
  }
 }
] in function onData
[2025-10-01 13:43:29.583] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car002, type=CAR, date=, count=,
    timeframe=, filters=] in onData
[2025-10-01 13:43:29.583] [info][Consumer]Temporal Query Reponse...
[2025-10-01 13:43:29.583] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-10-01 13:43:29.583] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car002",
      "battery": 78,
      "engine_temp": 90.1,
      "id": "car002",
      "speed": 70,
      "timestamp": {
        "$date": 1758793200000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:43:29.583] [debug][Consumer]Sanitized payload to:
[
  {
    "": {
      "_id": "car002",
      "battery": 78,
      "engine_temp": 90.1,
      "id": "car002",
      "speed": 70,
      "timestamp": {
        "$date": 1758793200000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:43:29.583] [debug][Consumer]Direct mode (single id in
    bucket).
[2025-10-01 13:43:29.583] [debug][Consumer]Got ID and type in direct:
     id=car002,type=CAR
[2025-10-01 13:43:29.583] [info][Worker]Received content: /?id=car002
    &type=CAR&date=&metafile=false&parentQueryHash=&lastInterest=
    true&mode=direct
[2025-10-01 13:43:29.583] [info][Worker]Received payload: [
  {
    "": {
      "_id": "car002",
      "battery": 78,
      "engine_temp": 90.1,
      "id": "car002",
```

https://doi.org/10.26219/heal.aueb.9822

```
      "speed": 70,
      "timestamp": {
        "$date": 1758793200000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:43:29.583] [info][Worker]Received type:
    VERSION_RESPONSE
[2025-10-01 13:43:29.583] [debug][Worker]Consumer intra message...
[2025-10-01 13:43:29.583] [info][Worker]Direct mode...
[2025-10-01 13:43:29.583] [info][Consumer]Reading packet...
[2025-10-01 13:43:29.583] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    getByIDForType:car001&CAR
[2025-10-01 13:43:29.583] [debug][Consumer]Extracted content length
    from message: 61
[2025-10-01 13:43:29.583] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
[2025-10-01 13:43:29.583] [debug][Consumer]Query.type: CAR
[2025-10-01 13:43:29.583] [debug][Worker]Cleared
    pending_get_by_id_requests for 'car002': erased=0
[2025-10-01 13:43:29.583] [debug][Worker]Cleared
    pending_get_by_type_requests for 'CAR': erased=0
[2025-10-01 13:43:29.583] [debug][Worker]Direct mode: no filters
    discovered; returning full item
[2025-10-01 13:43:29.583] [info][Worker]Reading packet...
[2025-10-01 13:43:29.583] [debug][Consumer]Query.id: car001
[2025-10-01 13:43:29.583] [info][Consumer]Created name /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR/temporalQuery:id%3Dcar001&type%3DCAR&
    date%3D&count%3D&timeframe%3D&filters%3D
[2025-10-01 13:43:29.583] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar001
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D?
    CanBePrefix&MustBeFresh&Lifetime=200
[2025-10-01 13:43:29.587] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar001
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in function onData
[2025-10-01 13:43:29.587] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car001",
      "battery": 87,
      "engine_temp": 92.4,
      "id": "car001",
      "speed": 75,
      "timestamp": {
        "$date": 1758794400000
      },
      "type": "CAR"
    }
  }
] in function onData
[2025-10-01 13:43:29.588] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car001, type=CAR, date=, count=,
    timeframe=, filters=] in onData
[2025-10-01 13:43:29.588] [info][Consumer]Temporal Query Reponse...
[2025-10-01 13:43:29.588] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-10-01 13:43:29.588] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car001",
      "battery": 87,
      "engine_temp": 92.4,
      "id": "car001",
      "speed": 75,
      "timestamp": {
        "$date": 1758794400000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:43:29.588] [debug][Consumer]Sanitized payload to:
[
  {
    "": {
      "_id": "car001",
      "battery": 87,
      "engine_temp": 92.4,
      "id": "car001",
      "speed": 75,
      "timestamp": {
        "$date": 1758794400000
      },
      "type": "CAR"
    }
```

https://doi.org/10.26219/heal.aueb.9822

