Service-Centric Networking

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Mikael Gasparyan

von Sierre, VS

Leiter der Arbeit:
Prof. Dr. Torsten Braun
Institut für Informatik

Service-Centric Networking

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Mikael Gasparyan

von Sierre, VS

Leiter der Arbeit:
Prof. Dr. Torsten Braun
Institut für Informatik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, Mai 2020

Der Dekan:
Prof. Dr. Zoltan Balogh

I would like to dedicate this thesis to my loving family…

Acknowledgments

The work presented in this dissertation was carried out as part of my doctoral studies at the University of Bern. My employment as a Research Assistant and Ph.D. student at the Institute of Computer Science (INF) of the University of Bern was partly financed by the SNF project "Service-Centric Networking" [1].

This work would not have been possible without the support of many people to whom I would like to express my gratitude. Firstly, I would like to express my deepest gratitude to my doctoral thesis supervisor Prof. Dr. Torsten Braun, for offering me the possibility to pursue a doctoral thesis under his supervision. I would like to express my thankfulness to my supervisor for the support he provided me during my entire doctoral studies. He provided me with advice, ideas, and corrections that enabled me to grow as a researcher, write publications and complete my doctoral degree. He guided me and at the same time gave me enough freedom to develop my own ideas and become an independent researcher throughout the years. I would like to express my deepest gratitude to my thesis examiner Prof. Dr. Marilia Curado for her valuable comments and suggestions and the president of the jury Prof. Dr. Paolo Favaro.

I want to express my thankfulness to all my colleagues for all the interesting exchanges and the excellent time that we had as a group. I would like to especially thank to Dr. Eryk Schiller for his support and feedback that he provided me with during the research activities throughout the past years, and for his valuable comments and suggestions on the thesis. I would also like to express my gratefulness to my former and present colleagues: Imad Aad, Carlos Anastasiades, Eirina Bourtsoulatze, José Carrera, Negar Emami, Andre Gomes, Gomes Duarte João do Monte, Mostafa Karimzadeh, Zan Li, Ali Marandi, Eirini Kalogeiton, Luca Luceri, Gaetano Manzo, Alisson Medeiros, Diego Oliveira Rodrigues, Jonnahtan Saltarin, Hugo Santos, Jakob Schärer, Marcel Stolz, Allan Mariano de Souza, Matthias Thoma, Dimitrios Xenakis, and Zhongliang Zhao. I would like to thank my only Bachelor thesis student Guillaume Corsini, working with such a committed and skilled student was a pleasure.

I want to address my special thanks to the CDS group secretary Daniela Schroth for her kind organizational support during all the administrative dealings with the University of Bern and beyond. I would like to address my thanks also to the institute's system administrator Dr. Peppo Brambilla. He has always provided fast support in the case of technical issues.

Finally, I want to address my thanks to my parents Ara and Nariné and my sisters Ani and Eliza. You have always been supporting me during all the periods of my life. You always stood on my side and encouraged me during all the difficulties and doubts. I would also like to thank all my friends and relatives, and also all the important people that I may have forgotten to mention.

Abstract

The current Internet is host-based, meaning that a content consumer needs to possess the content provider's address to retrieve the given content. Information-Centric Networking (ICN) is a future Internet architecture. ICN aims to substitute the current host-centric architecture of the Internet with an information-centric one. Service-Centric Networking (SCN) is a future Internet paradigm derived from ICN. SCN extends the ICN paradigm with service support; in SCN, services are the key component of the network. One of the most prominent ICN implementations is Named Data Networking (NDN). Our contributions in this doctoral thesis are the extension of NDN with service support and the satisfaction of SCN requirements. We have extended NDN with service support and developed architectures and mechanisms to satisfy a set of SCN requirements. The contributions are divided into two parts: L-SCN architecture and Session Support for SCN.

In the first part, we have developed L-SCN (layered SCN architecture with Supernodes and Bloom Filters), the first SCN routing architecture, which makes use of a two-layer forwarding scheme composed of inter-domain and intra-domain communication. Unlike existing SCN routing architectures relying on a flat organization, our design splits the network into domains. Nodes within a domain possess significant knowledge about existing services and available resources within the domain. Supernodes provide a significant advantage in comparison to other architectures. They assure the inter-domain communication and make use of a pull and push mechanism combined with Bloom filters. It allows us to minimize the protocol overhead and optimize the sharing of information about available services and resources in the network. We have extended the L-SCN architecture with an additional forwarding support mechanism and further communication approaches. The extended architectural approach offers alternative communication strategies and improves the design in multiple dimensions, such as service accessibility and protocol overhead. We make use of an extended version of the Named Link State Routing (NLSR) protocol to disseminate service provider prefixes and resource availability information within the network. We have designed a genuine parameter support scheme that enables us to identify the parameters contained in a given service request by using hashes computed for the parameters. Our design supports short and long types of service input parameters.

In the second part, we have designed the first service session support scheme for SCN. We extended NDN to support services and to prototype our session support. Our design makes use of existing hierarchical naming schemes to specify sessions using unique session identifiers. Sessions are established through a two-way handshake, which allows both the service consumer and provider to exchange their generated unique session identifiers. The concatenation of the unique identifiers produced by the consumer and the provider forms the session identifier for a given session. Upon the two-way information exchange, the intermediate nodes store the session identifier, which will enable them to forward requests

concerning a given session towards the session provider. However, the session identifier is only known by the intermediate routers along the single path over which the session was established. Therefore, we have enhanced our session support mechanism with multipath routing capabilities, which enable the session requests to be forwarded over different paths towards the service provider of a given session. We present the design and evaluation of three multipath routing techniques. The first mechanism is based upon the propagating of session identifiers within the network using Bloom filters, the second design is based upon the propagation of service provider identifiers, and the third design uses piggybacking for the propagation of service provider identifiers.

Contents

List of Figures

List of Tables

Acronyms

CS	Content Store
DHT	Distributed Hash Table
DIM	Data Information Message
EFIB	Extended Forwarding Information Base
FIB	Forwarding Information Base
ICN	Information-Centric Networking
IIM	Interest Information Message
IP	Internet Protocol
LSA	Link State Advertisements
L-SCN	Layered SCN architecture with Supernodes and Bloom Filters
NFaaS	Named Function as a Service
NLSR	Named data Link State Routing
NMO	Normalized message overhead
NPR	Normalized performance ratio
PIBI	Provider Identifier Broadcast Interest
PIT	Pending Interest Table
RAI	Resource Availability Information
SAI	Service Availability Information
SAL	Service access layer
SCN	Service-Centric Networking
SPI	Service provider identifier
SRID	Service Resource Information Data
SRII	Service Resource Information Interest
SSD	Service Start Data
SSI	Session Start Interest
NDN	Named Data Networking
PAI	Provider Availability Information
RAC	Resource Availability Changes
SAC	Service Availability Changes
SOFIA	Toward Service-Oriented Information-Centric Networking

Chapter 1 Introduction

This chapter presents the historical context that motivated the research topic and led to this doctoral thesis. We will also describe the main goals and the structure of the thesis.

1.1 Historical Background, Context, and General Motivation

The development of the Internet started in 1969. From the beginning, the Internet was designed according to the host-centric paradigm. The idea was to connect remote machines together and enable them to exchange information. The host-centric approach builds upon the end-to-end networking design principle, which relies on perpetual connectivity between intermediate forwarding nodes. The end-to-end computer network principle aims to install application-specific functions on the end nodes, rather than on the intermediate forwarding nodes, through which the communication between remote machines is established.

The host-centric approach has a solid historical foundation. It has been used by the telegraph and telephone, which were invented in the 19th century. The telegraph used dedicated wire cables installed between communication parties to exchange information encoded in electrical signals. At its beginning, the telephone network also used dedicated wires between all the communication entities to enable voice transmissions. However, due to the fast-growing development of the telephone, it was no longer possible to build dedicated cables between all the communicating parties, cf. Figure 1-1 (a), as it requires the installation of $n \times (n-1)$ wires, where n is the number of users in the universe of the communication parties. To this end, central switching stations were established, allowing two telephone users without a dedicated end-to-end wire to establish voice communication through the central relay point, i.e., the star topology was developed (cf. Figure 1-1 (b)). As the system grew in size, the switching stations were connected over the second-level switching infrastructure, as depicted in Figure 1-1 (c), however, the fundamental goal was to establish a 1-to-1 circuit between any two communicating parties.

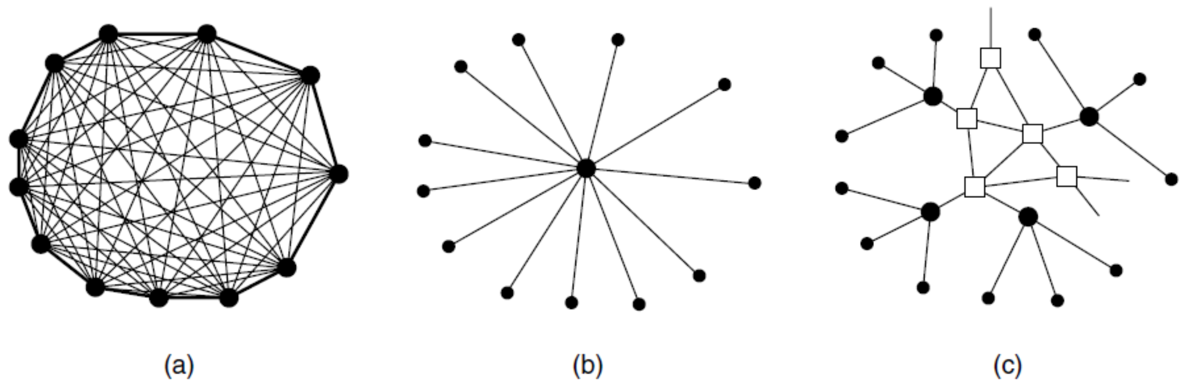


Figure 1-1: (a) Fully inter-connected network (b) Network managed by a central hub (c) Network managed at multiple levels [2]

The architectural approach used by computer networks inherits from the host-centric approach of the telephone. In computer networks, even though the switching has evolved from circuit switching to packet switching, the general idea is still identical as in telephone network, meaning that the network connects two communicating entities. The traditional telephone uses numbers (i.e., telephone numbers) to identify end-users. Therefore, a caller dials the telephone number of a callee to establish a circuit between them through which the voice can get transmitted in two directions. In computer networks, Internet Protocol (IP) [3] addresses are typically used instead of telephone numbers to identify and establish a connection between two communicating parties willing to exchange information. Furthermore, as mentioned, the underlying transmission mode differs. Traditional telephony is based on circuit switching, while computer networks generally rely on packet switching. In computer networks, information chunks (i.e., packets) are sent from a source to the corresponding destination without establishing a physical link between the communicating entities. The packets originate at the source, and are being forwarded to the destination by the set of intermediate forwarding nodes. The forwarding procedure typically depends on the destination address of a given packet. The forwarding node chooses an appropriate next-hop that leads the packet closer to the destination. The next-hop is decided by using the information about the network topology established by a routing protocol and written into the routing tables on every communicating entity.

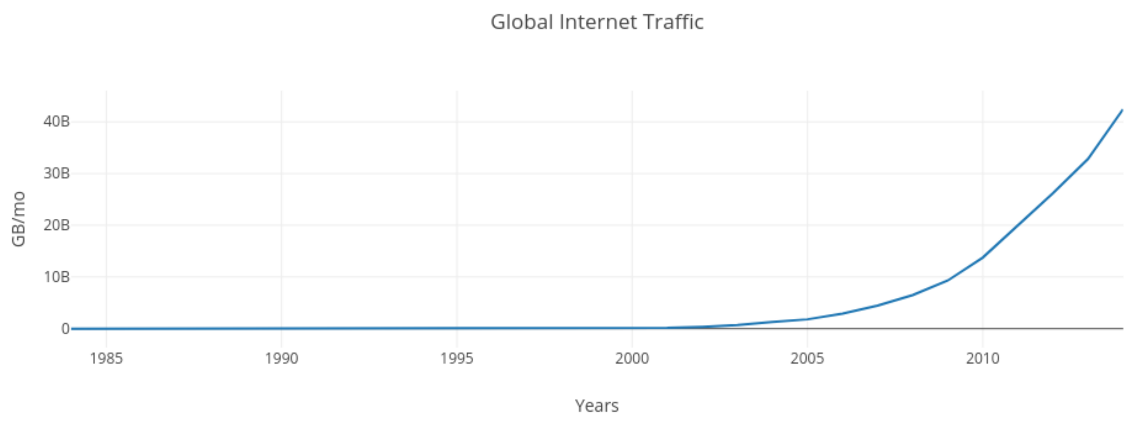


Figure 1-2: Monthly Internet traffic evolution [4]

Today’s Internet relies on a conceptual model commonly known as the TCP/IP protocol suite [5], which basically enables addressing and end-to-end data transmission. The Internet, since its conception in the 1980s, has grown significantly. Figure 1-2 illustrates the growth of the monthly traffic load in the Internet from its beginnings until 2014. The monthly Internet traffic has grown from less than 100 GB per month in the late 1980s to over 40 billion GB in 2014, and it is still expected to grow reaching nearly 400 billion GB in 2022 [4], [6]. It is not only the traffic that has grown dramatically, but also the number of users expanded significantly. Due to the ever-growing number of users, the IP protocol has been updated from its current version 4 [7] to version 6 [8] to accommodate for an increasing number of concurrent terminals through a significantly larger addressing space. It is worth noting that the Internet has experienced significant transformations in the last decades, but its underlying fundamental architecture remains the same, i.e., it depends on the host-centric TCP/IP protocol suite. Due to the fact, that the Internet is still growing, there is pressure put on researchers to focus on projects and consortiums establishing future architectures that fundamentally change the existing one.

One of the most prominent future Internet architectures is Information-Centric Networking (ICN) [9]–[11] with Named Data Networking (NDN) [12]–[13] being its most prominent specification and implementation. ICN addresses content using content identifiers and forwards packets based upon content identifiers instead of content locations (i.e., IP addresses). This shifts the current host-centric Internet paradigm towards a future content-centric one [14]. In ICN, a content requester sends an Interest, which carries a unique content identifier. The content identifier is used by intermediate forwarding nodes to forward the request from the content requester towards any content replica in the network held by a content provider. The content provider is, therefore, a storage entity in the network, where the requested content is stored, while the content consumer is any entity requesting content. ICN enables a consumer to request a given content object in the network without the knowledge of the content location. To this end, ICN specifies two main message types, i.e., Interest and Data messages. The Interest message, sent by the content consumer, contains the identifier of content requested, while the Data message, sent by the content provider to the content requester, is the reply to the Interest message sent. The Data message has to carry content identified with the requested content identifier. It is worth noting that the forwarding scheme for Interest and Data messages is also based upon unique content identifiers.

Services are an essential component of the Internet [15], [16], while typically, users request services rather than content. Services are pieces of software of certain characteristics that can be remotely called and executed using an underlying network infrastructure. As an example, a meteo service may be requested to provide the average temperature for a time period in a selected city, while a file compression service may be used to compress a file provided by the content consumer. A service is usually offered by a remote service provider and is requested by a service consumer. Moreover, the service uses a defined naming scheme that enables the communication between the service provider and service consumer. Furthermore, a request for a service typically possesses input parameters that are provided by the service requester to a given service.

As the future Internet will be even more service-oriented, only a service-oriented future architecture may be accepted by users. As previously mentioned, ICN is foreseen as a prominent candidate to replace the current host-centric Internet architecture, however, it still lacks the support for services. An extension to the ICN paradigm called Service-Centric Networking (SCN) [17]–[19] introduces service support within ICN. SCN does not alter the existing ICN primitives, e.g., Interest and Data messages, but instead extends them with service support, while still benefiting from the existing components of ICN. SCN enables service providers to offer services in the network. SCN Services are software functions that can be consumed by service consumers. Moreover, SCN requirements are any improvements that enable efficient service request handling, such as efficient service processing and session support. The aim of this thesis is to develop the SCN service capabilities in ICN through:

1. Support for load balancing optimizing the computing load in the system.
2. Session support organizing the interactive information interchanges between content consumers and service providers.
3. Appropriate handling of input parameters delivering arguments to remote software functions.

Load balancing is a crucial element of the system. Its task is to improve the overall service processing distribution in the network and organize the service request delivery. It is worth noting that SCN service requests need to be eventually processed by the corresponding service providers. As service processing of requests takes time, the proper load balancing among service providers can improve the overall performance of the system delivering results quicker. Moreover, sessions are vital, especially when the information interchanges need to order a group of incoming requests for a given SCN service in time to produce a valuable result. Sessions are vital, especially if the context creation process is time-consuming. The context establishment process can be time-consuming, for example, in the case of a video encoding service, where the video needs to be fetched before being encoded. In contrast to regular ICN content, SCN service requests need a developed parameter support mechanism, because software functions running as a service require complex types of input arguments.

1.1.1 Research Questions and Methodology

As previously stated, the main problem of ICN is the content-oriented system specification that does not take service support into account. This thesis aims to enhance ICN with SCN service processing of high performance, asking the following research questions:

1. How can we extend ICN to support SCN?
2. How to enhance the system performance with mechanisms that improve service request processing time?

The overall research methodology for different SCN enhancements in this thesis follows the following approach. First, the system performance indicators are established (e.g., the speed of request processing, overhead, etc.). Second, the architecture is defined using the top-down approach. Finally, the system performance is evaluated through simulations using prototype implementations.

We have grouped our contributions into two main parts:

1) The first part proposes an SCN architecture allowing flexible types of input parameters towards requested services. Especially, efficient service request forwarding is provided by using and propagating the knowledge on the available service providers and their load.
2) The second part proposes an SCN session support mechanism that enables a service consumer and service provider to create an execution context, in which a series of operations can be performed. The solution explores multipath routing, which enhances the session support with load balancing and fault-tolerance.

The aforementioned parts are listed and briefly introduced below. The contributions concerning (1) are described in Section 1.2.2, while contributions with respect to (2) are presented in Section 1.2.3. The contributions answer to the research questions stated above.

1.2 Thesis Contributions

1.2.1 Introduction

Primarily, we have approached the SCN requirements from a holistic perspective by developing L-SCN (Layered-SCN) [20] with the goal of supporting services over ICN. It is worth noting again that the regular ICN does not support services, hence, it does not provide the architectural foundations for efficient service requests forwarding and processing. We have, therefore, designed L-SCN extending ICN with enhancements providing service support through the introduction of parameters into the SCN architecture. Furthermore, L-SCN uses a two-layer forwarding scheme combined with supernodes and Bloom filters [21]. The supernodes shall be responsible for the inter-domain communication, while Bloom filters are used to inform the network about existing services in a compact manner. L-SCN aims to optimize the information disseminated in the network on services and resources available. There are different information propagation strategies considered. First, we specified the so-called consumer-driven information propagation approach, in which the service consumers request information about services and resources in the network. Later, L-SCN was extended with event-driven and provider-driven information propagation mechanisms [22]–[23]. In the provider-driven approach, the service providers periodically broadcast information about available services and resources towards the intra-domain network, while the event-driven mechanism propagates service and resource availability information upon certain events, e.g., high load. Our mechanisms improve service processing time and lower the protocol overhead. Furthermore, Named-data Link State Routing (NLSR) [24]–[26] protocol is used to disseminate service information.

Moreover, we have introduced session support in SCN [28]. Our design makes use of hierarchical naming to establish sessions through a two-way handshake. Additionally, we have extended our session support mechanism with multipath routing [29]. This feature enables session requests to be appropriately forwarded towards the service provider for the given session using different paths. The multipath routing capability may be beneficial, for example, in the case of network overload. Moreover, it enhances the session mechanism with fault-tolerance.

Furthermore, we present an alternative source of information based on Distributed Hash Tables (DHT) [30]–[31]. A DHT enables us to receive a significant number of service replicas that satisfy service requests. Especially, it enables us to satisfy service requests, for which there is no knowledge available in the forwarding table.

In the following two sections, we will outline the essential aspects of the main contributions of this thesis. Section 1.2.2 presents the L-SCN architecture and its extensions, and Section 1.2.3 describes the Session Support for SCN.

1.2.2 SCN Architecture

L-SCN

We have designed and evaluated a two-layered SCN architecture named L-SCN (Layered SCN architecture with Supernodes and Bloom Filters). L-SCN enhances ICN with SCN capabilities by enabling efficient SCN service request routing and processing. Our designed forwarding scheme allows service request traffic routing and load balancing. Load balancing is less critical for requesting content, but crucial for service request forwarding because processing is involved in handling service requests. As previously mentioned, L-SCN is a two-layered SCN architecture. It combines an intra-domain and inter-domain communication scheme to minimize the protocol overhead and to optimize the sharing of information about available services and resources in the network. Nodes within the same domain can directly communicate with each other. In each domain, there is at least one supernode, which is responsible for the inter-domain communication and the aggregation of available service and resource information. Supernodes possess an important amount of information about their domains such as available resources and reachability. The only mandatory requirement for a node to become a supernode in a domain is to have a connection link with at least one supernode of another domain. Preferably, supernodes are assumed to have relatively high-capacity links to neighboring nodes.

Furthermore, services in L-SCN can handle any complex type of input parameters. Our design is scalable and does not require a central coordinator. Thus, it does not have a single point of failure. We have not altered, but rather extended the NDN specification. Therefore, content traffic can be routed as usual. L-SCN can easily be enhanced with other SCN requirements, such as authentication and security.

A push and pull communication mechanism using supernodes and Bloom filters assures inter-domain and intra-domain communication. If a node from another domain requires detailed information on some services, it has to request it through the supernode. The supernode can directly answer to the request because it possesses current information about all services available in its domain. The layered approach enhances scalability and lowers down the protocol overhead. In a domain, information about available resources and services is propagated over the entire domain. However, only information about available services is broadcast outside the network with Bloom filters throughout supernodes. The Bloom filter is only propagated further if it is not a subset of a previously forwarded Bloom filter.

Communication Mechanisms for L-SCN

We have extended L-SCN with additional (i.e., provider-driven and event-driven) communication mechanisms and an overlay network [32] that provide an alternative search capability for service request forwarding. This extended architectural approach offers alternative communication strategies and improves usability in multiple dimensions providing enhanced service accessibility and low protocol overhead.

The provider-driven approach is based upon the broadcast of service provider information carried out by the provider; it relies on the push-based paradigm. The service providers have to push their information into the network periodically. This is achieved through special Interest messages that propagate information about available resources and services provided by a given service provider. Three types of Interest messages are used to materialize this protocol: Provider Availability Information (PAI) message, Service Availability Information (SAI) message, and Resource Availability Information (RAI) message. PAI messages are used by service providers to announce node availability information such as joining or leaving a given domain. SAI messages inform the intra-domain network about all available services in the network. RAI messages are sent periodically by the service providers to inform the intra-domain network nodes about available resources at a given time for a given service provider. We envision that a communication protocol built upon these three message types is especially suitable for domains quickly instantiating or disposing services and having highly variable resource utilization.

The event-driven approach propagates service provider information about available services and resources by using an event-driven mechanism. Event-driven means that the propagation is initialized as a reaction to specific events that occur on service providers. In our context, an event is a change in the current status of the service providers. A service provider starts sending information about its available resources and services if significant changes occur in the currently available services or available resources. The term significant is relative to the assessment of the overload performed by a service provider. The service provider propagates its available service information and resource status information in the case of changes with special signaling Interest messages. The service provider uses two messages for this purpose. The messages are called Service Availability Changes (SACs) and Resource Availability Changes (RACs), which communicate a change in service provider status in relation to available services and resources, respectively. RACs and SACs are not sent periodically but are based on the event-driven approach. We have compared the mechanisms against the existing approach.

Furthermore, we added an overlay network based on DHT, which acts as an alternative source to gather forwarding information. This means that nodes can query a DHT network to get information about the availability of a service if the forwarding information is currently not existing in the routing table for the requested service.

Input Parameter Support

Input parameters are essential components of service requests. An input parameter is a piece of data that the service consumer has to provide to the service provider for the given service. To be able to process a service request, the service provider needs the specified parameters. We present an input parameter support scheme that handles short and long types of input parameters for service requests. Parameters of type small are variables such as an integer or a string. Parameters of type long are NDN objects or content stored locally on the service consumers. The designed naming scheme allows us to send multiple parameters, and it uses a hash of the parameter content to distinguish between different parameters. This enables a service Interest request containing different input parameters to be easily distinguished by having a distinct Interest name.