```
  }
]
[2025-10-01 13:43:29.588] [debug][Consumer]Direct mode (single id in
    bucket).
[2025-10-01 13:43:29.588] [debug][Consumer]Got ID and type in direct:
     id=car001,type=CAR
[2025-10-01 13:43:29.588] [info][Consumer]Reading packet...
[2025-10-01 13:43:29.588] [info][Worker]Received content: /?id=car001
    &type=CAR&date=&metafile=false&parentQueryHash=&lastInterest=
    true&mode=direct
[2025-10-01 13:43:29.588] [info][Worker]Received payload: [
  {
    "": {
      "_id": "car001",
      "battery": 87,
      "engine_temp": 92.4,
      "id": "car001",
      "speed": 75,
      "timestamp": {
        "$date": 1758794400000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:43:29.588] [info][Worker]Received type:
    VERSION_RESPONSE
[2025-10-01 13:43:29.588] [debug][Worker]Consumer intra message...
[2025-10-01 13:43:29.588] [info][Worker]Direct mode...
[2025-10-01 13:43:29.588] [debug][Worker]Cleared
    pending_get_by_id_requests for 'car001': erased=0
[2025-10-01 13:43:29.588] [debug][Worker]Cleared
    pending_get_by_type_requests for 'CAR': erased=0
[2025-10-01 13:43:29.588] [debug][Worker]Direct mode: no filters
    discovered; returning full item
[2025-10-01 13:43:29.588] [info][Worker]Reading packet...
```

Let's also do the same *GET by TYPE* in the UMemphis node:

```
scn4ndn@seeds23:~$ curl -i -X GET http://localhost:8080/?"type=CAR"
HTTP/1.1 200OK
Keep-Alive: timeout=5, max=100
Content-Length: 328
Content-Type: text/plain

{
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1758793200000,
            "id": "car002",
            "type": "CAR"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}
```

Let’s check the Producer Service logs, after receiving the *GET by TYPE*:

Listing 38: Producer logs after receiving *GET by TYPE* from the UMemphis node.

```
[2025-10-01 13:45:36.170] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/
    CAR_registry/getByType%3ACAR
[2025-10-01 13:45:36.171] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR_registry, Command:
    getByType, Params: [CAR]
[2025-10-01 13:45:36.171] [info][Producer]Sending registry file
    payload for type CAR.
[2025-10-01 13:45:36.171] [debug][Producer]Sending data to the NDN...
[2025-10-01 13:45:36.499] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    temporalQuery%3Aid%3Dcar003%26type%3DCAR%26date%3D%26count%3D%26
    timeframe%3D%26filters%3D
[2025-10-01 13:45:36.499] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car003, type=CAR, date=, count=,
    timeframe=, filters=]
[2025-10-01 13:45:36.499] [debug][Producer]Created interest response:
     /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar003
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in: process_temporal_query_interest
[2025-10-01 13:45:36.499] [debug][Producer]TemporalQuery -> id:
    car003, date: , type: CAR, timeframe: , count: , filters:
[2025-10-01 13:45:36.499] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car003&type=CAR&count=&date=&timeframe=&
    filters=
```

https://doi.org/10.26219/heal.aueb.9822

```
[2025-10-01 13:45:36.503] [info][Producer]Successfully queried /
    temporal-query
[2025-10-01 13:45:36.503] [debug][Producer]Sending data to the NDN...
[2025-10-01 13:45:36.832] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    temporalQuery%3Aid%3Dcar002%26type%3DCAR%26date%3D%26count%3D%26
    timeframe%3D%26filters%3D
[2025-10-01 13:45:36.832] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car002, type=CAR, date=, count=,
    timeframe=, filters=]
[2025-10-01 13:45:36.832] [debug][Producer]Created interest response:
     /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar002
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in: process_temporal_query_interest
[2025-10-01 13:45:36.832] [debug][Producer]TemporalQuery -> id:
    car002, date: , type: CAR, timeframe: , count: , filters:
[2025-10-01 13:45:36.832] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car002&type=CAR&count=&date=&timeframe=&
    filters=
[2025-10-01 13:45:36.835] [info][Producer]Successfully queried /
    temporal-query
[2025-10-01 13:45:36.835] [debug][Producer]Sending data to the NDN...
[2025-10-01 13:45:37.168] [info][Producer]Received interest in
    function: onInterest, name: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    temporalQuery%3Aid%3Dcar001%26type%3DCAR%26date%3D%26count%3D%26
    timeframe%3D%26filters%3D
[2025-10-01 13:45:37.168] [info][Producer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car001, type=CAR, date=, count=,
    timeframe=, filters=]
[2025-10-01 13:45:37.168] [debug][Producer]Created interest response:
     /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar001
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in: process_temporal_query_interest
[2025-10-01 13:45:37.168] [debug][Producer]TemporalQuery -> id:
    car001, date: , type: CAR, timeframe: , count: , filters:
[2025-10-01 13:45:37.168] [debug][Producer]Requesting to endpoint: /
    temporal-query?id=car001&type=CAR&count=&date=&timeframe=&
    filters=
[2025-10-01 13:45:37.171] [info][Producer]Successfully queried /
    temporal-query
[2025-10-01 13:45:37.171] [debug][Producer]Sending data to the NDN...
```

Let's check the SeEDS Service logs of the UMemphis node:

Listing 39: Logs of the SeEDS Service of the UMemphis node, after getting a

response from *GET by TYPE*