Service Support for NLSR

The NLSR-based (Named Data Link State Routing Protocol) [24] routing scheme named IaDRA-SCN [27] extends NLSR with service support capabilities. In contrast to existing solutions, this solution's design is based upon a link-state routing protocol adapted for Service-Centric Networking. We rely on NLSR, which does not use flooding; instead, it uses a hop-to-hop synchronization mechanism based on ChronoSync [33]. Prefixes, Adjacency, and Resource Availability (e.g., CPU, RAM) for each node are disseminated with Link State Advertisements (LSAs). This propagation allows each node in the network to build the network topology and to identify nodes associated with prefixes. Additionally, the dissemination of resource availability allows us to rank outgoing faces for subsequent service request forwarding decisions. The scheme is suitable for networks with frequently changing service and resource availability due to its link-state nature.

1.2.3 Session Support for SCN with Multipath Capabilities

Session Support

We have specified, implemented, and evaluated a session support mechanism for SCN in the NDN framework to enable requests for a specific session to be forwarded towards the service provider responsible for the session. Without session support, the regular NDN request forwarding scheme forwards subsequent service requests towards different service providers. This behavior is not suitable for subsequent service requests that need to be processed by the same service provider. Sessions are especially important for service requests that require the service provider to create a context. We have chosen to focus on the session support mechanism for SCN due to the importance of service requests to establish a semi-permanent message exchange between communicating parties. A session allows creating an execution context, which is beneficial for the processing of continuous service requests.

Let us again use the video encoding service example from Section 1.1. The service provider, prior to encoding, has to fetch the requested part of the video from the network. By using a session, the service provider creates a context that allows prefetching the required video. Therefore, future service requests using the established session can be processed faster since the service provider has prefetched the requested video.

In our SCN session support mechanism, a service requester can establish a session with a service provider and use the created session during a time period. The existing ICN paradigm has not been altered. As previously mentioned, the service session support design uses a two-way handshake to establish a session between a service consumer and a service provider. To create a session that is routable in the NDN network, we need to derive a unique identifier for a session that is recognized by the provider and the consumer. To build such a session identifier, the designed session support mechanism concatenates two identifiers, that are randomly generated by the service consumer and service provider.

The session support mechanism has three main phases: session establishment, session use, and session termination. To establish a session, the service consumer sends a specially crafted

Interest message towards a service provider. The Interest message contains the session identifier generated by the service consumer. The intermediate nodes forward the Interest message to a given service provider. The service provider replies with a specially crafted message containing the session identifier of the consumer and the newly generated identifier of its own. The concatenation of these two identifiers will form the unique session identifier. Upon forwarding the requests back to the service consumer, the intermediate nodes populate their forwarding tables with an entry that contains the unique session identifier (i.e., the concatenation of the provider and the consumer identifiers).

The intermediate nodes store an entry that contains the unique session identifier, which enables to forward a future Interest request concerning the session to the proper service provider. The service consumer receives the reply and extracts the concatenated service session identifier. From now on, the service consumer can send service requests using the session by integrating the session identifier into the service Interest messages. The intermediate nodes can forward the messages to the service provider of the session by using the session identifier stored in their forwarding table. Upon finishing with the session use, a service consumer can terminate a session by sending a specially crafted Interest message. The intermediate nodes delete their forwarding table entries concerning the session upon forwarding the corresponding session "terminate" Interest message. It is worth noting that we do not build a direct path from a given service consumer to a service provider. However, we use the path selected by the NDN forwarding scheme to assure that subsequent service requests are forwarded to the service provider corresponding to a given session.

Multipath Session Support

The session support mechanism presented in the previous subsection does not provide alternative paths for a given session, i.e., it does not provide multipath routing capabilities. This means there is only a single (initially used) path that can be utilized for communication between the service provider and service consumer. In the case of a link failure or overload, service requests can be lost. Multipath routing seems, therefore, a key for fault-tolerance. By enhancing the session support scheme with multipath support, we solve the limitations of the single path solution.

Figure 1-3 shows the single path session support, the session between the service consumer (red circle) and service provider (green circle) was established through the Nodes 1 and 2. If between Node 1 and 2 (red cross) a link-failure occurs or the link is overloaded due to heavy traffic, then the session can no longer be appropriately forwarded because the session is only known by the path over which it was created. This is due to the ICN forwarding scheme which is based on the content (i.e., session) identifier.

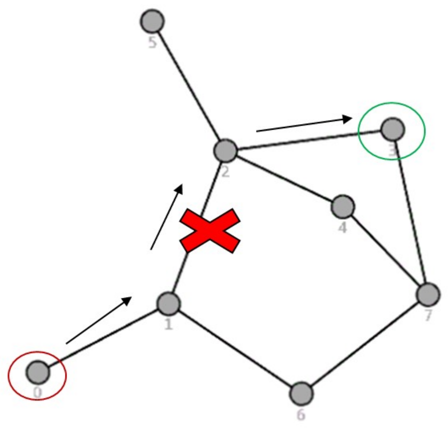


Figure 1-3: Link failure in single path session support

We have designed and evaluated three multipath support techniques for sessions in SCN. The first mechanism is based upon propagating session identifiers within the network using Bloom filters, the second design is based upon the propagation of service provider identifiers, and the third design uses piggybacking for the propagation of service provider identifiers.

The first multipath session management uses Bloom filters to propagate a set of session identifiers in the network. The service provider broadcasts Bloom filters containing session identifiers currently maintained by the service provider. The propagation of Bloom filters is realized through the broadcast of regular ICN Interests (i.e., Broadcast Interests) having a special build Interest name. When a node receives Broadcast Interests, it forwards them over all faces except the Interest's face of arrival (i.e., incoming face). Intermediate nodes store the incoming messages and forward them further. The stored Interest message which contains the Bloom filter with the session identifiers will be used for further forwarding, in the case, an incoming session request that is not known by the intermediate node has to be forwarded further.

The second mechanism propagates provider identifiers instead of session identifiers, which has as an advantage that only one identifier per provider needs to be propagated in the network. The third strategy is a mutation of the second strategy. The substantial difference between these strategies is that we do not broadcast Interest messages to propagate the information. Instead, we only rely on piggybacking to propagate the session information.

1.2.4 Conclusions

This section highlighted the motivations and gave an overview of the main contributions of this thesis. The following chapters will present the related work and contributions. This thesis is organized in the following manner. Related work is discussed in the following chapter (Chapter 2). We present the L-SCN architecture and the extension of the L-SCN architecture with additional mechanisms in Chapter 3. Chapter 4 describes the proposed session support mechanism with multipath capabilities. For the evaluation scenarios, we use selected graphs that resemble the Internet topology at a small-scale. Furthermore, we choose graphs and scenarios that enable us to ensure a fair comparison of the desired metric. Finally, we conclude this thesis and present some future research directions in Chapter 5.

Chapter 2 Background and Related Work

This chapter presents the background knowledge necessary to facilitate the understanding of the thesis and describes the related contributions of previous researchers. We outline the main architectural aspects of NDN, which is the core ICN architecture on which this thesis relies extensively. We also present Bloom filter and Distributed Hash Table (DHT), which are a data structure and a decentralized distributed system used in some portions of the thesis. Moreover, we present the related work that is relevant to the contributions of the thesis.

2.1 Background

2.1.1 Bloom filters

Bloom filters are memory-efficient probabilistic data structures presenting a set of elements in a compact way. Bloom filters are widely used in computer networks [34]–[37]. We use Bloom filters to transmit a large amount of data (e.g., service identifiers) in a compact manner. Bloom filters support two operations, i.e., insert and check. Insert adds an element to the Bloom filter, while check indicates whether a given element exists in a Bloom filter. Bloom filters use a bit vector to store a list of elements; prior to the storage, the elements are hashed with an arbitrary number of hash functions. The Bloom filter can be queried by checking whether a given element exists on the Bloom filter. Bloom filter queries do not produce false negatives (i.e., a Bloom filter cannot deny the existence of the previously inserted elements), but can produce false positives (i.e., a Bloom filter can confirm an element that was not inserted). Figure 2-1 shows the insertion process of a string into an empty Bloom filter. The empty Bloom filter has all its bits equal to 0. In this example, a service identifier “service123” is inserted. Three hash functions are used on insert to set the appropriate bits to 1 in a bit vector of size equal to 12. The check procedure is similar; a given service identifier exists in the Bloom filter if all Bloom filter bits for the corresponding label are set to 1 (i.e., according to the hash functions used).

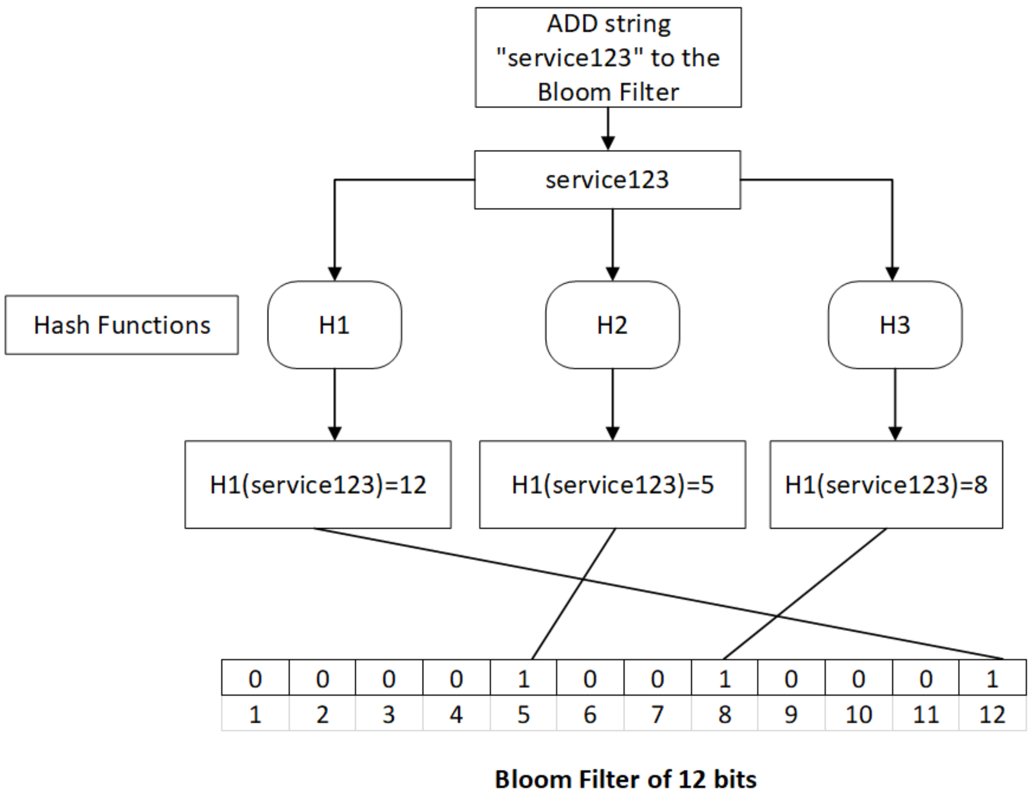


Figure 2-1: Insertion into a Bloom filter

Every Bloom filter comes with a particular trade-off between the size of the bit vector, the number of elements, the probability of false positives, and the number of deployed hash functions. Equation (1) expresses the trade-off, where *m* is the size of the bit vector providing the number of stored elements *n* and the desired accuracy *p*. Equation (2) shows the optimal number of required hash function *k* for a given number of vector sizes in bits and number of elements (i.e., bits per element). The equations show that increasing both the size of the bit vector and the number of hash functions deployed will improve the accuracy of the Bloom filter. Equation (3) displays the approximation of the false positive probability *p* in relation to the number of hash functions *k*, size of the bit vector *m*, and size of input elements *n*.

$$m = -\frac{n\ln(p)}{\ln(2)^2} \qquad (1)$$

$$k = \frac{m}{n}\ln 2 \qquad (2)$$

$$p = (1 - e^{-\frac{kn}{m}})^k \qquad (3)$$

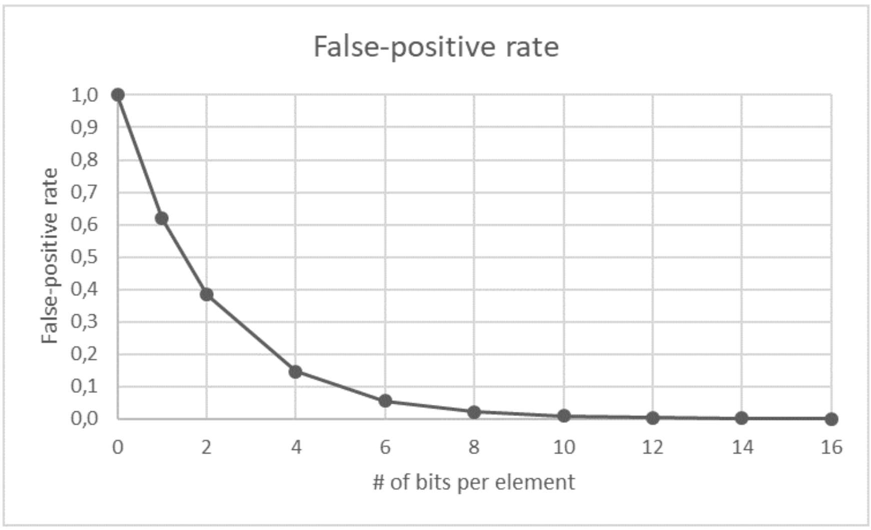


Figure 2-2: Bloom filter false-positive rate and number of bits per element relation

Figure 2-2 shows the relation between the number of bits per element and the false-positive ratio considering an optimal number of hash functions. With a bit per element ratio of 8, the plot provides the probability of a false positive equal to approximately 2%, while for an increased bit vector size of 15 bits per element, the plot displays a false positive rate of approximately 0.1%.

2.1.2 Distributed Hash Table

Distributed Hash Table (DHT) is a self-organizing overlay that stores key-value pairs in a decentralized and distributed manner. DHT permits us to enable DHT-based forwarding. This allows an alternative search for request forwarding if there is no available outgoing face corresponding to an incoming service request.

The DHT overlay network enables any node connected to the DHT network to provide lookup, insert, and delete operations on the distributed stored <key, value> table. Any node connected to the DHT network can retrieve the value associated with a unique key. It can, for example, query for a service identifier and get back a list of IP addresses associated with the service identifier from the DHT network. Chord [38] is an implementation of the DHT system. The list below shows the main characteristics; for detailed information, please refer to [38].

- Chord evenly distributes keys, hence, enforces a natural load balancing
- Chord is scalable; the lookup procedure requires O(log(n)) operations, where n is the number of nodes in the network
- Chord is robust because it automatically adapts to churn, meaning nodes joining or leaving the network (also including node failures)
- Chord has no single point of failure

The DHT network can serve, for example, as a gateway to interconnect two disjoint NDN networks together. This can be beneficial, especially during the deployment phase of the NDN network, when disjoint NDN networks need to be interconnected.

2.1.3 Information-Centric Networking

One of the most prominent implementations of ICN is Named Data Networking (NDN) [39]. In our work, we use NDN, more precisely, we conduct our evaluations in ndnSIM [40]–[42].

NdnSIM is an ns-3 based NDN simulator [43], [44]. Therefore, we will focus on NDN, which will be presented in the next section. Other implementations of the ICN paradigm also exist such as DONA [45], CCN [12], PSIRP [46], PURSUIT [47], and NetInf [48].

Figure 2-3 and Figure 2-4 show the differences between the host-centric networking and information-centric networking paradigms graphically. Figure 2-3 shows the host-centric paradigm with a content server and two content consumers, which request content from the content server. The users must know the address of the content server to reach it. Furthermore, when a connection to the server is established, then the consumer can start requesting the desired content. Figure 2-4 shows a network based on the ICN paradigm, in ICN the users (content requesters) do not need to know the server's address; they only need to know the content identifier. In ICN, the routers (i.e., intermediate nodes) have to take care of the request and forward it to the content replica holding the corresponding content. The content provider replies with a Data message containing the requested content. Finally, the message with the requested content follows the path spanned by the request backwards and reaches the consumer.

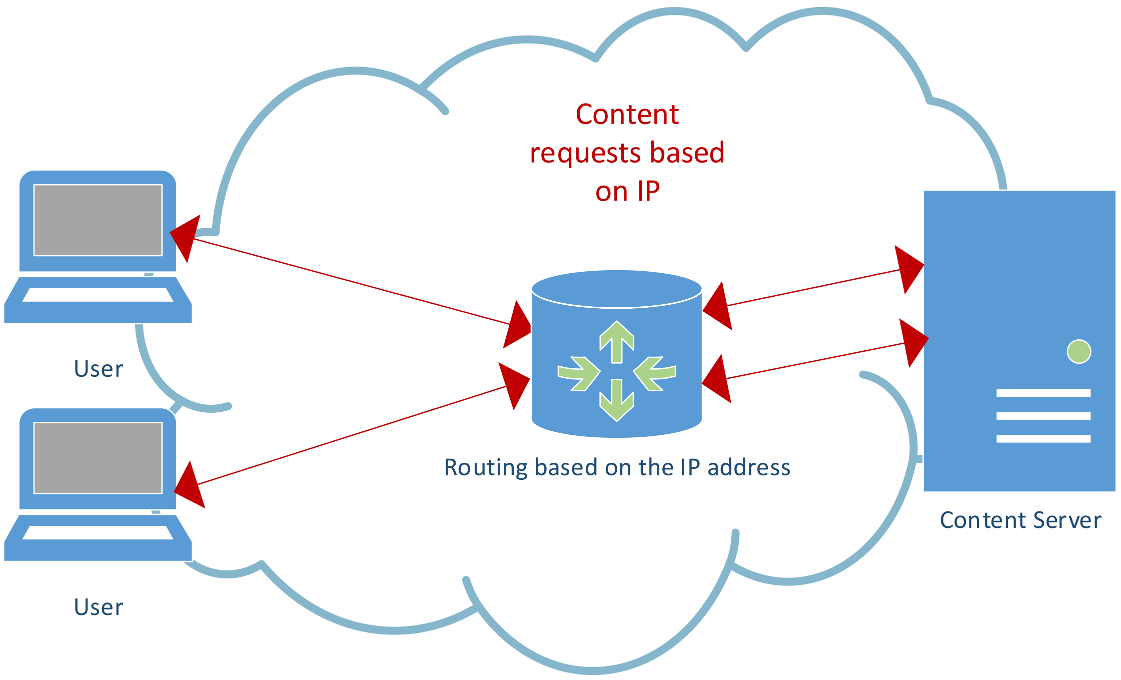


Figure 2-3: Host-Centric Networking

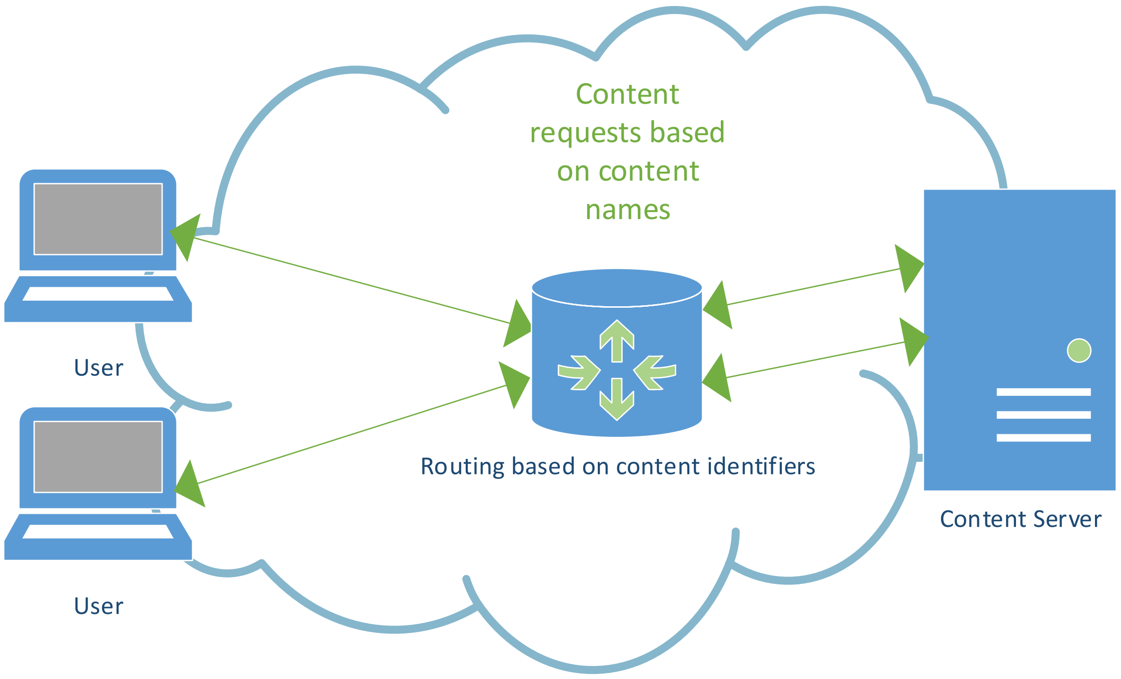


Figure 2-4: Information-Centric Networking

2.1.4 Named-Data Networking

Named-Data Networking (NDN), which has its roots in the CCNx [49] implementation of the CCN [12] architecture, is one of the most popular implementations of the ICN concept. In this work the terms NDN and ICN are used interchangeably. In NDN each object in the network is identifiable by its unique name. The NDN network consists of content providers, content consumers (or requesters), and intermediate nodes. The content provider provides a set of content items. The content consumer can, through messages, request a given content. The intermediate nodes forward the consumer's request towards the provider and forward back the reply from the service provider to the service consumer.

The NDN network has two types of messages: Interest [50] and Data [51]. The Interest message is used by the content requester to send content requests. The reply to a content request is a Data message forwarded to the content requester. The content request is forwarded towards a content replica. The basic idea behind this concept is that a requester can send Interest requests for content without prior knowledge of the content location. In NDN, the content is directly addressable, and the routing is done by using the content identifiers. NDN uses a hierarchical naming scheme for its content naming structure. The content name is composed of a set of components that are separated by a slash (/). The structure of the content names is in the form of "/Conponent_1/Component_2 /…/Comonent_N". The components do not indicate the location of the content but only identify the content. The content name can be aggregated for forwarding decisions, meaning that for the item "/Component_1/…/Component_N", an aggregated forwarding entry can be introduced "/Component_1/…/Component_(N-X)", where $X < N$, matching all entries at higher levels. Interest messages sent for content, also possess a set of attributes [52] such as the lifetime of the submitted content Interest request. The lifetime of an Interest sets the time period, in which a request remains valid in the network.

From the architectural perspective, NDN has three main components that are installed on all nodes in the NDN network: Forwarding Information Base (FIB), Pending Interest Table (PIT), and Content Store (CS). The FIB is a lookup table containing a match between outgoing faces (i.e., interfaces) and prefixes; it enables us to select an appropriate outgoing face towards the object of interest. The PIT stores pending Interests that have been forwarded further but have not yet been satisfied, i.e., the requests have not yet been satisfied with a Data reply. CS is a locally disposed cache that temporarily stores incoming Data packets. CS allows incoming Interest requests to be satisfied directly from the local cache without needing to forward the Interest request further on. Figure 2-5 shows the core elements of the NDN architecture. In Figure 2-5, the FIB table has only one entry, and the face list indicates that the corresponding incoming Interest is forwarded over faces 0 or 1. The concrete way of forwarding an incoming Interest request concerning a given prefix depends on the selected underlying forwarding strategy.