```
[2025-10-01 13:45:36.006] [info][HTTPServer] Received GET request for
     / with body
[2025-10-01 13:45:36.006] [info][HTTPServer] Received GET request for
     / with target /?type=CAR
[2025-10-01 13:45:36.006] [info][HTTPServer] Received GET request
    from 127.0.0.1
Message: {type=CAR}
Type: GET_BY_TYPE_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
type = CAR
[2025-10-01 13:45:36.006] [debug][HTTPServer] Writing message from
    GET request to service queue...
Message: {type=CAR}
Type: GET_BY_TYPE_REQUEST
Sender: HTTP_SRV
Payload:
HTTP Type: HTTP_GET
Params:
type = CAR

[2025-10-01 13:45:36.006] [info][Worker]Received content: {type=CAR}
[2025-10-01 13:45:36.006] [info][Worker]Received payload:
[2025-10-01 13:45:36.006] [info][Worker]Received type:
    GET_BY_TYPE_REQUEST
[2025-10-01 13:45:36.006] [debug][Worker]Server intra message...
[2025-10-01 13:45:36.006] [info][Worker]Pushing getByType request for
     type: CAR_registry
[2025-10-01 13:45:36.006] [info][Worker]Reading packet...
[2025-10-01 13:45:36.006] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR_registry/
    getByType:CAR
[2025-10-01 13:45:36.006] [debug][Consumer]Extracted content length
    from message: 58
[2025-10-01 13:45:36.006] [debug][Consumer]Extracted message type
    from message: GET_BY_TYPE_REQUEST
[2025-10-01 13:45:36.006] [debug][Consumer]Created name: /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR_registry/getByType:CAR in func:
    express_get_by_type_interest
[2025-10-01 13:45:36.006] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR_registry/getByType%3ACAR?
    CanBePrefix&MustBeFresh&Lifetime=6000
[2025-10-01 13:45:36.336] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR_registry/getByType%3ACAR in
    function onData
[2025-10-01 13:45:36.336] [info][Consumer]Data packet payload:
    Rcar003car002car001 in function onData
[2025-10-01 13:45:36.336] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR_registry, Command:
    getByType, Params: [CAR] in onData
[2025-10-01 13:45:36.336] [info][Consumer]Received Registry File
[2025-10-01 13:45:36.336] [info][Consumer]In Function: onData sending
     name CAR to worker Rcar003car002car001
[2025-10-01 13:45:36.336] [info][Consumer]Reading packet...
[2025-10-01 13:45:36.336] [info][Worker]Received content: CAR
[2025-10-01 13:45:36.336] [info][Worker]Received payload:
    Rcar003car002car001
[2025-10-01 13:45:36.336] [info][Worker]Received type:
    REGISTRY_PAYLOAD
[2025-10-01 13:45:36.336] [debug][Worker]Consumer intra message...
[2025-10-01 13:45:36.336] [debug][Worker]Number of IDs in payload: 3
[2025-10-01 13:45:36.336] [debug][Worker]Reading ID #1 with size 6
[2025-10-01 13:45:36.336] [info][Worker]Parsed ID #1: car003
[2025-10-01 13:45:36.336] [debug][Worker]Reading ID #2 with size 6
[2025-10-01 13:45:36.336] [info][Worker]Parsed ID #2: car002
[2025-10-01 13:45:36.336] [debug][Worker]Reading ID #3 with size 6
[2025-10-01 13:45:36.336] [info][Worker]Parsed ID #3: car001
[2025-10-01 13:45:36.336] [info][Worker]Size of ids in registry file:
     3
[2025-10-01 13:45:36.336] [debug][Worker]Processing registry id:
    car003
[2025-10-01 13:45:36.337] [info][Worker]Pushing request to the
    consumer for id car003
[2025-10-01 13:45:36.337] [debug][Worker]Processing registry id:
    car002
[2025-10-01 13:45:36.337] [info][Worker]Pushing request to the
    consumer for id car002
[2025-10-01 13:45:36.337] [debug][Worker]Processing registry id:
    car001
[2025-10-01 13:45:36.337] [info][Worker]Pushing request to the
    consumer for id car001
[2025-10-01 13:45:36.337] [info][Worker]Reading packet...
[2025-10-01 13:45:36.337] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    getByIDForType:car003&CAR
[2025-10-01 13:45:36.337] [debug][Consumer]Extracted content length
    from message: 61
[2025-10-01 13:45:36.337] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
[2025-10-01 13:45:36.337] [debug][Consumer]Query.type: CAR
[2025-10-01 13:45:36.337] [debug][Consumer]Query.id: car003
```