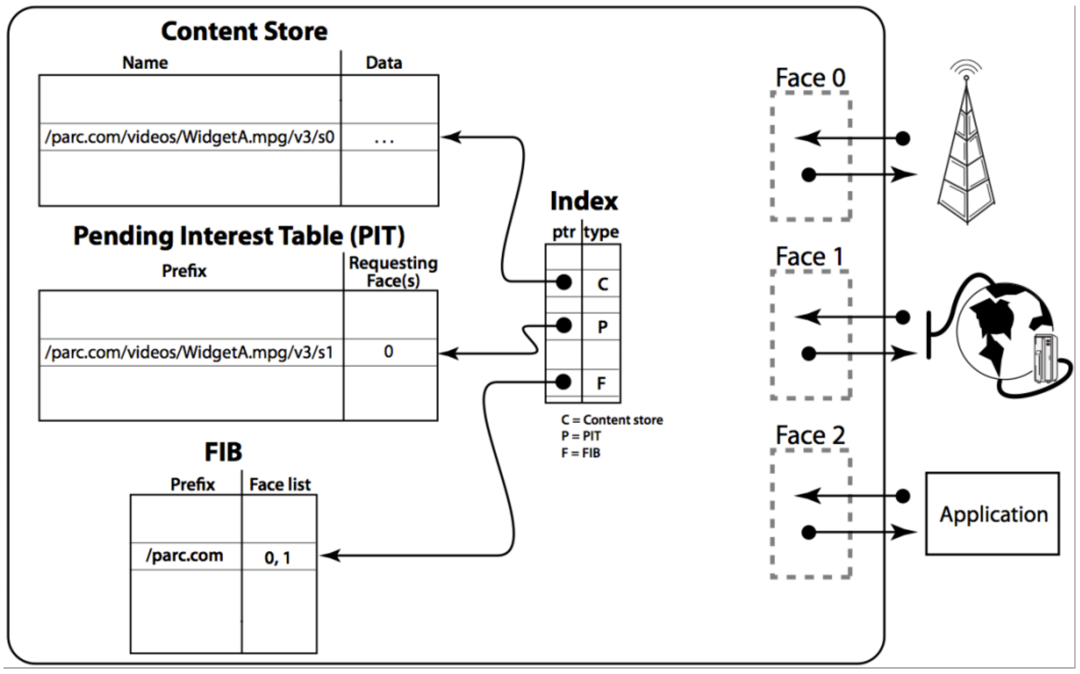


Figure 2-5: NDN architecture [12]

In Figure 2-5, we also see the PIT table with one entry and its corresponding incoming face. The PIT entry indicates that the Interest has been forwarded further, but it has not yet been satisfied with a response. Upon the arrival of a Data reply message, a node will check if an entry in its PIT table exists for the given Data name. If an entry exists, it will forward the data further on through the faces listed in the PIT table for the content name of the incoming Data message. The list of Requesting Face(s) in the PIT table of a given Interest name can contain one or multiple face identifiers, through which an Interest message arrived requesting a given object name.

NDN has a set of forwarding strategies. A forwarding strategy is an algorithmic process that decides on a forwarding face for a given incoming Interest message. The three main forwarding strategies offered by NDN are Best Route, Multicast, and Random [53]. The Best Route strategy forwards incoming messages through the face having the lowest routing cost. The cost of a face for Interest forwarding is determined by considering various metric values such as hop count. The Multicast strategy forwards the incoming Interest request over all faces through which the given Interest prefix is reachable. The Random forwarding strategy forwards the incoming request through a randomly selected face among all existing faces through which the given Interest name is available.

2.2 Related Work of SCN

This section describes the related existing approaches that are relevant to our contributions presented in Section 1.2. We describe the main SCN approaches and point out the elements in these designs that are related to our work.

2.2.1 Service-Centric Networking

Service-Centric Networking (SCN) [17],[18] is an extension of ICN. SCN extends the ICN paradigm towards services but does not alter it. This means that the core principles and architectural mechanisms remain untouched, and content traffic can be handled as usual. Additionally, in SCN, service providers offer services in the network. Services, being functions implemented in software, that can be consumed by service consumers, which send Interest requests for desired services.

2.2.2 Service-Oriented Centralized Solutions

There are many works on service-oriented architectures based on a centralized paradigm, in which a centralized controller possesses a global view of the network to instantiate services and forward requests among appropriate instances. The centralized controller has up to date information about available services and current load. A single point of failure and massive protocol overhead are, however, the main drawbacks of centralized solutions [54]–[56].

2.2.3 SoCCeR

Service over Content-Centric Routing (SoCCeR) [57] is an existing SCN architecture. SoCCeR extends CCN with integrated support for service routing decisions by leveraging Ant Colony Optimization (ACO) [58]. ACO is widely used in computer networks to solve optimization problems [59]–[62]. SoCCeR adds a control layer on top of CCN, which allows managing routing decisions. It modifies the CCN Data and Interest messages to enable a control layer, which is installed on all nodes and periodically broadcast Ant-Interest messages for distinct services. A message can traverse the whole network until arriving at a node serving the corresponding service. The node handling the service replies to the Ant-Interest messages with an Ant-Data message, which contains server status information such as CPU usage or memory consumption. The Ant-Data message makes use of breadcrumb information left by Ant-Interest messages to arrive at the service requester successfully. All forwarding nodes use the server status information gathered from the Ant-Data messages. The information is stored in a specific data structure on the node to alter the content of the forwarding table.

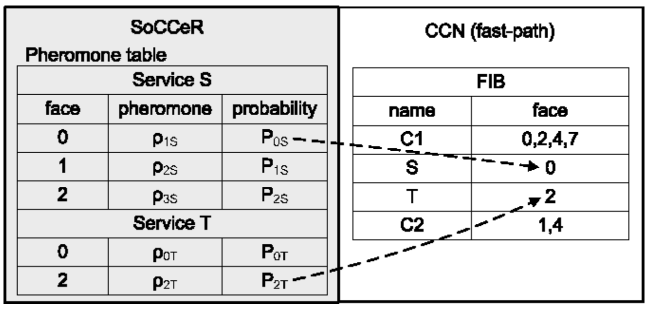


Figure 2-6: SoCCeR node design [57]

SoCCeR handles service and content forwarding in the same way. The only architectural difference is that the face defined as an outgoing face for a given service is updated based on the collected metric values coming from the Ant-Data messages. Figure 2-6 shows the design of a SoCCeR node with the SoCCeR and CCN layers. The CCN layer is not altered, the content and service request are forwarded as usual Interest messages. The SoCCeR layer contains the entries for each service with the corresponding pheromone value for each face and the probability calculated using the pheromone values. For each service in the SoCCeR layer, the highest probability face is set as the outgoing face in the CCN table. In Figure 2-6, the highest probability faces for the services *S* and *T* are set as an outgoing face in the CCN FIB table. In this example, face 0 is set as an outgoing face for service *S,* and face 2 is set as an outgoing face for service *T*. Upon the arrival of the Ant-Data messages, the pheromone table values are updated. If the order of the faces changes in terms of probability, then the respective entries in the CCN FIB table will be updated.

SoCCeR does not possess input parameters and session support capabilities. Hence, stateful services are not supported by SoCCeR's ACO-based design. Another disadvantage is the tremendous amount of Ant messages that traverse the network, which might generate a massive protocol overhead. Other ACO-based ICN routing mechanisms exist ([63]–[67]), which bring additional features, such as energy-efficiency or QoS. However, they inherit the disadvantages of SoCCeR or ACO-based solutions.

2.2.4 CCNxServ

CCNxServ [68] is built on top of CCNx [69], which is an implementation of the CCN architecture. CCNxServ provides NetServ [70], which is a software component extending CCNx with support for services. CCNxServ allows us to deploy services dynamically. A CCNxServ node has to fetch and deploy the service application from the network prior to executing the service request.

In CCNxServ, the process of requesting a service is as follows. The service consumer sends a service request that contains the service name and the content on which the service has to be executed. The service name indicates the name of the service module. The content can only be provided as a CCN regular object fetched from the network. CCNxServ has three main architectural components: CCN network, NetServ, and Service Proxy. The CCN network handles the Interest and Data packets. The task of the Service Proxy is to intercept the service request sent by the service consumer and split it into two distinct requests, which are responsible for fetching the service module with the corresponding content. NetServ is the last component, and its task is to deploy the fetched service module and execute it over the retrieved content. Further, it sends the computation result to the Service Proxy, which in turn forwards the result to the service requester.

CCNxServ has many limitations. Its design supports only services delivered as a single JAR file and implemented in JAVA [71], [72]. Forcing a service to be delivered in one single file and implemented in a specific programming language is quite unrealistic for today's services, where a service could be part of a bigger system (e.g., online banking). Moreover, the deployment of the service may become the leading cause of performance issues. Furthermore, CCNxServ does not fully comply with the CCN architecture because of IP-based NetServ. Finally, CCNxServ supports only CCN content objects as service input parameters and it does not offer any session support.

2.2.5 Serval

Serval [73] introduces a new service layer between the transport and the network layer of the TCP/IP protocol stack. The newly introduced layer enables applications to communicate by using service names. To enable forwarding decisions, Serval builds a dedicated service table to map service names to the corresponding network addresses. In Serval, special service routers are responsible for locating the best available service replica that satisfies a given service request.

Let us focus our attention on Figure 2-7. It shows a Serval service request and the Serval protocol stack abstraction. The figure displays the additional layer named SAL (service access layer), added between the transport and the network layer of the TCP/IP protocol. SAL possesses service and flow tables and uses these two tables to construct a mapping between service identifiers, service providers, flow identifiers, and interfaces. Furthermore, Serval relies on these special service routers to support functionalities such as service discovery and load balancing. Finally, service providers have to announce their availability to the special service router. Let us now consider an example incorporating one service consumer, two service providers, one service router, and one standard router (i.e., a traditional IP router). A service consumer requests a service through a service router, which supports the Service Access Layer. The service router selects the best replica and forwards the request over the standard IP router towards the service provider. The replies of the service provider go towards the service requester directly over the IP router bypassing the service router. Both communicating parties store the flow identifier of the connection to distinguish between different instances of service requests.

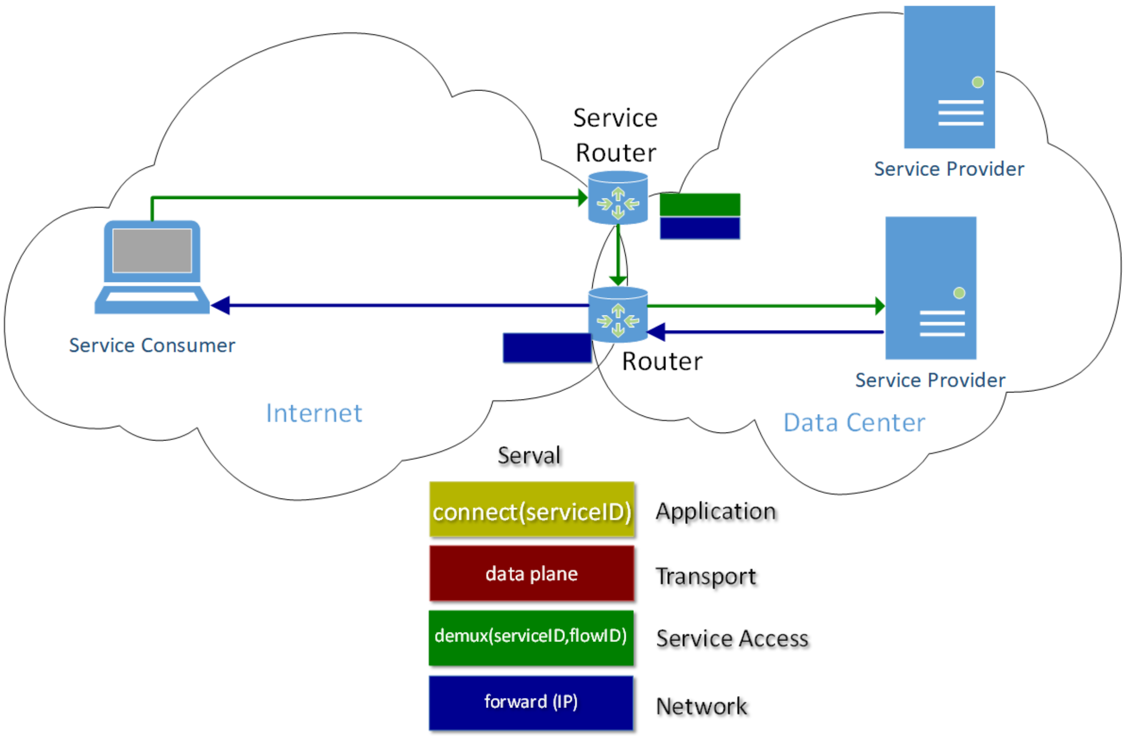


Figure 2-7: Serval service request

Serval fundamentally changes the TCP/IP stack, which makes the integration of Serval into the current Internet architecture difficult. Serval's design is mainly intended for data centers, while the centralized mechanisms in routers introduce a single point of failure. The Serval architectural approach can support function continuity for service requests by extending its designed flow identifier mechanism. However, a session support mechanism has currently not been implemented. Furthermore, Serval does not support input parameters, which is a fundamental requirement of service requests.

2.2.6 NextServe

NextServe [74] leverages CCN by integrating service support. NextServe uses a service composition and a human-readable naming scheme inspired by object-oriented programming with function chaining.

Figure 2-8 illustrates an example of service request processing in NextServe. The example contains three CCN routers, one content provider, two service providers, and one service consumer. In this example, two services are called by the service consumer: /scn/encrypt and /fileManager/zip being the file encryption and file compression services (cf. Chapter 1). The file encryption service takes two input parameters, which are the password and the file to be encrypted, while the input parameter file is of type content object. The Client sends a chained service request that is formatted as following: /scn/encrypt/(\P@ssw0rd",/_leManager/zip /(/university/pro_le.pdf/)/). The chained function call is structured in the same manner as traditional object-oriented function invocation. In this example, the client wants first to compress the file "/university/profile.pdf" and then encrypt it with the password given in the function call (i.e., P@ssw0rd). The CCN routers forward the client request to a service replica (1) offering the encrypt function (i.e., /scn/encrypt/). The forwarding is done as traditional longest prefix match, and the parameter component of the service request is ignored. Upon receiving the service request, the encrypt function service provider parses the parameters from the Interest message. As next (2), the service provider sends a service request for the compression function. The request is again forwarded as a traditional CCN Interest by the CCN routers. Upon receiving the request, the compression service provider parses it and sends a request (3) to gather the file that is set as an input parameter in the request (i.e., /university/pro_le.pdf/). When the provider receives the file (4), it compresses it and sends the result back to the encrypt service provider (5). The encrypt service provider encrypts the compressed file with the password and sends the result as traditional Data message to the service requester (6).

NextServe is similar to CCNxServ and inherits most of its limitations. A service consumer must fetch the required service executable from the network before executing processing. Furthermore, NextServe does not integrate load balancing or session support.

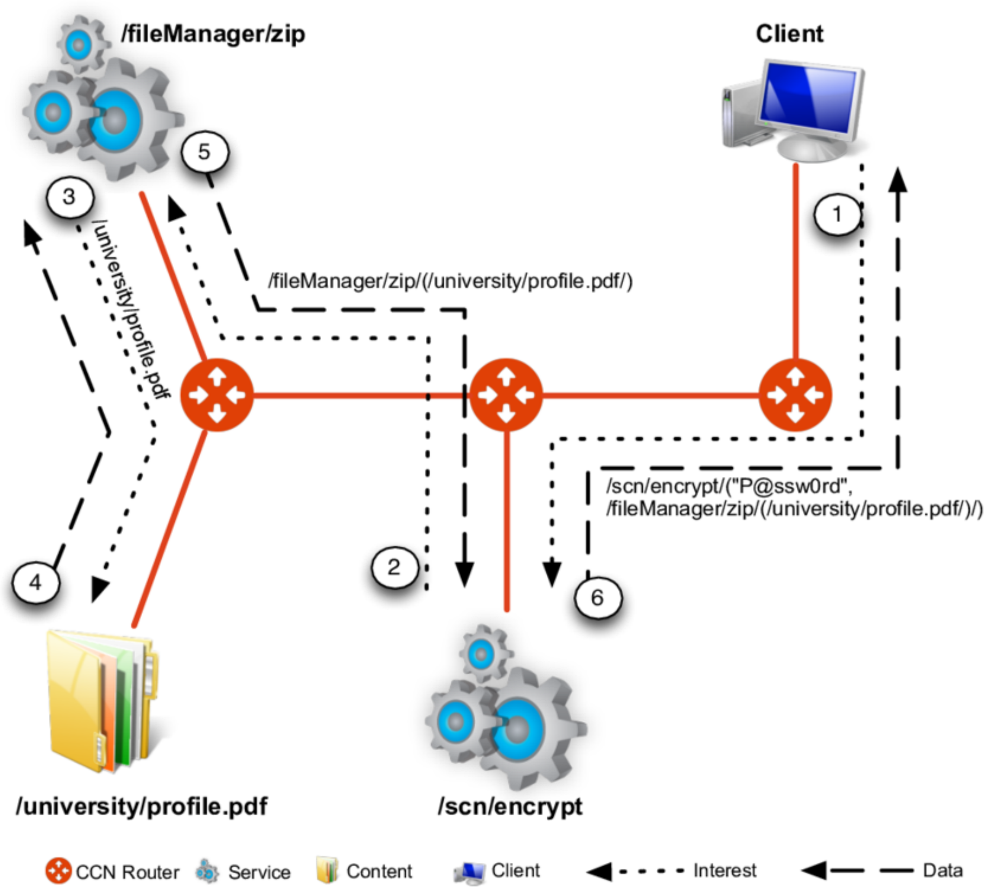


Figure 2-8: Request handling in NextServe [74]

2.2.7 SOFIA

SOFIA [75] is a service-oriented ICN architecture, which decouples communication into two layers: the network and service layers. The network layer is responsible for data transmission using network addresses (e.g., IP), and the service layer provides flexible service processing by using service sessions. The service layer indicates the service name and the instance, to which a given request is addressed. The architectural design of SOFIA is similar to that of Serval. However, it differs in service requests forwarding. As previously stated, in SOFIA, service requests are identified by sessions. SOFIA uses its added service layer to set up a session. However, the use of already created sessions relies on the network layer.

Consequently, after sessions are created, SOFIA forwards packets using destination addresses. This forwarding scheme goes against the fundamental principle of SCN, which is a belief that the future Internet requires a shift from the host (TCP/IP) to the service abstraction level.

2.2.8 NFN

Named Function Networking (NFN [76], [77]) extends the CCN framework by integrating service support. NFN adds two additional layers called NFN Layer and Service Layer on top of CCNx [3]. The NFN layer incorporates a lambda expression engine and is responsible for forwarding decisions. The service layer manipulates the data by running the NFN engine. In NFN, a service request holds the data block and a collection of functions that must be executed over the data. If a service provider is overloaded by service requests, computing can be redirected to the neighboring nodes.

NFN's naming scheme is inspired by the lambda expression language. For example, if we take the application expression funct1(funct2(data)), NFN will represent this expression as [ccn:nfn|/name /of/data | /name/of/funct2 | /name/of/funct1]. It means that the funct2 will be executed on the data, and then the func1 will be executed on the result of funct2. If we take an example of two functions named resize and compress that have to be executed on an image, then the lambda expression will look as follows: /tools/compress /tools/resize /images/img1. The image img1 will be first resized and then compressed. The equivalent NFN naming will look as follows: [ccn:nfn|/images/img1 |/tools/mpeg4|/tools/zip]. The image img1 will be first resized and then compressed.

In NFN, special NFN capable nodes will dissect the incoming NFN Interest request and start the result gathering. The logic provided by NFN is broad, so the result gathering supports different types of scenarios. The NFN enabled node may, for example, separate the data and the application part from the requested NFN lambda expression. Furthermore, they gather the data and the application code. The application code has to be executed over the collected data by finding a location to perform the code execution.

NFN does not provide the support for local arguments and does not provide session support. The naming scheme of NFN based on lambda expression is hard to read, especially when there are multiple function calls. The naming scheme also constrains the number of services that can be supported by NFN. Another drawback of NFN considerably limits the ease of program execution. For example, sophisticated processing requiring the support of custom libraries is practically impossible.

2.2.9 NFaaS

Named Function as a Service (NFaaS) [78] is another dynamic in-network computing approach similar to NFN. NFaaS uses hierarchical naming and unikernels [79] functions instead of lambda expressions. The NFaaS design enables nodes to retrieve and execute functions. NFaaS proposal is inspired by the Serverless architecture [80], which is a new approach in cloud computing that abstracts traditional servers allowing stateless functions to run on any node in the network. The allocation of servers to requested functions is, therefore, at the operator side, and the user does not request resources prior to computing. In NFaaS, rich clients contain most of the application logic. The NFaaS nodes incorporate a component called Kernel Store. The Kernel Store has two main functions, which are the storage of code and decision making on the functions to execute. The information about current executable functions is propagated to the network. The functions move between network nodes. Therefore, specific parts of the network can specialize in selected function sets. The Kernel Store keeps historical usage statistics enabling the decisions on necessary functions to download. NFaaS has two types of requests which are intended to either request execution or request the function itself (i.e., unikernel). For function execution requests, NFaaS uses a specially crafted Interest message beginning with the prefix “exec”. The message request for an execution also specifies the execution type. A type of application execution could be for example a delay-sensitive application that needs to be processed quickly, as it has strict delay constraints.

NFaaS does not provide session support, and it does not consider the relationship that might exist between a set of functions that has to be executed. The execution of a set of functions can generate a lot of traffic and delay, because the functions have to be downloaded prior to execution. As mentioned, NFaaS is based on a Serverless architecture, which has a set of drawbacks such as architecture complexity, migration effort, and updating issues regarding the functions executed.

2.2.10 Conclusions

In the previous subsections, we have presented various works that are related to our research. The table below compares the different approaches by outlining significant drawbacks considering the requirements of the SCN architecture. In contrast to related studies, our work supports these important requirements, including input parameter support of any type, session support, and load balancing of service requests. Furthermore, our work relies on ICN primitives and does not have restrictions on how services have to be implemented.

Drawbacks of different SCN related approaches	
Centralized Solutions	• The main drawback is its single point of failure
SoCCeR	• Does not support sessions • Does not support parameters • Uses ACO, which means it has an uncertain time of convergence, and a tremendous amount of Ant messages traversing the network
CCNxServ	• Its essential components do not rely on ICN, and it uses IP-based NetServ as a central component • Very restrictive requirements on the implementation of services, it supports services delivered in a single JAR file • Services must be fetched prior to their execution
Serval	• Uses a modified IP stack and does not rely on ICN principles • Does not support parameters • Single point of failure resulting from centralized routers

NextServe	• Service must be fetched prior to execution • Does not provide load balancing • Does not support sessions
SOFIA	• It does not entirely rely on ICN primitives. It uses the TCP/IP protocol suite as a major forwarding component
NFN	• Does not support local parameters • Services being restricted to lambda-functions, which prevents from developing sophisticated services
NFaaS	• Does not support sessions • Functions have to be downloaded before execution; a service is composed of a function set, in which each function has to be fetched prior to its execution • Relying on serverless architecture; it inherits drawbacks of the serverless architecture such as issues with the update of existing functions

Table 2-1 Weaknesses of SCN related approaches

Chapter 3 Layered SCN Architecture with Supernodes and Bloom Filters

This chapter presents the architecture and extensions of Layered SCN (L-SCN) based on Supernodes and Bloom Filters [20], [22], [27]. Service requests have to be efficiently delivered by forwarding nodes and processed by service providers. Therefore, efficient message forwarding and processing have to be researched. This chapter provides an architectural approach leveraging mechanisms for service provider information propagation that guarantees efficient service request forwarding and processing.

L-SCN is a two-layer routing architecture. In contrast to existing solutions, we provide significant differences in node clustering, server status information exchange, and enhanced service input parameter support. We cluster the network by introducing domains to significantly reduce the protocol overhead and better share the information about available services or resources in the network. This is achieved through the integration of the push and pull mechanism combined with the introduction of supernodes and Bloom filters. The L-SCN design and evaluation are described in Section 3.1.

Section 3.2 and Section 3.3 present extensions of the L-SCN architecture. We extended L-SCN with additional intra-domain and inter-domain mechanisms. These extensions offer additional communication strategies and improve the usability in multiple dimensions providing enhanced service accessibility and improve the protocol overhead. At the intra-domain level, two service provider information propagation mechanisms named event-driven and provider-driven are derived. The event-driven mechanism propagates service provider information based on events (e.g., high overload). The provider-driven mechanism disseminates service provider information periodically. At the inter-domain level, we introduce a new forwarding information table based on DHT. Nodes can use the DHT overlay as an alternative forwarding source in the case no forwarding entry is available in the FIB table.

Furthermore, we present a routing mechanism that integrates an extended version of the Named-data Link State Routing (NLSR) protocol. We have published this work in [27]. Our system design makes use of NLSR Interest and Data messages to disseminate service provider prefixes and resource availability information within the network.

In the different assessments, we have used different topologies because, in the real world, we do not always deal with the same topology. However, all topologies are of the same type, as they resemble the Internet topology at a small-scale.

3.1 L-SCN Design

L-SCN combines multiple forwarding mechanisms. The default CCN routing mechanism is not modified. Hence, traditional CCN traffic can be routed as usual. In the designed architecture, nodes are clustered and placed within domains. The clustering process is based on proximity, which does not necessarily mean geographical distance, but rather good connectivity by high capacity links.

The proposed communication is organized into inter-domain and intra-domain schemes. Nodes within the same domain can directly communicate with each other. In each domain, there is at least one supernode, which is responsible for the inter-domain communication and the aggregation of available service and resource information. Supernodes possess important information on their domains, such as available resources and accessibility. The only mandatory requirement to become a supernode of a domain is to have a connection link with at least one supernode of another domain. Preferably, supernodes also have relatively high-capacity links to neighboring nodes. A single point of failure can compromise the inter-domain communication in the case a domain possesses only one supernode. A push and pull communication mechanism using supernodes, and Bloom filters assure inter-domain and intra-domain communication. We use Bloom filters because they enable us to store the service identifiers in a compressed manner. As an alternative to Bloom filters, we could use traditional lossless compression methods. However, Bloom filters provide better compression results, as there is no redundancy in a set of service identifiers. Figure 3-1 shows two domains with nodes (grey) and supernodes (black).

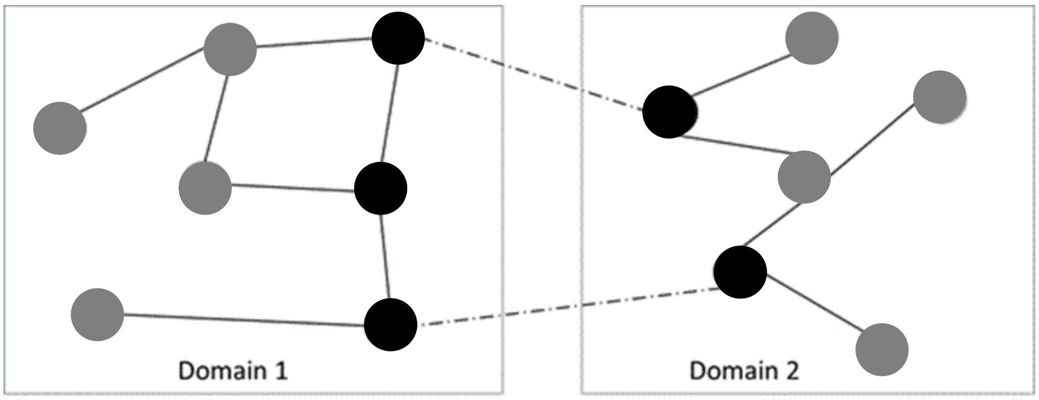


Figure 3-1: Example of two connected domains

If a node from another domain requires detailed information on some services, it has to request it through a supernode. The supernode can directly answer to this request because it possesses fresh information about all services available in its own domain. The following sections explain our architecture, as well as the inter-domain and intra-domain communication mechanisms. We first present the communication protocol among nodes in the same domain in Section 3.1.1, and later on the inter-domain communication mechanism in Section 3.1.2.

3.1.1 Intra-Domain Communication

Supernodes periodically broadcast Interest Information messages (IIM) to all nodes of their domain. Service provider nodes reply with a Data Information Message (DIM). The DIM contains an array with resource availability information and a Bloom filter storing all the services provided by the node. For example, a Bloom filter possessing a maximum false positive error rate of 1% and an optimal number of k hash functions storing 1000 elements requires only around 1 KB of storage space. Table 3-1 and Table 3-2 show the structure of the Interest and Data Information Messages.

Interest Information Message (IIM)	
Name	The name of the IIM message starts with the keyword broadcast followed by the requester's unique node ID (e.g., \broadcast\S1\)
Timestamp	Timestamp of the IIM requester supernode

Table 3-1: Information stored in the IIM

Data Information Message (DIM)	
Name	The name of the DIM message starts with the keyword broadcast followed by the requester's unique node ID (e.g., \broadcast\S1\)
Timestamp	The timestamp of the DIM provider node, it is updated when sent from the cache on an intermediate node
Available Services	Bloom filter storing the available services
Available Resources	Available resources of the node (e.g., CPU, RAM)
Node ID	The intra-domain node identifier (e.g., node4)

Table 3-2: Information stored in the DIM

The DIM follows the path of the received IIM towards the requesting supernode. The information on available services is stored in Bloom filters together with the service status information. All intermediate nodes on the way update their local tables with the information gathered from the reply messages, which is used for service request forwarding decisions.

The Interest and Data Information messages allow the supernodes to discover the intra-domain services, their availability (through what faces), and the server status information (e.g., CPU load, memory consumption). To reduce the protocol overhead in the case a node receives the same broadcast message multiple times, the broadcast requests are equipped with a unique identifier. The forwarding node does not forward the same IIM message over the same face again.

The periodic broadcast IIM messages are sent at the end of a specified time window. There is no explicit synchronization of time windows among different supernodes. However, the synchronization occurs implicitly, since a supernode never forwards incoming IIM messages from other supernodes. Instead, it either starts sending its own IIM message or directly replies with recently received and stored DIM messages. The following paragraph gives an example of how the implicit IIM broadcast synchronization happens, with the help of two supernodes A and B.

A specified time window (e.g., 10 seconds) is used by the supernodes to broadcast IIM requests periodically. B has the following obligations upon receiving the IIM request from A: if B has performed broadcasting previously, it then replies to A with its locally stored DIMs. Otherwise, it begins broadcasting its own IIM requests and puts the request from A in its Pending Interest Table (PIT), because as defined in NDN (cf. Section 2.1), PIT keeps track of received but not yet answered Interest requests. Afterwards, when the DIM responses arrive, it forwards them to A.

In the following, we illustrate the previously presented concept through an example depicted in Figure 3-2. There are two supernodes S1-S2 and five regular nodes N1-N5. If the broadcast time window in S1 expires, S1 starts sending a regular broadcasting IIM request. In this example, the broadcast arrives at N2 and N5, and they reply to S1 with a DIM message. N1 and N5 put the request of S1 into their PIT (\broadcast\S1\) and forward the request to all faces except the incoming face of the requesting IIM. Then, N1 and N3 receive the request from N2, and reply with their DIM.

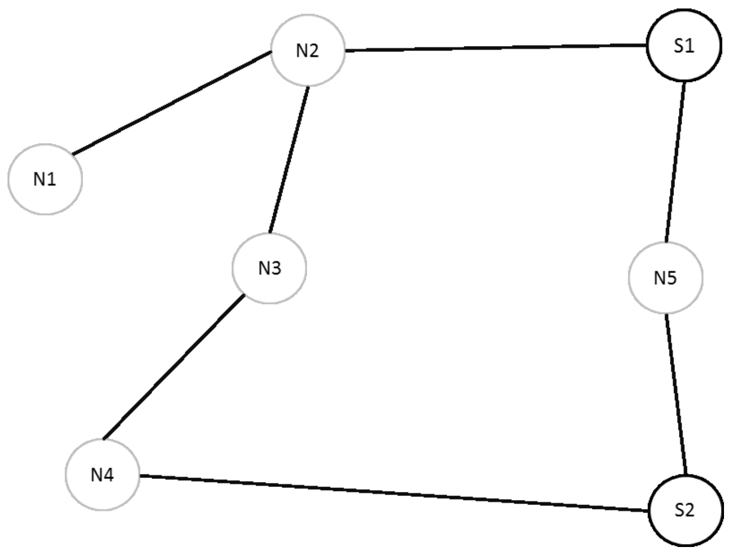


Figure 3-2: Example domain topology with two supernodes and five ordinary nodes

N3 puts the request in its PIT, because it will forward it to N4. When N2 receives responses from N1 and N3, it caches and forwards them to the requester S1. The caching is of high importance, because a broadcast arriving from another node within a short time period can be directly answered with cached DIMs. Caching relieves us from an execution of a costly forwarding procedure. The caching mechanism relies on the standard NDN caching scheme, which provides an efficient mechanism for caching purposes and avoids issues such as stale information.

When S2 receives S1 IIM request through N5 the following can happen. If S2 has recently sent an IIM, it can reply with previously received DIMs. If IIMs were not yet issued, S2 starts its IIM sequence. The S1 IIM request goes to the S2 PIT to forward incoming DIMs to S1 until the IIM broadcast of S2 is completed.

At the end of the supernode broadcast process, the supernodes possess the DIMs of all the nodes in the domain and the faces required to reach a given node. The DIM contains an array, which stores the information on available resources and a Bloom filter with available service names.

Table 3-3 summarizes the information stored by the nodes in a domain. The supernodes have a complete view of the domain, while regular nodes have only partial information, i.e., a regular node only knows the server status information of the fraction of clients, whose DIMs traversed this node.

Information stored by ordinary nodes and supernodes in the domain	
Service Provider's Node ID	Unique identifier of the service provider node (e.g., node4)
Face ID	Face to reach a given node
Available Services Bloom Filter	A Bloom filter containing the services provided by that node
Available Resources	Available resource information on that node

Table 3-3: Information stored in the nodes

Using this information from the DIMs, an upcoming service request can be forwarded to an appropriate node. If S1 has to forward a service request for a given Service A (example topology in Figure 3-2), it searches through available resource information and Bloom filters to find the best face (leading to the service provider with the most available resources). This face will not only be used for request forwarding but will also be stored inside the standard Forwarding Information Base (FIB) table. As defined in NDN (cf. Section 2.1), FIB is a table storing the forwarding face set for different name prefixes. It allows us to forward future service requests immediately without searching through Bloom filter data structures again.

We also employ other protocol optimization techniques, including caching and broadcast message identification to significantly reduce the overhead. 1) If a recent service status is available on a node, the broadcast is not forwarded, while the node can directly answer the request by using its cache. 2) While broadcast messages from a supernode are unique, they will not be processed twice on the same node.

3.1.2 Inter-Domain Communication

We have previously explained a mechanism to disseminate information within the same domain, i.e., intra-domain communication. A similar information broadcasting process disseminating available services and resources to other domains could definitely cause a significant amount of traffic in the network. We, therefore, decided only to broadcast Bloom filters that inform about services available in the domains. The information about available resources is not directly provided and has to be explicitly requested.

Nodes in the network are connected with each other through their faces. An intra-domain face is a face connecting two nodes from the same domain. However, an inter-domain face connects two supernodes from two distinct domains. The inter-domain faces are used to send Bloom filters as broadcast messages that allow supernodes to fill out their forwarding tables, which are used to forward requests along the inter-domain faces that lead towards the requested services.

The received broadcast messages will not always be forwarded. The intermediate supernode will discard the message if the received Bloom filter is a subset of the previously forwarded Bloom filters. Please notice that the forwarding process forwards a Bloom filter, which becomes a union of all the Bloom filters received so far from the neighboring domains.

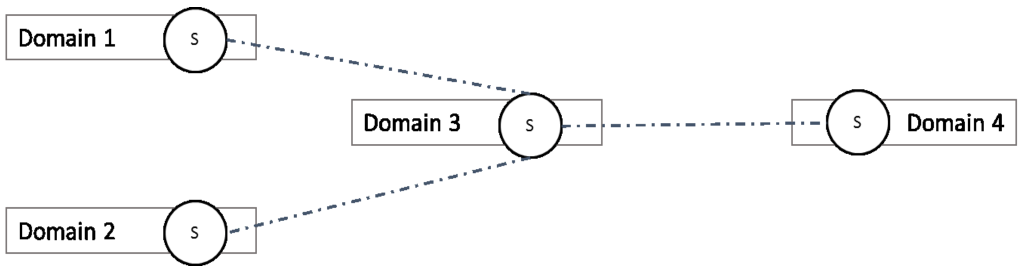


Figure 3-3: Example of four domains linked with supernodes

Figure 3-3 illustrates this process. Domain 3 receives a Bloom filter, which informs about available service names gathered by supernodes of Domains 1 and 2. Let us assume that Domain 1 sent its Bloom filter first. It was immediately forwarded by Domain 3 to Domain 4. Later on, when Domain 2 sent its Bloom filter to Domain 3, it had to go through a check, as it might be a subset of the already received and forwarded Bloom filter of Domain 1. If the Bloom filter of Domain 2 is only a subset of the information already provided by Domain 1, it has to be discarded. Otherwise, Domain 3 will prepare a union of Bloom filters issued by Domains 1 and 2 that will be forwarded to Domain 4.

To summarize, due to this broadcasting procedure, supernodes in the network get the information about inter-domain faces leading to services provided. The information about available resources (e.g., CPU, RAM) on the nodes is, however, not provided at this stage. Moreover, the inter-domain broadcasting process is optimized using two conditions. 1) If the set of services provided did not change since the last forwarding, the supernodes do not have to forward Bloom filters containing the available services again. 2) As already mentioned, if an incoming broadcast is a subset of already broadcasted Bloom filters, it will not be forwarded further.

To receive available resources for a given service, nodes send a Service Resource Information Interest (SRII). The Interest is forwarded through supernodes into the direction of a domain having a given service available. The destination supernode of the target domain will receive the request and reply to it with a Service Resource Information Data (SRID) message for the requested service. The response will follow back the path of the corresponding Interest message, while intermediate nodes will also cache this information for future forwarding decisions. SRID informs the nodes about forwarding faces leading to a node with the best available resources to handle a given service.

3.1.3 Service Requests

In this subsection, we present the intra-domain and inter-domain service requests. The main difference between them is that an intra-domain service request can be equipped with additional information, which is the internal domain nodeID of the service provider. The supernode does the change from one format into the other, i.e., by basically including the nodeID into the request. The insertion of the nodeID into the request is only useful if a large number of nodes and services are available in the domain. This additional information enables the intermediate intra-domain nodes to avoid checking the Bloom filters for forwarding decisions, i.e., finding the outgoing face. Hence, they can directly route the request based on nodeID gathered from the DIM messages. Table 3-4 summarizes the inter-domain and intra-domain service request forwarding.

Service Request Forwarding	
Inter-domain	***Intra-domain***
Based on the service name by using Bloom filters or FIB entries	Based on the service name by using Bloom filters or FIB entries, or by using the nodeID

Table 3-4 Service Request Forwarding

An intra-domain service Interest request with nodeID includes the following fields: the request prefix (e.g., “service”), nodeID (e.g., “node4”), the service identifier (e.g., “getWeather”), and a parametersID (described in the following paragraph e.g., “098f6bcd4621d373cade4e832627b4f6”). Therefore, the full intra-domain request can look like this: /service/node4/getWeather/098f6bcd4621d373cade4e832627b4f6). Additionally, the service request contains an input parameter data structure. Intermediate nodes know how to forward an incoming request, because they possess cached DIMs collected during the broadcast forwarding process. They do not need to search in stored Bloom filters, because the node can simply forward the request to an appropriate face using the nodeID. Again, the aim of the nodeID in the intra-domain request allows intermediate nodes to save time by not using cached Bloom filters for forwarding decisions. An inter-domain service Interest request does not have a nodeID in its name.

A service Interest request also contains a parametersID, which is a unique identifier of the service request input parameters created using a hashing algorithm. A traditional NDN content request is identified by its name, but a request for a service must also include input parameters. To identify the uniqueness of a request, we employ a service name and a hash value (delivered through a hash function) of the associated input parameters. The hash value is stored in a service request name (parametersID). Currently, we employ MD5 hash function to populate the parametersID identifier. This allows us to identify the uniqueness of a service request and enables caching operations similar to regular content caching of NDN. An alternative service request naming scheme solution enabling caching could be implemented by storing all the service parameters inside the request name. This solution, however, does not allow us to request services with complex input parameters.

Forwarding Information Base	
Face ID	Face to reach the node
Name Prefix	Name of the content

Table 3-5 FIB structure

Extended Forwarding Information Base	
Face ID	Face to reach the node
Timestamp	The timestamp of the DIM
Available Services	Bloom filter containing the available services
Available Resources	Available resources of the node (e.g., CPU, RAM)

Table 3-6: EFIB structure

The regular NDN tables (i.e., CS, FIB, and PIT) are not modified, meaning that the content traffic can usually be forwarded. We, however, extended the regular NDN implementation with additional tables to store further information required by our scheme. The table Extended Forwarding Information Base (EFIB) gathers information provided by DIM messages, i.e., the Bloom filters, the corresponding available resources (e.g., CPU, RAM), and the face to access the responding node (i.e., DIM incoming face). The structure is shown in Table 3-6, which is an extension of the original FIB table shown in Table 3-5. This information is used to set an optimal forwarding face in the FIB table, i.e., the information in EFIB is used to set the best forwarding face in the FIB table for future service forwarding. Afterwards, the arriving service requests (e.g., /service/ServiceName/parametersID) can be forwarded along the face leading to the server with the most available resources. When the request reaches a supernode of the domain providing the service, it will be directly forwarded to the service provider node, as the supernode has full knowledge of services, available resources, and nodeIDs in its domain.

As an example of a service request, let us use a hypothetical getAverageWeatherByMonth service, which provides monthly average weather for a given geographical location. It accepts two parameters: month and GPS coordinates. An example inter-domain request would look like /service/getAverageWeatherByMonth/5a105e8b9d40e1329780d62ea2265d8a. Again, the hash value in the request name is generated from the input parameters stored in the Interest packet in the same way as in the case of intra-domain requests. The forwarding of these requests is based on the traditional NDN forwarding mechanism [81].

3.1.4 Input Parameter Support for Service Requests

In this section, we present a scheme providing enhanced parameter support for service requests. Service input parameter support is an essential requirement for service requests. Our service input parameter support scheme includes innovative support for service requests requiring short (i.e., small variables) and long (i.e., NDN objects or content stored locally on the service consumers) input parameters.

We assume that a short parameter does not exceed 1 KB. Please notice that the NDN naming convention allows us to include short parameters in the NDN Interest name. However, in general, NDN is not designed to push a large amount of content with Interest messages from the consumer towards the provider. We also believe that pushing long objects contradicts the core principle of NDN meaning that the content has to be pulled. Therefore, long input parameters should be fetched by the service provider. We aim to integrate a parameter handling scheme that natively handles long parameters (i.e., massive amounts of data). A service Interest name starts with prefix service and specifies that a given Interest is a service request. The prefix is followed by the service identifier (e.g., /service/service1). We further extend the Interest name of the service request with additional pieces of information about the service input parameters. Short parameters can be directly provided within Interest names. For long parameters, we, however, only provide the information (i.e., a pointer) allowing the service provider to fetch the long parameter from the network. Please notice that the long parameter can target a public object residing in the NDN network or a private resource on the service consumer, i.e., not registered in the NDN network. A long parameter identified has to be ordinarily fetched by the service provider as regular content from the NDN network. If the content remains private, the service provider has to directly contact the service consumer and request a given parameter. The subsequent paragraphs describe our input parameter scheme.

Each input parameter uses a combination of two components of the NDN naming convention. The first component indicates the type of the input parameter and its position in the set of parameters. The type could be either short (i.e., *s*) or long (i.e., *l*). For example, *s1* indicates that the first input parameter is of type short, *l2* indicates that the second parameter is of type long. The role of the second component varies depending on the parameter type. In the case of a short parameter (i.e., *s*), the second component carries the content of the parameter. In the case of a long parameter, the second component is a pointer towards the parameter, we call it an input parameter identifier. The input parameter identifier can be a content Interest name or an Interest name containing the result of a hash (e.g., MD5, SHA) [82],[83] computed over the content. When the service consumer uses an Interest name as an input parameter identifier, the service provider is able to fetch the content of the requested input parameter from the NDN network using a regular Interest for content. When the content of the input parameter is private to the consumer (not registered in the NDN network) meaning that there is no prefix available for such content in the FIBs, we implement a special parameter retrieval procedure. In this case, the service consumer provides the hash of the content. With very high probability, the content can be uniquely distinguished through the hash. The service provider must, in turn, use a special Interest (containing the hash of the parameter) to fetch the parameter from the consumer. The Interest message, the Data message, and the forwarding procedure in the retrieval of the long parameter will be explained in subsequent paragraphs.

To illustrate the naming convention for input parameters, let us consider the following example of a service using two parameters. The first parameter is a short value, equal to an integer (e.g., 7). The second is a long parameter (e.g., 0cc175b9c0f1b6a831c399e269772661) content stored privately at the consumer. Such content does not have a prefix registered in the network. The component set appended to the service request providing input parameters is equal in this example to /s1/7/l2/0cc175b9c0f1b6a831c399e269772661. The first two components indicate that the first parameter is of type *s* and has a data value of 7. The second two components describe the second parameter of type long (*l*). The input parameter identifier is a hash computed over the parameter, e.g., a large file that is too big to be provided through the service request Interest name. Therefore, the object needs to be fetched from the consumer by the service provider using a subsequent Interest equipped with the corresponding hash value (0cc175b9c0f1b6a831c399e269772661 in our example). On the other hand, the input parameter identifier can also be a regular Interest name. In that case, if an Interest name is given, the input parameter can be retrieved from the network, because the content prefix exists in the network.

To conclude, the service request Interest name consists of the following elements: service prefix, service identifier, and service input parameter(s). The service prefix indicates that the request is for a service (not for content). The service identifier specifies the requested service. The remaining part indicates the service input parameters of short and long types.

Components of a service request with one input parameter	
Name	***Complete service request with one long hash identifier parameter***
Service prefix	/**service**/zip/l1/4124bc0a9335c27f086f24ba207a4912
Service identifier	/service/**zip**/l1/4124bc0a9335c27f086f24ba207a4912
Service input parameter	/service/zip/**l1/4124bc0a9335c27f086f24ba207a4912**

Table 3-7: Components of a service request

In general, when a service provider receives a service request, it extracts the short type parameter values from the Interest name and fetches the long type parameter content by using the provided long parameter hashes or content names. Let us consider an example in Table 3-7, which presents a final service request. The service name is zip. The zip service request has a single input parameter of type long identified through a hash. The parameter identifier is a hash equal to 4124bc0a9335c27f086f24ba207a4912. The service provider must then request the content from a consumer using a special content Interest. In the following, we will explain the procedure of retrieving long parameters identified through a hash by using a special content Interest. The procedure of retrieving long objects identified through a content identifier will not be thoroughly explained as it is a regular NDN content retrieval explained in [13]. For a regular NDN content, the service provider sends an Interest with the extracted content name received from the consumer.

However, when the parameter is private to the consumer and its corresponding content name is not registered in the network (i.e., no FIB entries), the service provider uses the hash received from the consumer and sends an Interest name to fetch the content: /HASH, in which "HASH" corresponds to the received hash from the consumer. Let us consider the example in Table 3-7. Upon receiving the service request, the service provider realizes that it needs to fetch the parameter from the service consumer by using the hash 4124bc0a9335c27f086f24ba207a4912. The service provider issues a data parameter request using the following Interest name: /4124bc0a9335c27f086f24ba207a4912. Typically, in NDN, regular content and service requests are forwarded using FIB entries. The intermediate nodes need to update their FIB tables with a new entry that allows message forwarding for such special Interest names. This allows the Interest containing the hash sent by the provider to reach the service parameter content at the service consumer. The process of updating FIB tables on the forwarding nodes is described in the next paragraph. Security and privacy concerns are beyond the scope of this work. However, a public-key encryption mechanism can be introduced between a service provider and a consumer for secure communication, e.g., keep private objects protected from third parties.

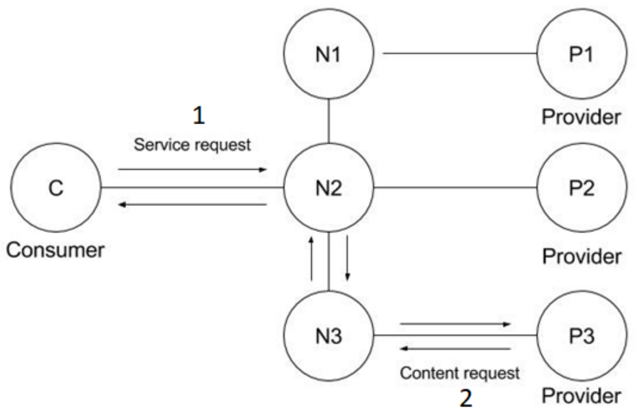


Figure 3-4: Service request sent by service Consumer C followed by the input parameter retrieval request of service Provider P3.

Figure 3-4 illustrates the service request containing an input parameter identifier issued by the service consumer C. Please consider the form of the service request described in Table 3-7. The service request is forwarded to N2 and N3 until it arrives at service Provider P3. The forwarding uses the NDN forwarding scheme, which forwards the Interest according to a selected NDN forwarding strategy. Upon the service request forwarding process, the intermediate nodes verify whether the service request contains hash identifiers for input parameters. In the case hash identifiers are discovered, forwarding nodes alter the necessary FIB tables. In our example, the forwarding of the service request /service/zip/i1/4124bc0a9335c27f086f24ba207a4912, requires the intermediate nodes to update their FIB table with a new entry: /4124bc0a9335c27f086f24ba207a491, that leads towards the service consumer. Such an entry allows subsequent Interest messages for a given input parameter identified by a hash to be forwarded from the service provider towards the consumer through appropriate faces. When the service request sent by consumer C passes through intermediate nodes N2 and N3 (cf. Figure 3-4), N2 and N3 update their FIB tables by adding corresponding entries for the input parameter name (hash). When the request arrives at provider P3, P3 extracts the input parameters from the Interest name and sends an Interest for each input parameter identifier. In our example, the service provider sends an Interest message to fetch the content identified by /4124bc0a9335c27f086f24ba207a4912 from the consumer. N3 and N2 can forward the Interest to Consumer C, because they constructed appropriate FIB entries upon service request forwarding. It is worth noting that an Interest message containing an input parameter uses the same underlying forwarding mechanisms as regular content Interest messages.