```
[2025-10-01 13:45:36.337] [info][Consumer]Created name /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR/temporalQuery:id%3Dcar003&type%3DCAR&
    date%3D&count%3D&timeframe%3D&filters%3D
[2025-10-01 13:45:36.337] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar003
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D?
    CanBePrefix&MustBeFresh&Lifetime=6000
[2025-10-01 13:45:36.668] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar003
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in function onData
[2025-10-01 13:45:36.668] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car003",
      "battery": 95,
      "engine_temp": 79.4,
      "id": "car003",
      "speed": 55,
      "timestamp": {
        "$date": 1758793800000
      },
      "type": "CAR"
    }
  }
] in function onData
[2025-10-01 13:45:36.668] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car003, type=CAR, date=, count=,
    timeframe=, filters=] in onData
[2025-10-01 13:45:36.668] [info][Consumer]Temporal Query Reponse...
[2025-10-01 13:45:36.669] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-10-01 13:45:36.669] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car003",
      "battery": 95,
      "engine_temp": 79.4,
      "id": "car003",
      "speed": 55,
      "timestamp": {
        "$date": 1758793800000
      },
    "type": "CAR"
   }
  }
]
[2025-10-01 13:45:36.669] [debug][Consumer]Sanitized payload to:
[
  {
   "": {
    "_id": "car003",
    "battery": 95,
    "engine_temp": 79.4,
    "id": "car003",
    "speed": 55,
    "timestamp": {
      "$date": 1758793800000
    },
    "type": "CAR"
   }
  }
]
[2025-10-01 13:45:36.669] [debug][Consumer]Direct mode (single id in
    bucket).
[2025-10-01 13:45:36.669] [debug][Consumer]Got ID and type in direct:
     id=car003,type=CAR
[2025-10-01 13:45:36.669] [info][Consumer]Reading packet...
[2025-10-01 13:45:36.669] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    getByIDForType:car002&CAR
[2025-10-01 13:45:36.669] [debug][Consumer]Extracted content length
    from message: 61
[2025-10-01 13:45:36.669] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
[2025-10-01 13:45:36.669] [debug][Consumer]Query.type: CAR
[2025-10-01 13:45:36.669] [debug][Consumer]Query.id: car002
[2025-10-01 13:45:36.669] [info][Consumer]Created name /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR/temporalQuery:id%3Dcar002&type%3DCAR&
    date%3D&count%3D&timeframe%3D&filters%3D
[2025-10-01 13:45:36.669] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar002
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D?
    CanBePrefix&MustBeFresh&Lifetime=6000
[2025-10-01 13:45:36.669] [info][Worker]Received content: /?id=car003
    &type=CAR&date=&metafile=false&parentQueryHash=&lastInterest=
    true&mode=direct
[2025-10-01 13:45:36.669] [info][Worker]Received payload: [
  {
   "": {
      "_id": "car003",
      "battery": 95,
      "engine_temp": 79.4,
      "id": "car003",
      "speed": 55,
      "timestamp": {
        "$date": 1758793800000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:45:36.669] [info][Worker]Received type:
    VERSION_RESPONSE
[2025-10-01 13:45:36.669] [debug][Worker]Consumer intra message...
[2025-10-01 13:45:36.669] [info][Worker]Direct mode...
[2025-10-01 13:45:36.670] [debug][Worker]Cleared
    pending_get_by_id_requests for 'car003': erased=0
[2025-10-01 13:45:36.670] [debug][Worker]Cleared
    pending_get_by_type_requests for 'CAR': erased=1
[2025-10-01 13:45:36.670] [debug][Worker]Direct mode: no filters
    discovered; returning full item
[2025-10-01 13:45:36.670] [info][Worker]Reading packet...
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1758793800000,
            "id": "car003",
            "type": "CAR"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}

```