Consequently, it benefits from NDN capabilities such as caching. This allows the content of input parameters to be equally cacheable in the network as regular content. Multiple service requests requiring the same input parameters may be effectively handled because input parameters are considered regular content distinguished by a unique naming prefix.

3.1.5 L-SCN Performance Evaluation

In this section, we present an evaluation of our architecture compared to the three forwarding strategies integrated into NDN: Random, Multicast, and Best Route. As its name suggests, the random strategy randomly forwards incoming requests along one of the faces that lead to the service requested. The Multicast strategy forwards the incoming request to all faces leading to the service requested, while the Best Route forwarding strategy forwards a request to the lowest cost forwarding node established by using network-based metric values.

Evaluation Scenario

We have implemented our architecture in ndnSIM. We have modified the Interest and Data packets in ndnSIM (cf. Section 2.1), and new forwarding mechanisms were implemented. The existing information storage was extended with new data structures required.

We have evaluated the designed protocol implementation on a test topology (Figure 3-5) containing 100 nodes divided into 10 domains. This topology was selected, as it resembles the topology of the Internet at the small-scale (with routers of high connectivity in the center and regular leaf-nodes with only one link provided).

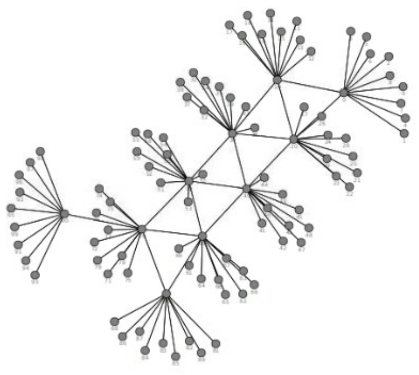


Figure 3-5: The used topology in the evaluation with 100 nodes clustered into ten domains

Node clustering is out of the scope of this work. It was, however, performed by a real-world clustering algorithm, where ordinary nodes get connected to supernodes with the best connectivity. Every domain is equipped with one supernode. We have randomly selected 10 leaf nodes as service consumers and 15 leaf nodes as service providers. Service consumers send service requests using the random exponential function with the mean value equal to 1 second. Service providers in turn process the service requests coming from service requesters. The processing time of a service request is uniformly distributed between 1 and 2 seconds. Moreover, the resources of the processing node are consumed for that period. The simulations measure the service request processing time. We define the processing (or execution) time as the time elapsed from the moment the consumer sends a service request until the processed response is sent back by the service provider. The mean processing/execution time is simply the average execution/processing time of all the service requests. Figure 3-6 shows the processing (or execution) time definition. The scheme shows three nodes: service consumer C, an intermediate node N, and service provider P. The service consumer sends a service request, and the service provider replies to this service request. The processing/execution time is the time t shown on the left side of Figure 3-6.

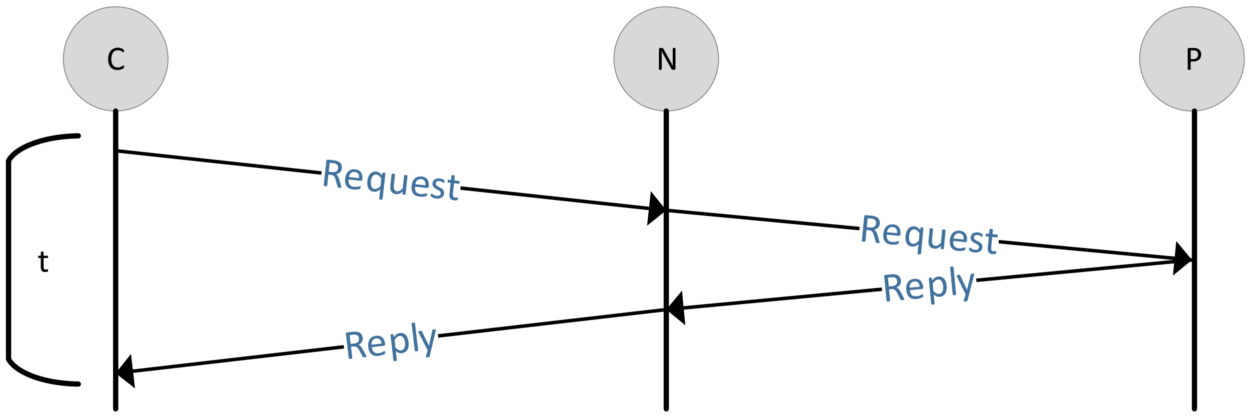


Figure 3-6: The processing time or execution time is the time t

In our scenario, each service consumer sends 100 unique service requests, which gives us a total number of 1000 unique service requests to be processed in the network per simulation round. We have repeated the simulation 10 times to establish the mean execution time and confidence intervals.

Evaluation Results

We compared the processing time of L-SCN with the Random, Multicast, and Best Route forwarding strategies existing in ndnSIM. Figure 3-7 shows the mean processing time denoted by circles, and confidence intervals labeled by horizontal lines. The confidence intervals for the different approaches do not overlap, which suggests significant differences between the mean execution time of the four aforementioned strategies. With high confidence, the mean execution time of the executed scenario resides between the narrow confidence limits.

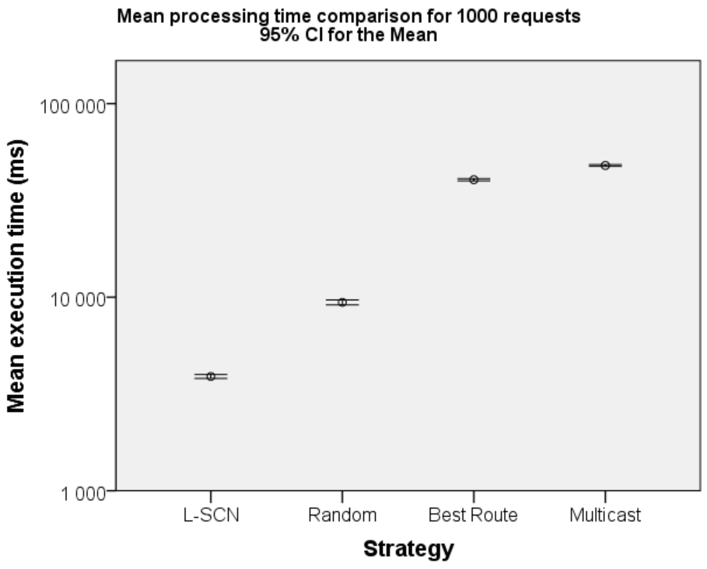


Figure 3-7: Mean execution time for 1000 requests,
comparing L-SCN to the forwarding strategies implemented in ndnSIM (y-scale is logarithmic)

In the evaluation, L-SCN has demonstrated a mean processing time for the 1000 requests of 3895ms (±94ms). It significantly outperforms the remaining three integrated forwarding strategies of ndnSIM. This is realized with the integration of an efficient load balancing mechanism that takes into account server load. In contrast to simple content retrieval, services often require some level of processing at the service provider. Consequently, load balancing is an essential aspect.

The relatively high mean processing time of the Multicast strategy is due to the fact that each service request will reach multiple nodes and, therefore, will be processed multiple times in the network. The Best Route strategy of ndnSIM selects the Best Face for forwarding based on network-based metric values (e.g., hop count, delay). It was developed mainly for content forwarding. Figure 3-7 proves that it is not possible to achieve efficient load balancing for service requests with this strategy. The Random strategy of ndnSIM was the most efficient scheme providing a mean processing time of 9404 ms (±280 ms). Nevertheless, L-SCN outperforms it by a factor of 2.4 with a mean processing time of 3895 ms (±94 ms). The evaluation of our implementation in ndnSIM shows that L-SCN works as expected and produces better results when compared to other prominent contributions in this domain.

3.2 Communication Mechanisms for Provider Information Propagation

In this section, we extend our two-layered L-SCN architecture with additional mechanisms for service provider information (i.e., available services and resources) propagation and an additional forwarding source for service requests.

The initially specified intra-domain information propagation mechanism (cf. Section 3.1.1) uses a consumer-driven technique to gather service provider related information such as available services and resources. We introduce two mechanisms for intra-domain service and resource information propagation: provider-driven and event-driven service provider information propagation. In the provider-driven approach, the service provider nodes periodically broadcast information about available services and resources towards the intra-domain network. This enables the nodes in the domain to get knowledge about existing services and available resources in the domain. The initially designed mechanism is service consumer-driven, meaning that nodes periodically request service and resources availability information from the service provider through Interest requests. The second information propagation approach (i.e., event-driven) does not rely on any periodic broadcasting, but instead sends service and resource availability information only as a reaction to a particular situation, which can be either a change in the provided services or a variation in available resources. This mechanism rarely propagates provider information to the network and is especially characterized by low protocol overhead. Table 3-8 summarizes the three information propagation mechanisms.

Service Provider Information Propagation Mechanisms	
Consumer-Driven	The information about available services and resources are propagated periodically through a pull-based mechanism
Provider-Driven	The information about available services and resources are propagated periodically through a push-based mechanism
Event-Driven	The information about available services and resources (i.e., available or busy) are propagated upon an event through a push-based mechanism

Table 3-8: Service provider information propagation mechanisms

As explained in Section 3.1.2, inter-domain communication is realized through supernodes based on the information on available services stored in the forwarding tables. The service request is discarded if there is no routing entry describing a given service. It is important to provide an additional source of service search in the case a request cannot be satisfied in the NDN network. Therefore, we allow supernodes to act as a gateway between NDN and a DHT overlay network. DHT enables a distributed storage of service names and the corresponding reachability information. Supernodes can use either the NDN network if the service is found in NDN forwarding tables or a DHT overlay to retrieve forwarding information about a currently available service provider. This enables the satisfaction of a larger fraction of requests by accessing an additional network through DHT. This means that L-SCN architecture has access to a DHT overlay network, that can be used by supernodes to search for providers that can forward a service request by using the DHT network.

3.2.1 Provider and Event Driven Service Provider Information Propagation

Currently, the L-SCN architecture uses a consumer-driven approach to propagate service provider related information in the network. The consumer-driven mechanism propagates information based on the request-response principle, this mechanism is presented in Section

3.1.1. This section presents the two newly designed routing mechanisms for provider information propagation: provider-driven and event-driven. In the provider-driven mechanism, a service provider periodically propagates service and resource availability information in the intra-domain network. The event-driven mechanism propagates service and resource availability information upon certain events, e.g., high resource utilization on a service provider. The provider-driven mechanism is more accurate, because it sends information with high granularity. The event-driven mechanism is typically slightly less accurate, while it sacrifices high information granularity to introduce lower overhead.

Selection of the Service Provider Information Propagation Mechanism

Different domains can deploy arbitrary intra-domain propagation mechanisms. The selected mechanism should be adapted to the characteristics of a given domain, such as the number of nodes, supernodes, services, and variance of resource utilization. The network conditions of the domains can dynamically change. Therefore, the propagation mechanism can be selected through careful analysis of the current network condition. The selection of the deployed propagation mechanism and resource-related strategy can be accomplished by supernodes, while they possess substantial information about the intra-domain activity. Using the collected information, supernodes can select the best dissemination strategy to be used in the network dynamically.

Provider-driven Mechanism for Service Provider Information Propagation

The first strategy for service provider information propagation relies on a push-based paradigm. The service providers have to push their information to the intra-domain network periodically. This is achieved through special Interest messages that propagate information about available resources and services provided by a given service provider. Three types of Interest messages are used to materialize this protocol: Provider Availability Information (PAI) message, Service Availability Information (SAI) message, and Resource Availability Information (RAI) message. PAI messages are used by service providers to announce node availability information such as joining or leaving a given domain. SAI messages inform the intra-domain network about all available services in the network. RAI messages are sent periodically by the service providers to notify the intra-domain network nodes about available resources at a given time for a given service provider. We envision that a communication protocol built upon these three message types is suitable for domains quickly instantiating or disposing services and having highly variable resource utilization.

Protocol Messages

The following sections will present the three protocol messages: PAI, SAI, and RAI.

Provider Availability Information (PAI): The service provider uses the PAI Interest messages to communicate important events such as joining and leaving the network. Again, the mechanism is push-based. Therefore, no reply messages to the Interest messages are sent. The PAI message uses a specific naming convention. It is composed of four parts. It starts with two keywords "bcast" and "pai". They indicate that a given message is of the broadcast type (i.e., bcast) and carries PAI signaling. The first two components are followed by the service provider's unique identifier. The message name concludes with the desired signaling message. There are two key PAI signaling messages: join and quit. A join signaling message is sent by a service provider to announce within a domain that the service provider joins this domain. The quit signaling message is sent by a service provider to indicate that the service provider quits the domain. The PAI message structure is given by the following naming convention: "/bcast/pai/[providerID]/[join or quit]". In particular, a join signaling message looks as follows: /bcast/pai/63203bae7501acfe26246efab348c03f/join, where 63203bae7501acfe26246efab348c03f is the identifier of a service provider. Upon forwarding of the join PAI, the intermediate nodes populate their FIB tables acknowledging the presence of a new node in the system and configuring faces leading towards this destination (i.e., using the PAI face of arrival). In the case of a quit signaling message, the intermediate nodes remove all entries corresponding to the given provider from their FIB tables. Let us consider the following example. Figure 3-8 illustrates a domain composed of 8 nodes. Node 1 (in grey) is a service provider, which joins the network and originates a PAI join message. As shown in Figure 3-8, the PAI message is propagated through the entire domain. Therefore, the FIB tables

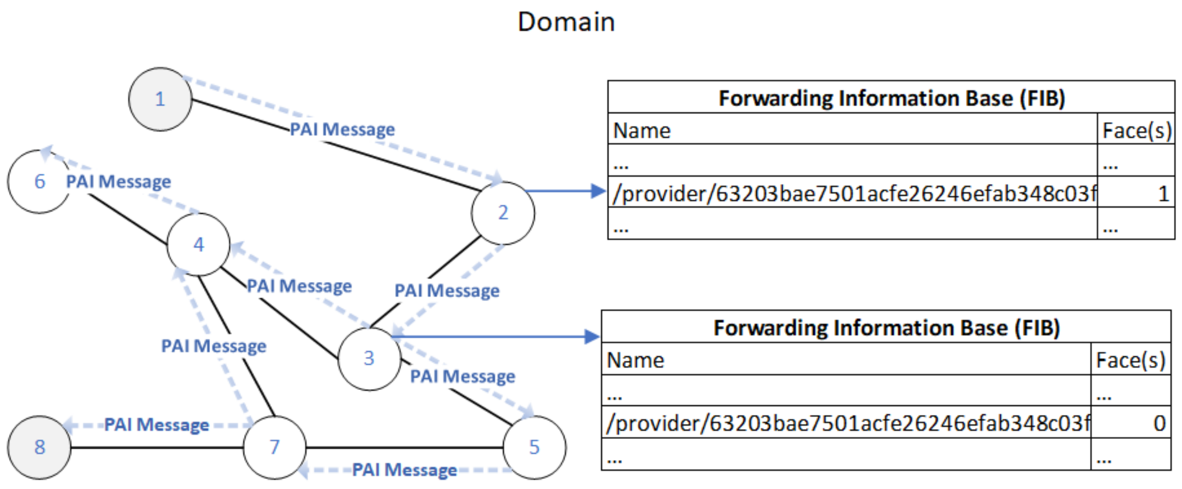

Domain
1
2
3
4
5
6
7
8
PAI Message
Forwarding Information Base (FIB)
Name
Face(s)
...
/provider/63203bae7501acfe26246efab348c03f
1
0

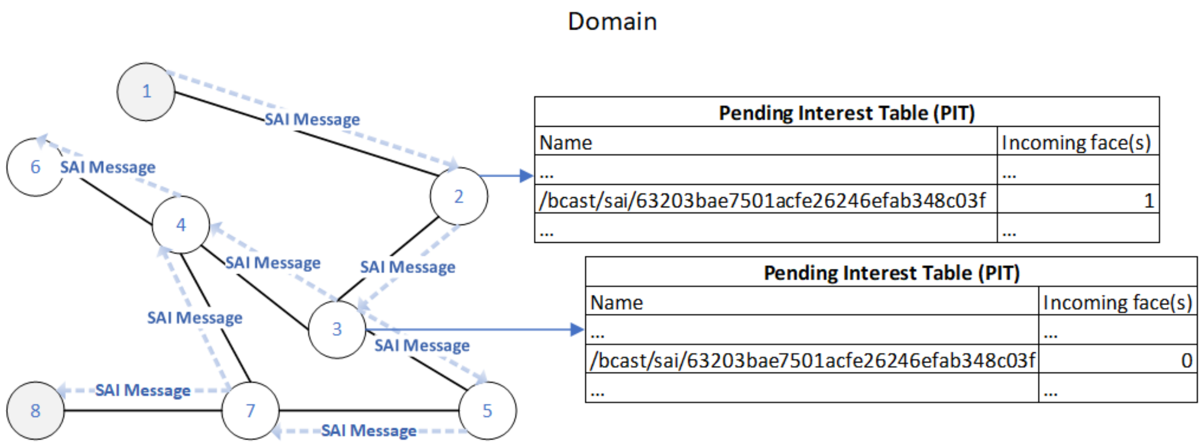
Figure 3-8: PAI message propagation

of all the nodes in the domain are populated by entries indicating the face, through which the newly joining service provider can be reached. Furthermore, Figure 3-8 displays the FIB tables of nodes 2 and 3. The inserted FIB entry begins with the keyword ”provider” followed by the provider identifier gathered from the PAI message. In the case of a quit signaling message, the nodes remove all their FIB and PIT entries corresponding to the quitting service provider. The PAI messages enable proper joining and leaving operations of service providers. The PAI content structure is shown in Table 3-9.

Provider Availability Information (PAI)	
Prefix	Prefix indicating that this is a PAI broadcast message
Provider identifier	The unique provider identifier
Signaling Message	The join or quit action keyword

Table 3-9 PAI structure

Service Availability Information (SAI): The service providers periodically broadcast the SAI Interest messages. They carry a Bloom filter containing the currently available services offered by a given node. The SAI Interest name follows a specific naming convention to indicate that the given Interest is of the SAI type. The SAI message starts with the prefix ”bcast” followed by the keyword ”sai”. The prefix ”bcast” again indicates that this is a broadcast Interest message, while the ”sai” keyword specifies the SAI request type. The prefix and keyword are followed by the identifier, which indicates the originating service provider of the SAI message. The provider identifier is a randomly generated identifier that uniquely identifies a given service provider in the intra-domain network. The Interest message also holds a timeout, which indicates the validity period of a given Interest message. As regular Interest messages, the SAI messages are stored in PIT tables of the intermediate nodes prior to forwarding. A node receiving an SAI message broadcasts it through all faces except the message face of arrival. Nodes receiving multiple messages from distinct service providers populate the PIT table by adding a new entry for every encountered service provider. Figure 3-9 shows a domain with two service provider nodes 1 and 8 (grey circles). The service providers broadcast SAI messages to the entire domain. Figure 3-9 shows the SAI broadcast of the service provider 1 and the resulting population of FIB tables of nodes 2 and 3. The SAI Interest message is propagated to the entire domain, while the carried nonce guarantees loop freeness of the broadcast operation. The PITs of node 2 and node 3 for our previous example are shown in Figure 3-9. Upon forwarding of the SAI message, all nodes populate the PIT tables using the incoming Interest message. Nodes 2 and 3 (i.e., as other nodes as well) use the entry to make a forwarding decision if the FIB does not contain forwarding information for a given service request. The PIT contains the Interest message with the corresponding outgoing face leading to every service provider. An SAI PIT entry is linked to a service provider by the unique identifier carried in the Interest name. In Figure 3-9, the PIT table of node 2 contains a new entry that has been added upon the forwarding of the SAI Interest. The entry follows the SAI naming convention, in which the unique identifier of a given node is carried as the last component. The second column of the PIT table contains the incoming face or faces of the Interest message. Using this information, a forwarding node can link the Interest to its sender and also find out a face, over which a given sender is reachable. The structure of the SAI message is shown in Table 3-10.



Pending Interest Table (PIT)	
Name	Incoming face(s)
...	...
/bcast/sai/63203bae7501acfe26246efab348c03f	1
...	...

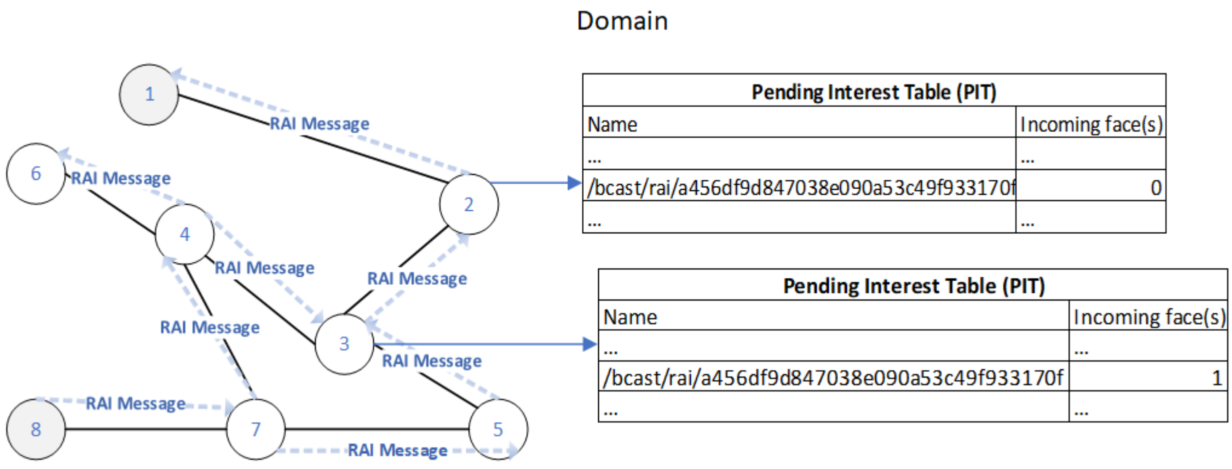
Pending Interest Table (PIT)	
Name	Incoming face(s)
...	...
/bcast/sai/63203bae7501acfe26246efab348c03f	0
...	...

Figure 3-9: SAI message propagation

Service Availability Information (SAI)	
Prefix	Prefix indicating that this is an SAI broadcast message
Provider identifier	The unique provider identifier
Timestamp	Timestamp indicating the validity period of the message
Available Services	A Bloom filter containing the services provided by the service provider

Table 3-10 SAI structure

Resource Availability Information (RAI): RAI Interest messages are propagated independently of PAI and SAI messages. RAI messages contain resource availability information (e.g., CPU, GPU, RAM). Service providers periodically broadcast RAI messages within the entire domain. A RAI message contains the current state of resource availability on a given service provider. Similarly to SAI messages, RAI messages follow a defined naming convention. They start with the keywords ”bcast” and ”rai” followed by the service provider identifier. Figure 3-10 illustrates the RAI propagation process originated by service provider 8. The RAI message is propagated through the entire domain. The intermediate forwarding nodes save the RAI message in the PIT tables and broadcast it further. The intermediate forwarding nodes broadcast a RAI message among all faces except the message face of arrival. In this example, the last component of the RAI message is a456df9d847038e090a53c49f933170f, which is the service provider identifier of node 8. The structure of a RAI message is shown in Table 3-11.



Pending Interest Table (PIT)	
Name	Incoming face(s)
...	...
/bcast/rai/a456df9d847038e090a53c49f933170f	0
...	...

Pending Interest Table (PIT)	
Name	Incoming face(s)
...	...
/bcast/rai/a456df9d847038e090a53c49f933170f	1
...	...

Figure 3-10: RAI message propagation

Resource Availability Information (RAI)	
Prefix	Prefix indicating that this is a RAI broadcast message
Provider identifier	The unique provider identifier
Available Resources	Available resources of the service provider (e.g., CPU, RAM)

Table 3-11 RAI structure

Service Interest Forwarding

Figure 3-11 illustrates the forwarding process of an incoming service request Interest message. First, the nodes check whether there is an entry in the FIB table for the requested service. If a FIB entry is available, the request will be forwarded using the FIB table entry. If no entry is available in the FIB table, the node will query the stored Bloom filters to find out which service providers offer a given service. The service request is then forwarded to a service provider having the highest amount of available resources. A FIB entry for the corresponding service is added to the FIB table. This enables further requests for the given service to be directly satisfied by the FIB table. However, the entries in the FIB tables are updated when new SAI and RAI messages arrive. Furthermore, the FIB entries that have not been used for a certain configurable amount of time are removed from the FIB table. This enables keeping only relevant and recently requested services in FIB tables.

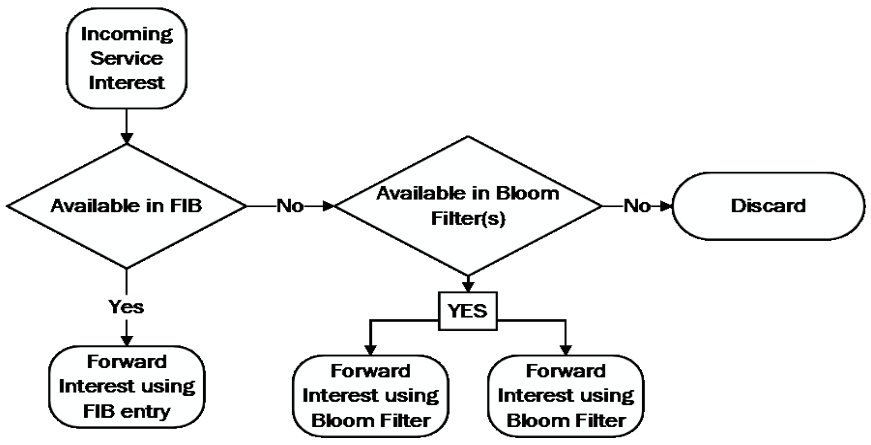

Incoming Service Interest
Available in FIB
No
Available in Bloom Filter(s)
No
Discard
Yes
YES
Forward Interest using FIB entry
Forward Interest using Bloom Filter
Forward Interest using Bloom Filter

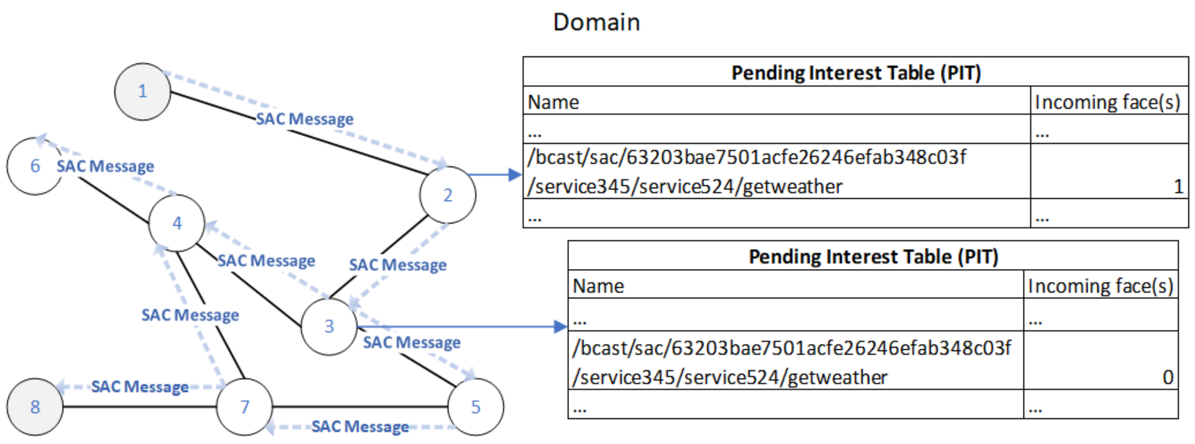
Figure 3-11: Interest forwarding decision for the provider-driven mechanism

Event-driven Mechanism for Service Provider Information Propagation

Protocol Messages

The Event-driven mechanism propagates service provider information about available services and resources based on an event-driven approach. Event-driven means that the propagation is initialized as a reaction to specific events that occur on service providers. In our context, an event is a change in the current status of the service provider. A service provider starts sending information about its available resources and services if significant changes occur in the currently available services or available resources. The service provider propagates its available service information and resource status information in the case of changes with special signaling Interest messages. The service provider uses two messages for this purpose. They are called Service Availability Changes (SACs) and Resource Availability Changes (RACs), which indicate a change in service provider status in relation to available services and resources, respectively. RAC and SAC messages are not sent periodically but are based on the event-driven approach.

Service Availability Changes (SAC) are Interest messages sent by a service provider to inform the network about updates on available services. The service providers have two possible data structures to store the list of available services in the SAC message. The service provider can send the list of available services provided as raw data or Bloom filters. The service provider node decides what data structures shall be used in the SAC message. The choice of the data structure mainly depends on the number of available services and the frequency of changes in offered services. When a service provider joins the network, it either sends the available services as raw data or Bloom filters. Further, the SAC message is not broadcast periodically, but its transmission is based on an event-driven approach. There are two events that may occur with respect to service creation, namely add and remove, indicating the creation or disposal of a service respectively. Upon a service creation event, the service provider either broadcasts a SAC message requesting the network to recognize the newly created service or a Bloom filter containing the set of offered services on a given service provider. In case of a service disposal event, the service provider either broadcasts a SAC message indicating the disposal of a given service or a Bloom filter with the new reduced set of offered services.



Pending Interest Table (PIT)	
Name	Incoming face(s)
...	...
/bcast/sac/63203bae7501acfe26246efab348c03f /service345/service524/getweather	1
...	...

Pending Interest Table (PIT)	
Name	Incoming face(s)
...	...
/bcast/sac/63203bae7501acfe26246efab348c03f /service345/service524/getweather	0
...	...

Figure 3-12: SAC message propagation

Message handling of the service “add” and “remove” events is similar. Upon an add event, the SAC message will be immediately sent to the network. The immediate broadcast of the message allows the entire domain to recognize a new service on a given service provider quickly. Upon the service removal event, the SAC message is only sent when the disposed service was recently solicited. Otherwise, the message will be sent upon receiving a request for the service. The SAC Interest message adopts a specific naming convention composed of five parts. The SAC Interest name starts with the keywords ”bcast” and ”sac” (/bcast/sac/)

indicating that this is a broadcast message carrying a SAC Interest. The next component indicates the nature of the SAC signaling message, which can be of type ”add”, ”remove”, or ”bloomfilter”. The ”add” keyword indicates that the Interest handles the service creation, the ”remove” keyword specifies that the Interest contains disposed service identifiers, while the ”bloomfilter” keyword indicates that the Interest contains a new Bloom filter with the set of currently available services on a given service provider. The fourth component contains the provider identifier. Finally, the fifth component is provided or neglected, depending on the nature of the SAC message. It is neglected in Bloom filter SAC messages; in this case, the Interest message, looks as follows: /bcast/sac/[providerID]/bloomfilter/, where [providerID] is the service provider identifier. In the case of add or remove SAC signaling messages, the last part contains the list of available services as name components. For example, for an add message the Interest name is structured as follows: /bcast/sac/ [providerID] /add/getweatheravg/getweatherinfahrenheit/getweather. It contains ”bcast” and ”sac” to indicate a SAC broadcast message. The third component is the identifier of the service provider node followed by the keyword ”add” indicating that this is a SAC message of type add. The following components (getweatheravg, getweatherinfahrenheit, getweather) form the set of available service identifiers. The corresponding remove SAC message has a similar structure, but remove replaces the add keyword. The SAC messages sent by the service providers are propagated across the entire domain. Before forwarding a SAC Interest message, the intermediate nodes store the Interest messages. Upon an add or remove SAC message, the respective service identifiers transported with the Interest message are added or removed from the existing table entries. If no entry exists for a given service provider, a new entry will be created. Figure 3-12 displays the SAC propagation of service provider 1. Node 1 broadcasts the SAC add message containing three services. The Interest message is propagated through the whole domain. Before forwarding the SAC message, intermediate nodes store or extend the PIT table entry corresponding to a given service provider. The entry contains the identifier of the SAC message and the service provider identifier followed by the currently available service identifiers. The SAC message incoming face indicates that such a face can be used to access a given service provider. The structure of the SAC message is shown in Table 3-12.

Service Availability Changes (SAC)	
Prefix	Prefix indicating that this is a SAC broadcast message
Provider identifier	The unique provider identifier
Message type	SAC message type: add, remove, or bloomfilter
Services	Services as name components for add and remove message types, or a bloom filter containing the services for the bloomfilter message type

Table 3-12 SAC structure

The *Resource Availability Changes (RAC)* Interest is a signaling message communicating that a given service provider is overloaded for a certain time period. The RAC message is composed of four parts. It starts with two-component keywords "bcast" and "rac" indicating a RAC Interest broadcast message. The Interest name is followed by the service provider identifier and the RAC message type. The RAC message can be of two types: busy or available. The busy RAC message indicates that a given service provider is currently not available for some period of time due to the exhaustion of resources. It does not provide information about the available resources. The subsequent service requests should, therefore, not be forwarded towards this service provider. The available RAC message indicates that the service provider is again accessible. Therefore, subsequent service requests may be again forwarded to this service provider. The RAC busy messages are stored in the PIT and will remain there until the RAC timeout has expired, or a RAC available message from a given service provider has arrived. In such a case, the busy RAC entry is removed from the PIT table. Consequently, the service provider is no longer considered busy in terms of resource availability. The structure of the RAC message is shown in Table 3-13.

Resource Availability Changes (RAC)	
Prefix	Prefix indicating that this is a RAC broadcast message
Provider identifier	The unique provider identifier
Timestamp	Timestamp indicating the validity period of the message
Message type	RAC message type: busy or available. Indicates if the service provider is available or busy.

Table 3-13 RAC structure

Service Interest Forwarding

Figure 3-13 illustrates the forwarding procedure of an incoming service Interest request. First, the intermediate nodes check whether there is an entry in the FIB table for the corresponding service. If the FIB entry exists, the service request is regularly forwarded using the FIB. Otherwise, if no FIB entry exists, the SAC and RAC messages stored in the PIT table are used to find the best service provider for a given service. Furthermore, a new entry for the given service is added to the FIB. The FIB entry permits subsequent requests corresponding to a given service to be directly forwarded using the FIB without employing the knowledge gathered from RAC and SAC messages. The FIB entries are also updated upon receiving RAC and SAC messages. Moreover, FIB entries not solicited for an extended period of time are disposed.

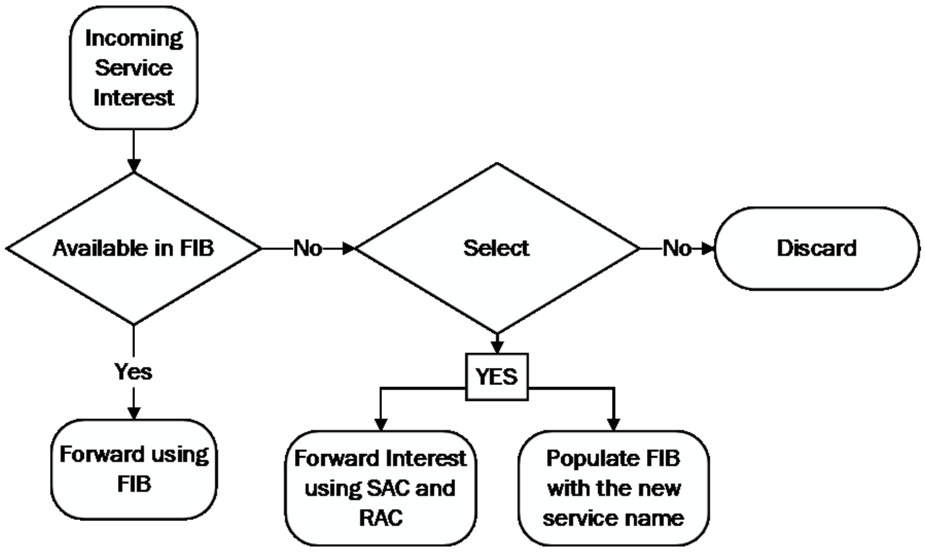

Incoming Service Interest
Available in FIB
No
Select
No
Discard
Yes
YES
Forward using FIB
Forward Interest using SAC and RAC
Populate FIB with the new service name

Figure 3-13: Interest forwarding decision for the event-driven mechanism

3.2.2 Supernodes with DHT Support

This section describes the inter-domain extension of the architecture with DHT query capabilities to better support service discovery. The DHT query capability adds an additional mechanism that enables supernodes to establish inter-network communication. Please notice that in this context, a network consists of a set of interconnected domains. Figure 3-14 displays two disjoint networks, one network is formed with the Domain 1 and Domain 2 that are interconnected, and the second network is formed with the Domain 3. Each network contains a set of domains, and every domain owns a set of nodes and supernodes. The two networks are not interconnected, therefore, they cannot communicate. For example, we can assume that these two networks are geographically located in two different countries and that the two locations are connected only through an IP-based network. Without introducing IP these two networks are not able to communicate together because there is no network supporting NDN between them. The main goal of the DHT support is to enable these disjoint networks to communicate together. Also, a node can perform a lookup in the DHT network for a given service name and get back the IP addresses of disjoint networks where the service is available. It is important to state that we use IP only as an overlay to interconnect the networks; the NDN packets are encapsulated in IP packets to reach the destination NDN network. Currently, there is no global interconnected NDN network [84], consequently, IP is required to interconnect disjoint networks together. Figure 3-14 shows the problem that exists, and why we require IP, and what the role of the supernodes are. Figure 3-14 depicts three domains: Domain 1, Domain 2, and, Domain 3 that are connected together. Between Domain 1 and Domain 2 a native NDN network exists, hence these domains can communicate together over the native NDN network. However, there is no native NDN network between Domain 3 and the two other domains because we do not currently have a global scale native NDN network. The Domain 3 and the other domains can communicate over the global IP network, where NDN runs as an overlay on top of the global IP network [85]. As shown in Figure 3-14, the supernode of Domain 3 establishes an IP connection to the supernodes of Domain 2, for this purpose Domain 3 needs the IP address of a supernode in Domain 2. Once the connection is established, the NDN packets are encapsulated into TCP packets and exchanged between the supernodes. Again, the reason we need IP is because of the lack of a global scale NDN network. If a global network existed, we would not need IP anymore.

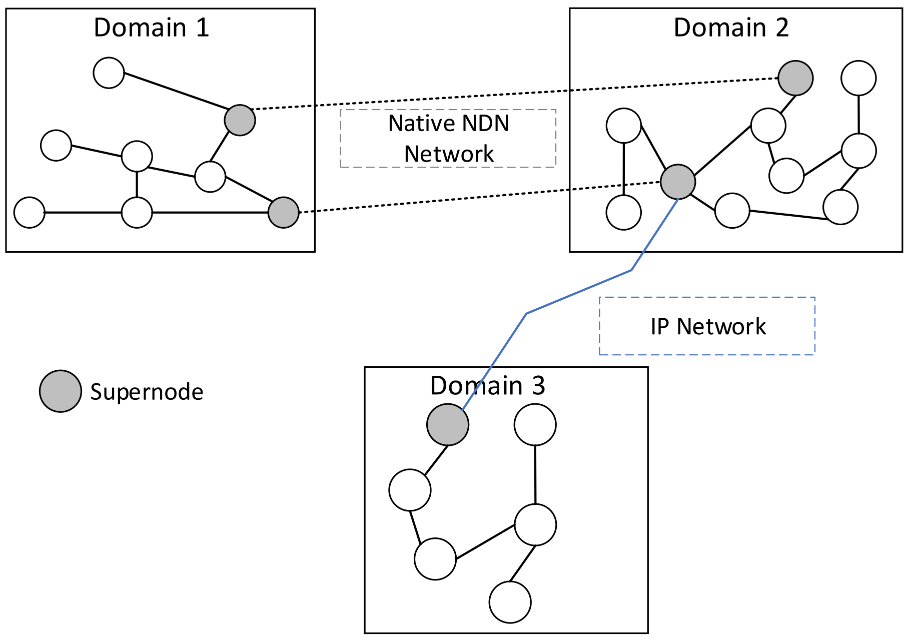


Figure 3-14: Connecting disjoint domains together

NDN enables us to establish an IP-based connection to a distant NDN network by running NDN on top of the traditional IP network [86],[87]. This functionality is natively supported by NDN, which enables the coexistence of different network paradigms and eases the NDN deployment. It is especially important if no direct NDN connection is available between any two disjoint networks. To establish a connection between NDN networks through the traditional IP network, the supernodes need the IP address of one of the supernodes of the distant NDN network. Furthermore, the supernodes should be able to query for a given service and receive back the network addresses where the service is available. This allows an alternative search for request forwarding if there is no available outgoing face corresponding to an incoming service request.

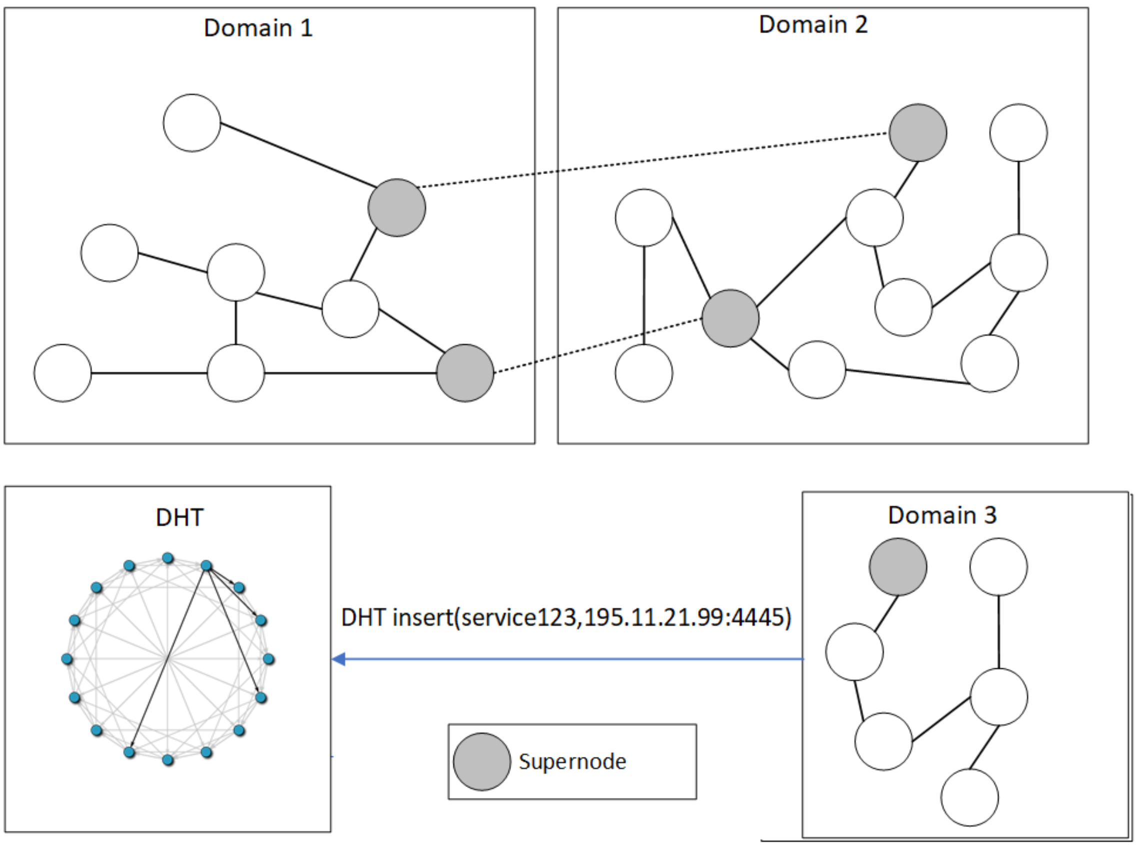


Figure 3-15: Adding a service identifier to the DHT network

Figure 3-15 shows the connection of disjoint domains over the DHT middleboxes. It shows three domains, named Domain 1, 2, and 3. The domains have supernodes (in grey) that interconnect domains together. Domains 1 and 2 are connected via supernodes. However, Domain 3 is disconnected, and, therefore, no connection between Domain 3 and the other two domains exists. In this example, we have two networks, one network is formed with Domain 1 and 2, and the second network is formed by Domain 3. These two networks are not interconnected over NDN, for example, they can be located in two different countries. In this example, the “service123” is requested by a service consumer located in Domain 2. However, this service is only available in Domain 3. With the help of DHT the service request from Domain 2 can be forwarded towards the service provider in Domain 3.

In Figure 3-15, the supernodes of Domain 3 adds the service "service123" to the DHT network, this service is available only on one of the nodes of the Domain 3. The insert operation contains the IP address of the supernodes and a service that is available in that domain. The DHT network takes care of storing a new entry in the DHT network linking the given service to the supernode of the domain. Additionally, DHT primitives allow for a lookup of an entry with a given name and disposing the entry, if it is not needed anymore. The nodes participating in the DHT network do not necessarily have to be members of the NDN network. In other words, theoretically, there is no restriction on who can join the DHT network and store information about domains and the offered services. The solution can be extended with centralized or decentralized solutions for authentication and DHT bootstrap procedures. However, authentication and bootstrap propagation mechanisms are out of the scope of this work. The next paragraph explains how this DHT network is used by Domain 2 to establish a connection to Domain 3 and be able to send service requests for the "service123" towards Domain 3.

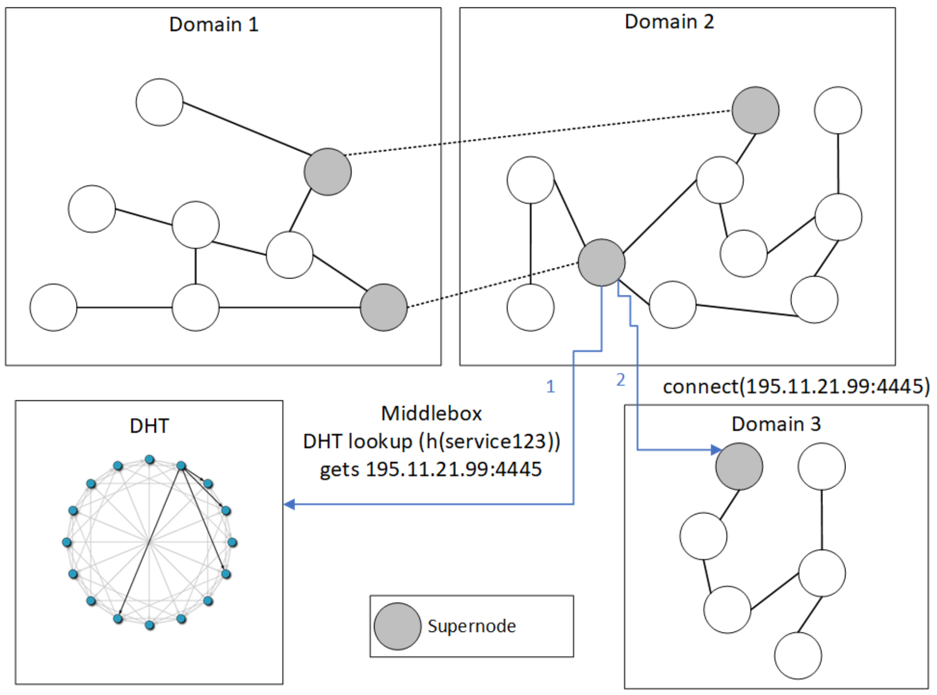


Figure 3-16: DHT request for service forwarding

In Figure 3-16, a supernode of Domain 2 sends a DHT request to get the list of networks, in which the requested service (i.e., service123) is available. Thus, we can connect to the network and send service requests towards it; this is especially interesting if the service is not available in the given network. In the first step (1), the supernode requests the service name (i.e., service123) and receives the list of IP addresses of the supernode of Domain 3 from the DHT network. By using the gathered IP address, supernodes can establish a direct connection to the supernode of Domain 3 using the IP network, and forward Interest requests to the corresponding service provider. A regular forwarding process is used to forward a given service request in the destination domain. The NDN packets are encapsulated in IP packets and sent towards the destination network. Again, we cannot substitute IP with NDN because there is no NDN network between the domains.

From a holistic perspective, DHT allows us to interconnect disjoint domains together. DHT enables us to have access to a more significant number of service replicas to satisfy service requests. Especially, it allows satisfying service requests for which there is no knowledge available in a given network, i.e., no existing FIB table entry.

3.2.3 Performance Evaluation

This section presents the evaluation results of the two introduced mechanisms (i.e., event-driven and provider-driven) compared to the existing consumer-driven solution. We have conducted the evaluations in ndnSIM.