```
HTTP Type: HTTP_EMPTY
Params:
[2025-10-01 13:45:36.671] [info][HTTPServer] Read Message from queue:
Message: BATCH
Type: GENERIC
Sender: WORKER
Payload: {
  "batch_size": 1,
  "messages": [
    {
      "name": "",
      "payload": {
        "items": [
          {
            "date": 1758793800000,
            "id": "car003",
            "type": "CAR"
          }
        ],
        "mode": "direct"
      },
      "sender": 6,
      "type": "TEMPORAL_QUERY_RESPONSE"
    }
  ]
}

HTTP Type: HTTP_EMPTY
Params:

[2025-10-01 13:45:37.000] [info][Consumer]Received Data with name: /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar002
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in function onData
[2025-10-01 13:45:37.001] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car002",
      "battery": 78,
      "engine_temp": 90.1,
      "id": "car002",
      "speed": 70,
      "timestamp": {
        "$date": 1758793200000
      },
      "type": "CAR"
    }
  }
] in function onData
[2025-10-01 13:45:37.001] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car002, type=CAR, date=, count=,
    timeframe=, filters=] in onData
[2025-10-01 13:45:37.001] [info][Consumer]Temporal Query Reponse...
[2025-10-01 13:45:37.001] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-10-01 13:45:37.001] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car002",
      "battery": 78,
      "engine_temp": 90.1,
      "id": "car002",
      "speed": 70,
      "timestamp": {
        "$date": 1758793200000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:45:37.002] [debug][Consumer]Sanitized payload to:
[
  {
    "": {
      "_id": "car002",
      "battery": 78,
      "engine_temp": 90.1,
      "id": "car002",
      "speed": 70,
      "timestamp": {
        "$date": 1758793200000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:45:37.002] [debug][Consumer]Direct mode (single id in
    bucket).
[2025-10-01 13:45:37.002] [debug][Consumer]Got ID and type in direct:
     id=car002,type=CAR
```

```
[2025-10-01 13:45:37.002] [info][Worker]Received content: /?id=car002
    &type=CAR&date=&metafile=false&parentQueryHash=&lastInterest=
    true&mode=direct
[2025-10-01 13:45:37.004] [info][Worker]Received payload: [
  {
    "": {
      "_id": "car002",
      "battery": 78,
      "engine_temp": 90.1,
      "id": "car002",
      "speed": 70,
      "timestamp": {
        "$date": 1758793200000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:45:37.005] [info][Worker]Received type:
    VERSION_RESPONSE
[2025-10-01 13:45:37.005] [debug][Worker]Consumer intra message...
[2025-10-01 13:45:37.005] [info][Worker]Direct mode...
[2025-10-01 13:45:37.002] [info][Consumer]Reading packet...
[2025-10-01 13:45:37.005] [debug][Worker]Cleared
    pending_get_by_id_requests for 'car002': erased=0
[2025-10-01 13:45:37.005] [debug][Consumer]Extracted content from
    message: /ndn/gr/edu/mmlab1/aueb/thomasi/CAR/
    getByIDForType:car001&CAR
[2025-10-01 13:45:37.005] [debug][Consumer]Extracted content length
    from message: 61
[2025-10-01 13:45:37.005] [debug][Consumer]Extracted message type
    from message: GET_BY_ID_FOR_TYPE_REQUEST
[2025-10-01 13:45:37.005] [debug][Consumer]Query.type: CAR
[2025-10-01 13:45:37.005] [debug][Consumer]Query.id: car001
[2025-10-01 13:45:37.005] [debug][Worker]Cleared
    pending_get_by_type_requests for 'CAR': erased=0
[2025-10-01 13:45:37.005] [debug][Worker]Direct mode: no filters
    discovered; returning full item
[2025-10-01 13:45:37.005] [info][Consumer]Created name /ndn/gr/edu/
    mmlab1/aueb/thomasi/CAR/temporalQuery:id%3Dcar001&type%3DCAR&
    date%3D&count%3D&timeframe%3D&filters%3D
[2025-10-01 13:45:37.006] [info][Worker]Reading packet...
[2025-10-01 13:45:37.006] [info][Consumer]Sending actual Interest /
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar001
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D?
    CanBePrefix&MustBeFresh&Lifetime=6000
[2025-10-01 13:45:37.336] [info][Consumer]Received Data with name: /
```