Evaluation Scenario

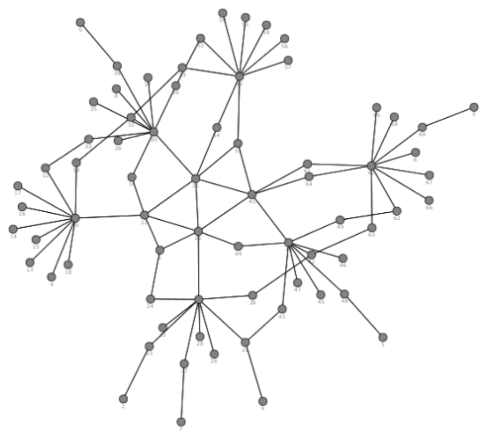


Figure 3-17: Topology composed of 70 nodes

We have provided an evaluation network of 70 nodes. The network topology is illustrated in Figure 3-17. It resembles the Internet topology at a small-scale having high degree connectivity nodes and low degree leaf nodes. We have randomly selected 3 leaf nodes as service consumers and 5 leaf nodes as service providers. Each service consumer sends 100 service requests with a frequency of one request per second. In total, 300 distinct service requests are sent by the service consumers that need to be satisfied by service providers, that process the incoming service requests. The processing time of the service request is uniformly distributed between 1000 and 1500 ms. During the simulation, we evaluate the service processing time, which is understood as the time elapsed between sending the service request and receiving the reply from the provider. The periodic broadcast of the consumer and provider driven approaches is set to 10 s. The busy threshold for the event-driven approach is set to 5 service requests waiting in the processing queue, and a busy message with a validity period of 5 s. These parameter values and this topology represent a realistic setup. We have repeated the execution of this scenario ten times and compared the approaches in terms of processing time of the requests and generated overhead on a statistical basis.

Evaluation Results

We have compared the presented provider-driven and event-driven mechanisms against the existing consumer-driven strategy. Figure 3-18 reveals the mean processing time of the three mechanisms with its respective confidence intervals for the mean value. The bars show the mean processing time for the 300 requests and the horizontal lines mark the confidence intervals for the mean.

The confidence intervals for the consumer and provider driven mechanisms are overlapping (Figure 3-19, horizontal lines), suggesting insignificant differences between the mean processing time of these two strategies. The event-driven approach's confidence intervals are not overlapping with the confidence intervals (horizontal lines) of the two other approaches in Figure 3-19. This suggests a significant difference between the mean processing time of the event-driven approach and the two other approaches. The consumer and provider driven mechanisms have a mean processing time of 1427 ms (±20 ms) and 1417 ms (±19 ms) respectively. The results of these two strategies in terms of mean processing time are close together because both mechanisms request or periodically push service provider resource availability information to the network. The event-driven approach does not use periodic resource availability information propagation, hence its mean processing time for the 300 requests is 2076 ms (± 53 ms). The comparatively high mean processing time of the event-driven mechanism is because of its information propagation approach, which propagates information only in case of service provider overload. The advantage of the event-driven approach is its low overhead (i.e., number of protocol messages), as shown in Figure 3-19.

Figure 3-19 illustrates the NMO (normalized message overhead) of the three aforementioned approaches. We have normalized the overhead (i.e., number of protocol messages) to draw a fair comparison between the mechanisms. The NMO is computed by $NMO = \frac{M}{N}$, where M is the number of message transmissions concerning the overhead and N is the number of unique nodes that have received the message. NMO reflects the overhead cost per node. The respective NMO values for the three strategies are 18, 15, and 8.5. The event-driven approach has the lowest NMO value, because it does not perform periodic broadcasting, but only broadcasts in case of high overload. The consumer-driven and provider-driven mechanisms have similar NMO values. The provider-driven strategy is better in terms of NMO compared to the consumer-driven approach. This is due to the request-response nature of the consumer-driven strategy, which generates more overhead.

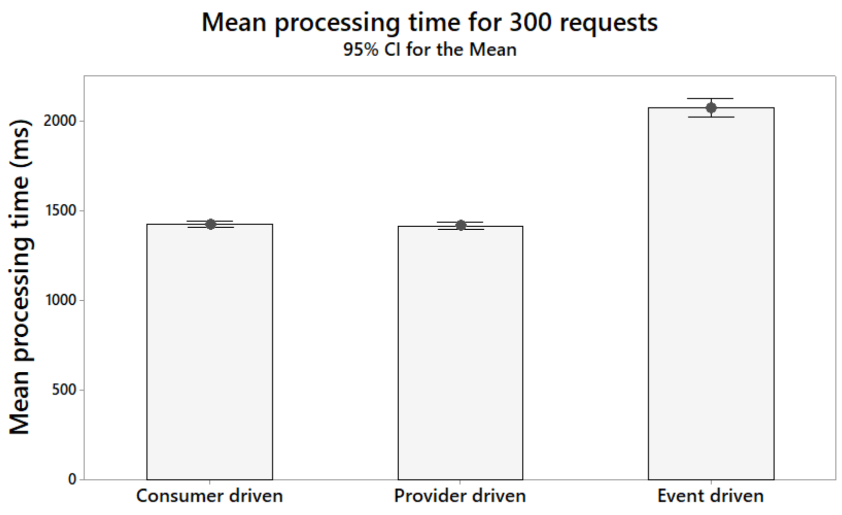


Figure 3-18: Mean processing time for the 300 requests and the respective confidence intervals

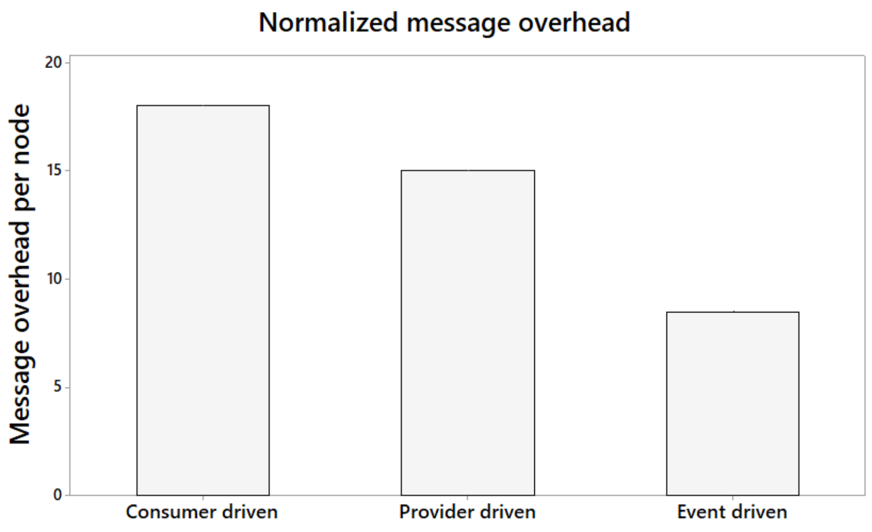


Figure 3-19: Normalized message overhead

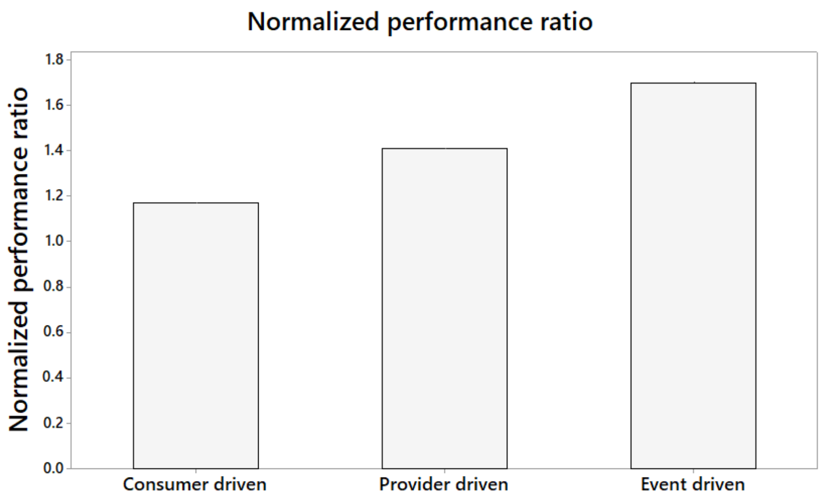


Figure 3-20: Normalized performance ratio

Figure 3-20 shows the normalized performance ratio (NPR), which is an index combining the performance of the three mechanisms in terms of processing time and protocol message overhead. The higher the index value is, the better is the performance achieved by the aforementioned mechanisms. The index is computed as a product of the average processing time and the normalized message overhead divided by the resulting number of satisfied requests. The NPR is computed by the following formula: $NPR = \frac{NSSR}{MPT \cdot NMO} \cdot 100$, where NSSR is the number of satisfied service requests, MPT is the mean processing time, and NMO is the normalized message overhead. We multiply it by 100 to remove decimal points. The NPR values shown in Figure 3-20 are 1.17, 1.41, and 1.7, respectively, for the three mechanisms. The event-driven approaches processing time value is the highest. However, taking into account the overhead in NPR, the event-driven mechanism delivers the best NPR index value.

3.3 NLSR-based Routing Scheme

In this section, we present an intra-domain routing scheme and load balancing mechanism based on NLSR (Named-data Link State Routing) [24] [27]. We implement and integrate an extended version of NLSR named IaDRA-SCN, which is capable of propagating service and resource availability information in the network. IaDRA-SCN relies on a link-state routing protocol suite, this type of routing protocol has high overhead and is less suitable for inter-domain communication. However, it is suitable as an intra-domain communication scheme.

This section is organized in the following manner. The NLSR-based routing scheme is described in Section 3.3.1, and its evaluation is presented in Section 3.3.2. We present IaDRA-SCN in this separate section because it is not a mechanism using NDN primitives but rather a scheme based on a link-state routing protocol suite relying on NDN primitives.

3.3.1 Service Provider Information Propagation with NLSR

This subsection tackles the dissemination of information about services in the network with the help of NLSR. The service providers need to disseminate information about the services provided and the available resources to the network. For this purpose, service providers propagate Service Prefix Information and Resource Availability Information LSAs to the network, LSA dissemination mechanism is described in detail in [33]. The Service Prefix Information LSA is advertised each time a new service (identifier prefix) is added or deleted by a service provider. The Service Prefix Information LSA is used to update the FIB routing tables of the nodes for future forwarding decisions. The Resource Information LSA periodically updates the available resource information (e.g., CPU, RAM) of a given service node. The nodes use the resource availability information for future forwarding decisions. It allows forwarding requests to the provider with the currently most available resources. The design respects the NDN primitives because the information is solicited with Interest messages. We will in the subsequent paragraphs, briefly explain the main components of NLSR that are important for us and how we use it for our purpose.

NLSR [24],[25] is a link-state routing protocol that populates FIB tables of NDN nodes. NLSR uses Interest and Data packets to disseminate routing information. It discovers adjacencies and advertises information about the topology and name prefixes available in the network. For this purpose, NLSR disseminates two Link State Advertisements (LSAs) – Adjacency LSA and Prefix LSA. The Adjacency LSA propagates all active links connecting a node to its neighbors. The Prefix LSA is used to propagate information about existing prefixes registered on each node. We add an additional LSA, which periodically propagates node resource availability information. LSA messages are not propagated by a host-centric push mechanism, but through regular NDN Interest and Data messages. LSAs are gathered by LSA Interest messages, which receive a Data reply containing the LSA content. NLSR uses ChronoSync [33] as a synchronization protocol. ChronoSync uses cryptographic digests to summarize the state of a dataset in a condensed way. The digest is exchanged among network parties to detect differences in the datasets and to disseminate it to the network efficiently. NLSR uses a Hello protocol, which allows detecting a node failure by sending periodic Interest messages to the neighbors. A Data message reply to the Interest message informs us that the neighboring node is alive. As a link-state protocol, NLSR disseminates Link State Advertisements (LSAs) to build a network topology and to distribute name prefix reachability. We extended the LSAs by adding the ability to propagate resource availability periodically. The resource availability dissemination allows efficient load balancing for service requests. This is important because service requests often require substantial processing.

Nodes need knowledge about available services and resources in the network to forward service requests to the service provider with adequate available resources. A node needs information on faces about service reachability and available resources at service providers. Service providers propagate the service and resource information to the network. To enable the propagation of routing information in the network, our design makes use of LSAs. NLSR uses NDN Interest and Data packets to ensure the propagation of routing information. It uses LSAs to propagate information and a Link State Database (LSDB) on every node to store the latest version of LSAs. It uses a specific naming convention for information dissemination among routers, which is of the following structure: /{network}/NLSR/LSA/{site}/{router}/{lsa-type}/{version}. The components of the naming scheme are described in Table 3-14.

Component	**Description**
{network}	the network name to which the router belongs
NLSR	indicates that it is an NLSR request
LSA	indicates that it is an LSA
{site}	the site name to which the router belongs
{router}	the router that originates the LSA
{lsa-type}	the type of the LSA
{version}	version of the LSA incremented each time a new LSA is built

Table 3-14: LSA naming format

The latest version of the LSAs are stored on each router in a database called LSDB. NLSR uses the ChronoSync [33] protocol to synchronize LSDB changes in the network. Information propagation is realized through content synchronization among routers. The ChronoSync keeps LSAs in the LSDB as a name set. To synchronize the latest version of the LSAs, the ChronoSync protocol exchanges a hash as a compact expression of the LSA name set. Using the hash exchange, ChronoSync avoids unnecessary flooding of the network. Instead of requesting LSA names by flooding the network, NLSR first checks if there are new LSAs available by exchanging a hash of the locally available LSA set. If the received hash is different (i.e., the LSAs were updated), NLSR starts requesting new LSAs.

3.3.2 Performance Evaluation

This section presents an evaluation of our extended NLSR scheme. We compared our scheme with existing forwarding strategies in NDN, which are the Best Route, Multicast, and Random strategies [42]. The Best Route strategy forwards incoming requests to the lowest cost provider node by using networking inspired metric values. The Multicast strategy forwards an incoming request to multiple faces, over which the requested Interest is accessible. The Random strategy forwards the request through a randomly chosen face from the set of faces that can lead to the requested service.

Evaluation Scenarios

We have implemented and evaluated our scheme in ndnSIM-NLSR [89],[88]. ndnSIM-NLSR (aka nlsrSIM) is an altered ndnSIM version integrating NLSR for simulation purposes. We have modified ndnSIM-NLSR to provide service request forwarding. We extended the message and forwarding scheme. For service information propagation, we have integrated and extended NLSR with the service information dissemination. The underlying architecture of NDN was left unchanged. We only extend it with additional NDN-native mechanisms to support services. Thus, regular NDN traffic is handled as usual.

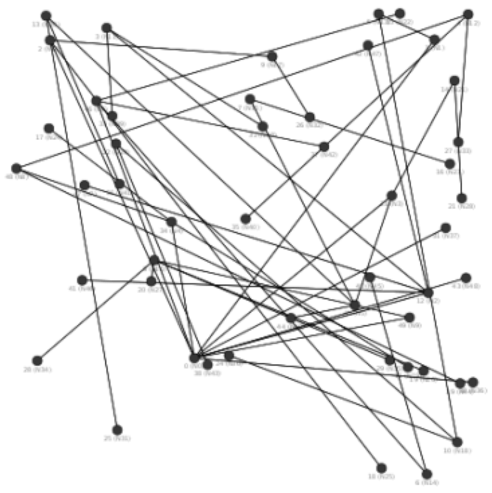


Figure 3-21: Evaluation topology composed of 50 nodes

We evaluate our architecture using an example topology (cf. Figure 3-21) generated by the Internet topology generator BRITE [90], which is supported by the ndnSIM-NLSR framework. We use the flat Albert-Barabasi model [91] to generate a router level topology. The created topology is composed of 50 nodes with a minimum node degree of 1. In BRITE, we use preferential connectivity and incremental growth setting allowing us to produce a topology that highly resembles the real-world Internet. We have randomly designated 4 service provider nodes and 6 service consumer nodes. The LSA refresh time interval is set to 5 seconds. Service consumer nodes send service requests using values drawn from a random exponential function with a mean of 2 seconds. The service provider nodes aim to process incoming service requests sent by the service consumers. The processing time of incoming service requests is set to a uniformly distributed value between 1 and 2 seconds. This implies that the resources on the node are busy during the service processing time. We have designed three scenarios with different number of service requests. In the defined scenario, each service consumer node sends 50, 100, and 150 unique service requests. That sums up to 300, 600, and 900 unique service requests that need to be processed by the service provider in the network. The processing time is defined as the time elapsed at the service consumer between a request origination and the response arrival. The simulation was repeated 10 times for each strategy considered. Finally, we have calculated the mean processing time and the 95% confidence intervals for the mean.

Evaluation Results

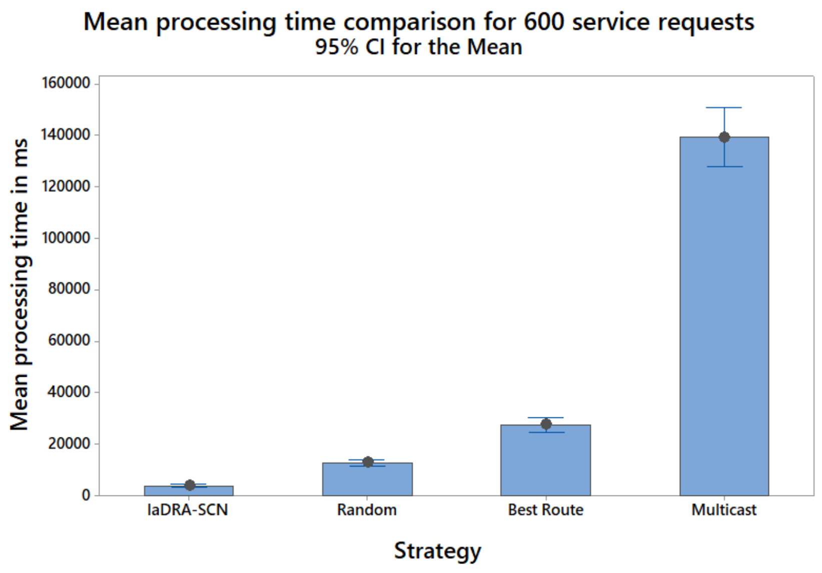


Figure 3-22: Mean processing time for 600 service requests

We compare the processing time of our scheme (i.e., IaDRA-SCN) with the Best Route, Multicast, and Random forwarding strategies existing in ndnSIM-NLSR with default settings. The mean processing times for all strategies considered for the different number of service requests are shown in Figure 3-22, Figure 3-23, and Figure 3-24. We display the results in three distinct figures to ease the readability of the charts. The processing time is defined as the time elapsed at the service consumer between a service request origination and the service result arrival. The circles and bars represent the mean processing times, and the ranges express the confidence intervals for a given strategy. The confidence levels for the strategies considered in the different scenarios do not intersect at all, suggesting significant differences between the mean processing time of the four strategies considered.

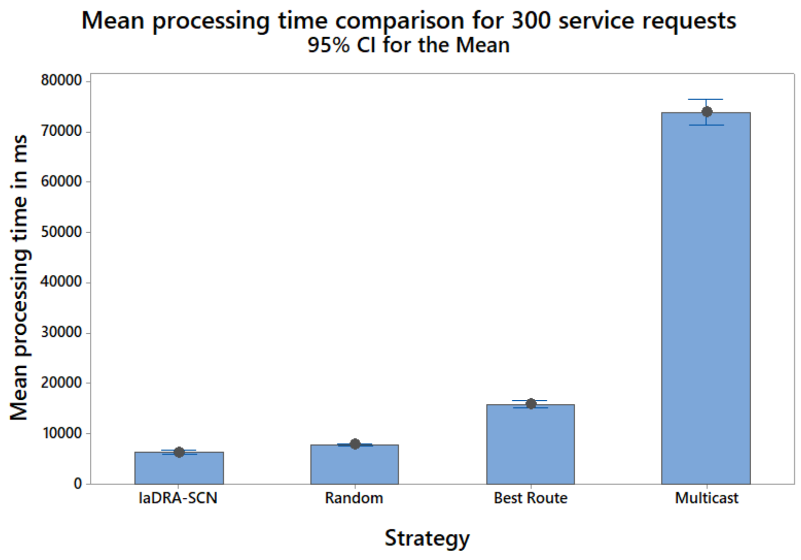


Figure 3-23: Mean processing time for 300 service requests

The Multicast strategy reveals the highest mean processing time in all three scenarios. This is because each service request is forwarded to multiple nodes. Therefore, this strategy quickly saturates computing resources resulting in substantial processing time. The Best Route strategy relies on network-based metric values (e.g., hop count and the number of previously forwarded service requests) to select the outgoing forwarding face. The use of network metrics overloads the best reachable providers, thus increasing their load and effectively slowing down the service. In Figure 3-22, the Best Route strategy provides a mean processing time of 27709 ms (±3011 ms). The Random strategy achieved the best load balancing performance with a mean processing time of 12916 ms (±1089 ms). Our scheme, however, significantly outperforms the three strategies with a mean processing time of 3974 ms (±621 ms).

The evaluation results for the scenarios with 300 and 900 service requests in Figure 3-23 and Figure 3-24 provide the same rank order of the strategies. However, the mean processing times are for all the strategies lower for the scenario with 300 requests and higher for the scenario with 900 requests compared to the scenario with 600 service requests. The mean processing time raises with the number of service requests because the higher the number of requests, the longer the processing queues are on the service providers, and on average service requests need more time to be satisfied. The evaluation shows that our scheme produces better results compared to the existing strategies of ndnSIM. This is due to the integration of a load balancing scheme based on information propagation with the link-state routing protocol.

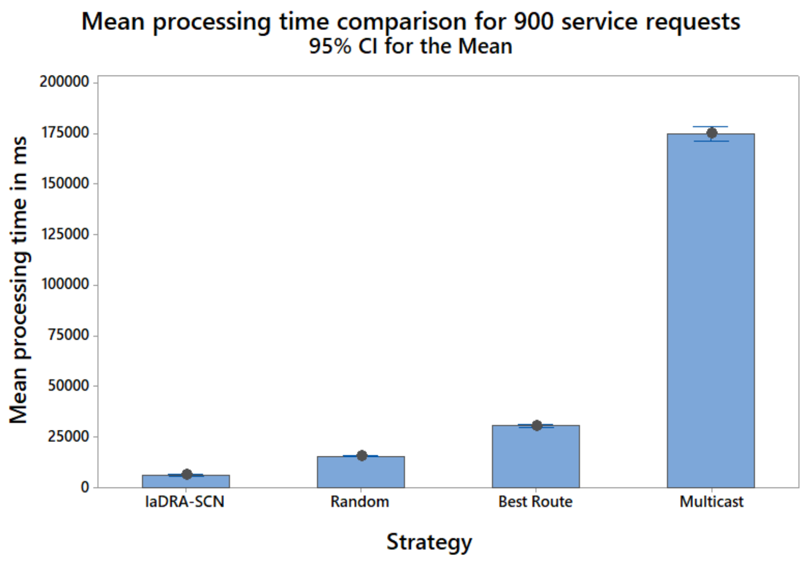


Figure 3-24: Mean processing time for 900 service requests

3.4 Conclusions

To the best of our knowledge, L-SCN was the first SCN architecture combining a two-layer design, supernodes, and Bloom filters to optimize service availability, load balancing, and protocol overhead. We do not require a global coordinator for service requests to reach their respective providers. The nodes share the information on the services provided through Bloom filters, combined with a push and pull mechanism. This allows for a proper balance between protocol overhead and the amount of exchanged information. We have implemented and evaluated the L-SCN architecture in ndnSIM by comparing its performance with the Random, Best Route, and Multicast forwarding strategy available in ndnSIM. The simulation results show that L-SCN outperforms the three forwarding strategies implemented in ndnSIM.

We have presented additional mechanisms for SCN, i.e., event-driven and provider-driven service provider information propagation. The event-driven approach propagates service provider related information to the network only upon an event (i.e., an overloaded service provider). In the provider-driven approach, the service provider is responsible for sending its status information to the network, which results in better mean processing time due to the fine-grained information propagation. We have compared our extended communication approaches against the existing consumer-driven approach of L-SCN. The simulation results show that the approaches offer better results compared to the existing native L-SCN approach in terms of mean processing time and protocol overhead. The provider-driven approach pushes information to the network, hence enabling faster propagation of service provider status information. The event-driven approach significantly lowers the protocol overhead and offers a better trade-off ratio between overhead and mean processing time. Furthermore, we have enhanced the forwarding scheme with DHT-based collection of alternative forwarding information. Our NLSR-based SCN routing scheme uses a link-state routing protocol and integrates short and long type input parameter support. Our scheme disseminates information about link connectivity, available services, and resources to the entire network. For the propagation of network-related information, it uses a synchronization mechanism instead of flooding. The disseminated information allows nodes to maintain a global view of the network. Consequently, it enables us to forward incoming service requests to the service provider with the most available resources. Our design does not push content to the network, hence, it respects the NDN primitives, which aim to solicit content with Interests. We have implemented a prototype of our NLSR-based scheme and compared it against existing forwarding strategies available in the ndnSIM-NLSR. The evaluation results reveal that our scheme outperforms existing strategies in terms of processing time.