```
    ndn/gr/edu/mmlab1/aueb/thomasi/CAR/temporalQuery%3Aid%3Dcar001
    %26type%3DCAR%26date%3D%26count%3D%26timeframe%3D%26filters%3D
    in function onData
[2025-10-01 13:45:37.336] [info][Consumer]Data packet payload: [
  {
    "": {
      "_id": "car001",
      "battery": 87,
      "engine_temp": 92.4,
      "id": "car001",
      "speed": 75,
      "timestamp": {
        "$date": 1758794400000
      },
      "type": "CAR"
    }
  }
] in function onData
[2025-10-01 13:45:37.336] [info][Consumer]Parsed interest: Prefix:
    ndn/gr/edu/mmlab1/aueb/thomasi, Name: CAR, Command:
    temporalQuery, Params: [id=car001, type=CAR, date=, count=,
    timeframe=, filters=] in onData
[2025-10-01 13:45:37.336] [info][Consumer]Temporal Query Reponse...
[2025-10-01 13:45:37.336] [debug][Consumer]Removed extra characters
    from payload (first 32 bytes hex): 5b 0a 20 20 7b 0a 20 20 20 20
     22 22 3a 20 7b 0a 20 20 20 20 20 20 22 5f 69 64 22 3a 20 22 63
    61
[2025-10-01 13:45:37.336] [debug][Consumer]Candidate JSON slice:
[
  {
    "": {
      "_id": "car001",
      "battery": 87,
      "engine_temp": 92.4,
      "id": "car001",
      "speed": 75,
      "timestamp": {
        "$date": 1758794400000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:45:37.336] [debug][Consumer]Sanitized payload to:
[
  {
    "": {
      "_id": "car001",
      "battery": 87,
      "engine_temp": 92.4,
      "id": "car001",
      "speed": 75,
      "timestamp": {
        "$date": 1758794400000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:45:37.337] [debug][Consumer]Direct mode (single id in
    bucket).
[2025-10-01 13:45:37.337] [debug][Consumer]Got ID and type in direct:
     id=car001,type=CAR
[2025-10-01 13:45:37.337] [info][Consumer]Reading packet...
[2025-10-01 13:45:37.337] [info][Worker]Received content: /?id=car001
    &type=CAR&date=&metafile=false&parentQueryHash=&lastInterest=
    true&mode=direct
[2025-10-01 13:45:37.337] [info][Worker]Received payload: [
  {
    "": {
      "_id": "car001",
      "battery": 87,
      "engine_temp": 92.4,
      "id": "car001",
      "speed": 75,
      "timestamp": {
        "$date": 1758794400000
      },
      "type": "CAR"
    }
  }
]
[2025-10-01 13:45:37.337] [info][Worker]Received type:
    VERSION_RESPONSE
[2025-10-01 13:45:37.337] [debug][Worker]Consumer intra message...
[2025-10-01 13:45:37.337] [info][Worker]Direct mode...
[2025-10-01 13:45:37.337] [debug][Worker]Cleared
    pending_get_by_id_requests for 'car001': erased=0
[2025-10-01 13:45:37.337] [debug][Worker]Cleared
    pending_get_by_type_requests for 'CAR': erased=0
[2025-10-01 13:45:37.337] [debug][Worker]Direct mode: no filters
    discovered; returning full item
[2025-10-01 13:45:37.337] [info][Worker]Reading packet...
```

We observe that both UMemphis and MMLab2 successfully retrieve the data from the `@type` registry of their respective Producer nodes. This demonstrates that our *GET by TYPE* prototype operates correctly across geographically distant sites and remains robust at large network scales.