Chapter 4 Session Support for SCN

An important component of services are sessions, which allow communicating parties to establish a semi-permanent message exchange. This chapter provides the first, to the best of our knowledge, session support for SCN published in [28]. Sessions are important because they allow communicating parties to create a context and perform a series of operations in the established context. For example, consider a cloud computing application, where the service provider should instantiate a virtual machine before it can process the incoming request. Without session support, requests requiring the same virtual machine might reach different service providers that should all instantiate the required virtual machine. Since instantiating a virtual machine is a time-consuming operation, this behavior is not desired. By using sessions, the requests requiring a single virtual machine will be routed to the same service provider. The selected service provider instantiates the virtual machine, when the first session request arrives. Then, further requests concerning the created session may use the same instance of the virtual machine. Another example of application type that benefits from sessions are security-related applications, e.g., encryption/decryption services. To process incoming data, these services require a key exchange, i.e., to create a context, which allows for efficient and secure processing. Sessions are beneficial for processing of continuous service requests requiring an execution context.

Though, the initial session support design does not provide multipath capability. Therefore, we extend it with multipath support, which enhances the scheme with fault-tolerance and improves the distribution of service requests, especially central in case of traffic overload. Section 4.2 presents multipath session management using three different mechanisms. The first mechanism uses Bloom filters to propagate service session identifiers. The second mechanism propagates service provider identifiers to the network. The third mechanism uses piggybacking for propagating service provider identifiers, which significantly reduces the traffic overhead.

This chapter is organized in the following manner. The session support design is presented in Section 4.1, and our three multipath session support mechanisms are presented in Section 4.2. Finally, we conclude this chapter in Section 4.3.

4.1 SCN Session Support Design

In our mechanism for SCN session support, a service requester can establish a session with a service provider and use the created session during a time period. For our purposes, we modified the NDN framework to support services and sessions. The existing NDN implementation remains unchanged but gets extended with service and session support. The following subsections will describe in detail our session support concept and its evaluation.

4.1.1 Introduction

The basic idea of a session is that two parties share a unique session identifier, which allows both sides to send and identify requests for a given session. In our design, both the service consumer and the service provider generate a unique identifier. The identifiers are then concatenated to provide a unique session identifier, which is used by intermediate routers to forward the Interest and Data messages for a given session. The session route, as well as the pending session requests, are stored in the traditional FIB and PIT tables. FIB is an NDN table, which saves forwarding faces for different prefix names, while PIT keeps track of pending Interest requests (cf. Section 2.1). Please note that session security aspects are out of the scope of this work. However, the presented work can be extended to integrate session security support. Furthermore, our session support mechanism is not designed with mobility support in mind. Hence it is not intended to be used in mobility scenarios. For connection-oriented communication in mobility scenarios, there are approaches such as [92]-[93] aimed for mobility support.

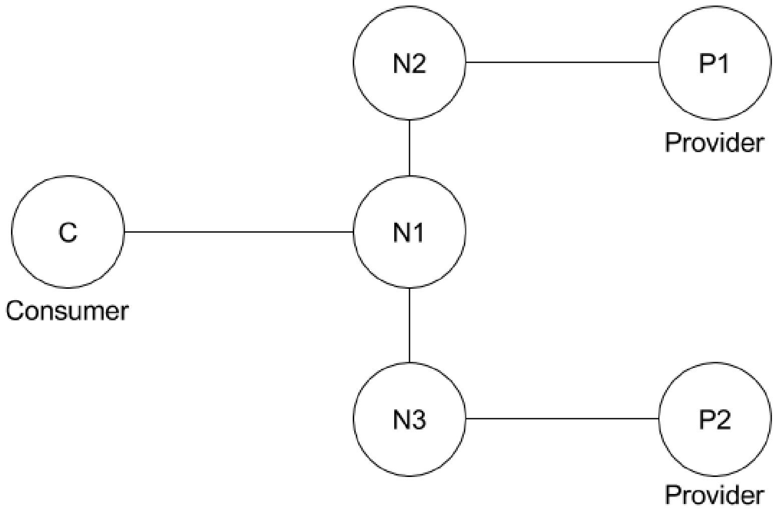


Figure 4-1: Topology with a service consumer, a service provider, and three intermediate nodes

Figure 4-1 illustrates a network topology with a service consumer C, two service providers P1 and P2, and three intermediate nodes n1, n2, and n3. To handle service requests, we use the traditional NDN content naming and forwarding scheme. Consider a request for service getWeather. The service name always starts with a prefix service that indicates a request for a service, allowing nodes to distinguish between content and service requests and preventing the service provider from additional matching operations. Incoming service requests can be sorted and sent faster to the underlying service for processing. The service name follows the prefix. In this example, the request asks for service/getWeather, where '/' is a separator.

Our aim is to concentrate on session support. Therefore, we do not consider other architectural aspects related to services. In our example, topology (cf. Figure 4-1), consumer C establishing a service session has to create and send a specific Interest message towards a service provider. The Interest message sent is provided to an arbitrary face leading towards the service name using the underlying forwarding strategy of NDN. The Interest message is then traditionally forwarded as NDN traffic by the intermediate routers to the provider, which in turn replies to the Interest with a Data message. This two-way handshake allows exchanging unique identifiers among the provider and consumer. The created session is stored by the intermediate nodes for future forwarding decisions. The session information is saved in existing NDN data structures. The session can be deleted by a timeout or a specific delete request. It is important to note that we only extend NDN by introducing a two-way handshake mechanism to create a session without modifying the underlying architecture of NDN. In the following, we first extensively elaborate on session establishment. Second, we show how to use the established session, and finally, we illustrate session termination.

4.1.2 Session Establishment

First, a service consumer establishes a session. The session creation between two communicating entities is initiated through a Session Start Interest (SSI) request sent on the client-side. SSI is a message sent by the session requester to initiate a session. The SSI carries a name, which follows a specific naming convention. The name is divided into three parts. The first part contains two elements: the prefix service indicating a service request followed by the service name.

As an example, /service/getWeather/ could be used, where getWeather is a specific service name. The second part provides the session establishment keyword session/request. Effectively, /service/getWeather/session/request is generated by concatenating the two parts. The third remaining part carries the unique session identifier (generated on the consumer side) that is attached to the end of the request. The complete name containing the three elements could look like /service/getWeather/session/request/kdi32jd329j92rgq, where kdi32jd329j92rgq is a unique session identifier.

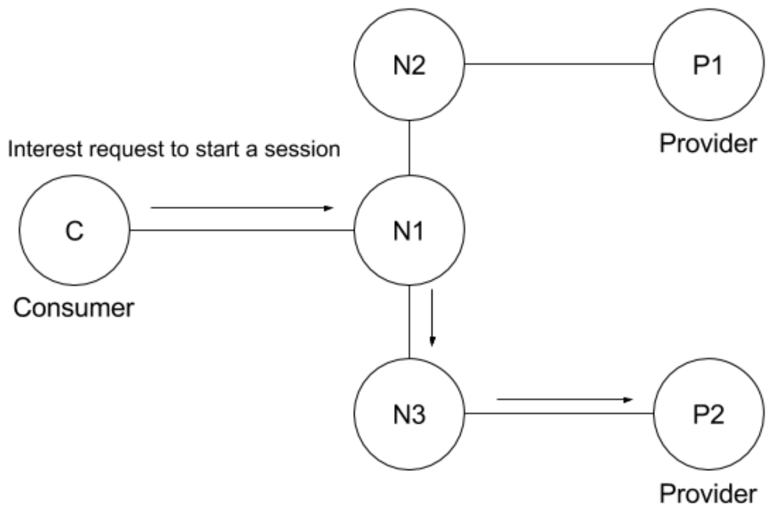


Figure 4-2: Initiation of a new service session by consumer C

Such a request initiates a two-way handshake establishing a new session. SCN forwards the SSI message as a traditional Interest message, as illustrated in Figure 4-2. Service consumer C requests the establishment of a session with any provider handling the required service (e.g., getWeather). First, consumer C sends an SSI message through an appropriate face leading to the corresponding service providers. The decision on the face selection depends on the underlying forwarding strategy of NDN. Second, intermediate node N1 forwards the request to node N2 or N3, depending again on the NDN forwarding strategy. In our example, the requested service is provided by service providers P1 and P2. Assuming that the underlying forwarding strategy sends the request along the path C – N1 – N3 – P2, service provider P2 will handle the session request. Again, the path selected by the SSI message is determined by the underlying NDN forwarding strategy.

Pending Interest Table	
Node	***Name***
N3	service/getWeather/session/request/kdi32jd329j92rgq

Table 4-1: PIT table of provider node P2

A session identifier is a concatenation of two unique identifiers generated by both the consumer and the provider. A provider, who receives an SSI message from a consumer with the consumer's unique session identifier, has in turn to generate its own provider unique session identifier. The provider's identifier is sent backward to the consumer. To accomplish this goal, the provider replies to the SSI message with a Service Start Data (SSD) message that contains the provider's unique identifier. The SSD message is forwarded as a usual NDN Data message through intermediate nodes N3 and N1 to reach consumer C. Table 4-1 shows the FIB and PIT tables of the provider node, after P2 received the SSI message. The PIT contains one entry created when the SSI message arrives. Like traditional NDN traffic, this PIT entry will be removed upon a data response. In our example, the SSD message corresponding to this PIT entry has already been sent.

Forwarding Information Base	
Node	***Name***
P2	service/getWeather
P2	service/getWeather/session/kdi32jd329j92rgqlkoq8i47ehobajua

Table 4-2: FIB table of provider node P2

The FIB table (cf. Table 4-2) of service provider P2 displays two entries pointing towards P2 itself. The first entry indicates that service requests towards getWeather should be handled locally and do not need to be forwarded any further. The second entry holds the session name, which is created upon the SSD reply. The session name contains the service name and a concatenation of the unique identifiers produced by the consumer and the provider. It indicates that provider P2 should locally handle messages on a per session basis.

Forwarding Information Base	
Node	***Name***
N2, N3	service/getWeather
N3	service/getWeather/session/kdi32jd329j92rgqlkoq8i47ehobajua

Table 4-3: FIB table of node N1

Pending Interest Table	
Node	***Name***
C	service/getWeather/session/request/kdi32jd329j92rgq

Table 4-4: PIT table of node N1

The SSD message is forwarded through intermediate nodes on the route back towards the consumer. The standard routing mechanism of SCN is left unchanged; we take advantage of the traditional routing mechanism. Therefore, both ordinary content and services can be routed as usual. Forwarding of SSD messages in the backward direction is based upon the PIT entries on the corresponding routers. In our example, SSD is forwarded by intermediate nodes N1 and N2. Upon every forward, we create a new FIB entry on a given node to handle future forwarding decisions. The name of the newly created entry is composed of the service request name and concatenated unique session identifiers (from both consumer and provider). Table 4-3 illustrates the FIB entry of the intermediate node N1 when the SSD message arrived. The

PIT entry (cf. Table 4-4) reflects pending SSI requests. The pending entry will be removed once a corresponding SSD message goes through. We illustrate a newly created entry in the FIB table (cf. Table 4-3); it has the same name as the FIB entry of P2 and specifies the route for incoming interests in the newly created session.

The FIB entry is created when the SSD message as a reply for a corresponding SSI message arrives. The name is composed of the Interest Name, which contains information about the service name and the consumer session identifier. To construct the full entry name, the provider session identifier taken from the provider's SSD reply message is concatenated with the consumer's session identifier. This FIB entry sets the face, through which future requests belonging to a given session should be forwarded such that they reach the correct provider for the given session. In this example, N1 forwards to N3, which in turn forwards to P2.

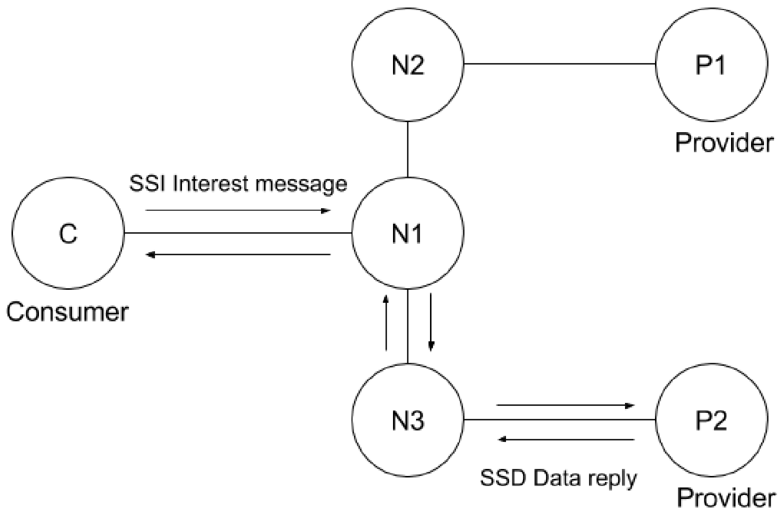


Figure 4-3: Complete process of session establishment

The complete process of the session creation between service consumer C and service provider P2 is illustrated in Figure 4-3. To conclude, in a session initialization, a service consumer sends an SSI message, which specifies the name of the service requested in the name field, session/request indicating the request for the session start, and the consumer's generated session identifier. The forwarding routers usually forward the request to a service provider and populate their PIT table. Upon receiving the SSI message, the service provider generates another (provider's) session identifier and replies to the SSI with an SSD message containing the provider session identifier. Intermediate forwarding nodes save the session name in their FIB tables for future forwarding decisions. Upon receiving the SSD message, the service requester can start sending service requests using the newly created session name. In the following, we describe the use and termination of an established session.

4.1.3 Session Use

To use the newly created session, the consumer sends its Interest request with the following naming convention, i.e., service/[service-identifier]/session/[session-identifier]. It contains two keywords: service and session, i.e., the service name, and the session identifier. This allows the requester to send Interests in the session, which are appropriately forwarded by the intermediate routers based on the standard forwarding and naming system of NDN, i.e., the use of FIB entries.

4.1.4 Session Termination

To terminate a session, the consumer sends an Interest for the given session equipped with the keyword "terminate". When the provider receives such an Interest, it replies with the confirmation of the session end. When the intermediate routers receive the provider's confirmation, they delete the corresponding session FIB entry preventing from future use of a given session.

4.1.5 Single Path Session Support

So far, session support for SCN was presented. Sessions allow for a dynamic establishment of paths between two communicating entities, i.e., the service provider and consumer. This path is used to exchange service requests in an arbitrary session. This means the session support mechanism works only over a single path that was used during the session establishment phase, i.e., it does not provide multipath session support. Section 4.2 introduces mechanisms that enhance our service session support with multipath capabilities, i.e., enabling alternative paths for an established session.

4.1.6 Evaluation of the Session Support Mechanism

This section presents the evaluation of our service session support mechanism. We compared our service session support mechanism against the three existing strategies in NDN, namely, the Random, Multicast, and Best Route strategies. The choice of these methods for comparison was motivated by the fact that there is no session support available in NDN. The Random strategy forwards the incoming request to a randomly selected face, which leads to the required service. The Multicast strategy, as its name suggests, broadcasts the request to all faces leading to the service required. The Best Route strategy uses network metrics to classify faces and forwards an incoming request to the face with the lowest cost.

Furthermore, we assess the impact of creating and managing sessions of our design.

Evaluation Scenario

The implementation and evaluation of our session support have been performed in the ndnSIM simulator. Our implementation does not alter the NDN primitives but extends it by integrating our session concept in the existing forwarding scheme and data structures. This is important, because it allows traditional NDN traffic to be handled as usual.

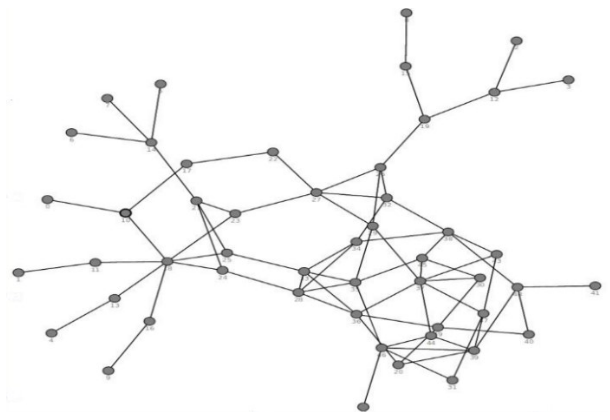


Figure 4-4: Our evaluation topology composed of 50 nodes

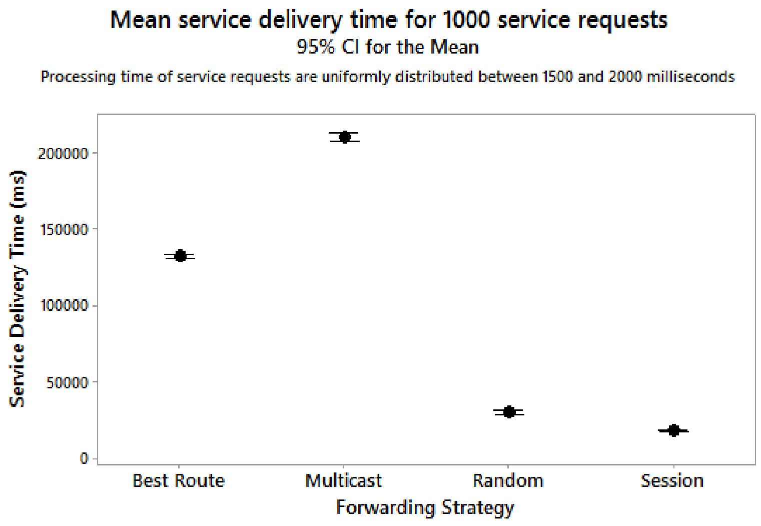
We have evaluated our implementation in a testing scenario composed of 50 nodes (cf. Figure 4-4). We chose a graph that resembles, on a small-scale, the topology of the Internet. In our evaluation, we focus on the evaluation of the service delivery time with and without the integrated session support. The service delivery time is defined as the elapsed time between the request sent and the response received by the consumer. Our evaluation topology contains 4 provider and 8 consumer nodes, which are randomly selected leaf nodes in the network graph. The service consumers send service requests approximatively every second, by using a random variable of exponential distribution with the mean value equal to 1 second. The service provider nodes process each incoming service request with a processing time equal to a uniformly distributed random variable. We have performed two evaluations, in which processing time varies between 1500 ms and 2000 ms, and between 2500 ms and 3000 ms. All provider nodes run three different services that can be requested by service consumers.

We have executed this scenario comparing our session support mechanism against the three existing forwarding strategies of NDN (i.e., Best Route, Multicast, Random). When the session mechanism is used, a node establishes a session, uses the session twice sending a service request, and closes the session.

For each strategy, every service consumer sends 1000 service requests towards service providers. The service name of the Interest request is randomly selected from one of the three services that are installed on the providers. We ran two different simulations by varying the service request processing time. Each simulation was repeated 10 times, and finally, we used the mean service delivery time and confidence intervals for the mean to compare the strategies considered.

Evaluation Results

The charts in Figure 4-5 show the mean service delivery time for the different strategies with different request processing time distributions. Each chart in Figure 4-5 shows the mean service delivery time (circle) with their respective 95% confidence intervals (horizontal lines) for the average of the 1000 service requests for the given request processing time distribution. In both charts, the confidence intervals for the four strategies do not overlap; this suggests a significant difference between them in terms of mean service delivery time for 1000 requests. Moreover, the mean service delivery time for our session support is significantly lower than that of the existing strategies in NDN.



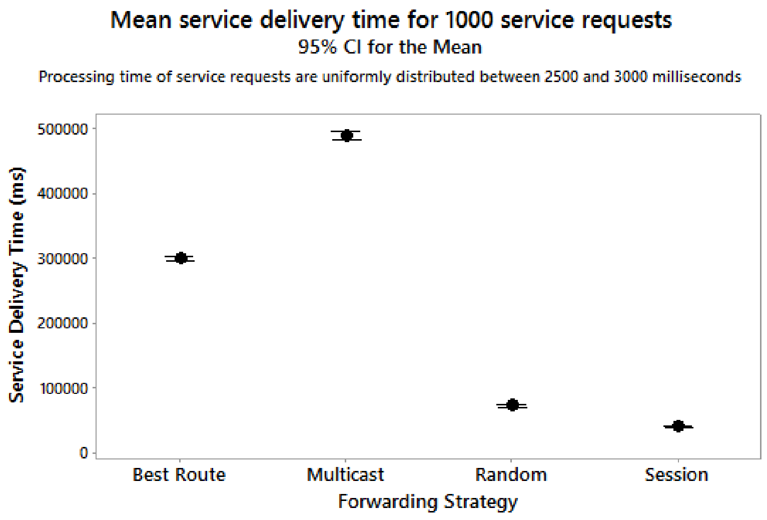


Figure 4-5: Mean service delivery time and 95% confidence intervals for the mean concerning 1000 requests.

The very high mean value for the Multicast strategy is due to missing load balancing for service requests. The Multicast strategy forwards all requests to multiple providers, and consequently, its requests are handled multiple times in the network. The relatively high service delivery time of the Best Route strategy is due to inefficient load balancing among processing nodes. Out of the three strategies available in NDN, the Random strategy was the most efficient. The Random approach performance is an effect of forwarding requests randomly to different service providers. The charts show that a change in the processing time distribution does not affect the performance rank of the strategies considered. Therefore, our evaluation shows that our session support mechanism outperforms existing strategies in ndnSIM.

Impact of Creating and Managing Sessions

Introducing sessions into the architecture comes with some additional overhead. The two central overheads theoretically analysed are the storage of the session information and the traffic required to control the session. The information required to handle sessions on the node grows linearly with the number of sessions terminated at that location. Concerning the additional message overhead required to manage sessions, our mechanism uses a two-way handshake to create a session and an additional message to terminate a session. This means that three overhead messages per session need to be used. The total number of overhead messages in the network, therefore, grows linearly as $3 \times s$, where s is the number of sessions established in the system. It is worth noting that the total relative message overhead depends on how the established sessions are used.

Figure 4-6 illustrates the relation between the number of service requests per session and the total network overhead for 1000 service requests that need to be satisfied. For example, if a session is established to send one service request, 5000 messages (i.e., SSI, SSD, Interest, Data, and Termination) in total instead of 2000 messages (i.e., Interest and Data), which implies relative overhead of 150%. However, if the number of service requests handled by a session is 100, then we only suffer from overhead of 1.5%.

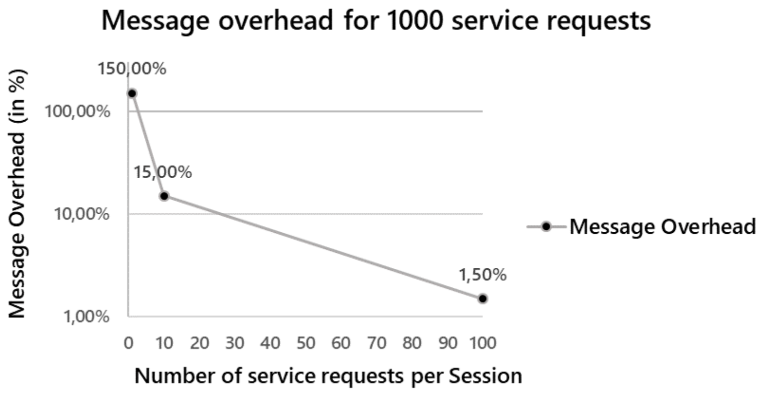


Figure 4-6 Overall message overhead for 1000 service requests, compared with different number of service requests per session

4.2 Multipath Session Support

In this section, we enhance the standard session support presented in the previous section with multipath (and fault-tolerance) capabilities. To ease the understanding, we refer to the SCN session mechanism from the previous section (cf. Section 4.1) as the single path session support, while our enhancements presented in this section (i.e., Section 4.2) are referred to as the multipath session support.

The support for sessions over a single path means that the session has been established over the path selected by the underlying NDN forwarding scheme. Then, only the nodes, over which the session has been established, possess the information on the presence of a given session. Therefore, the session is recognized over the initial path (single path), and hence, only this single path may forward session requests towards a corresponding service provider.

On the other hand, multipath session support indicates that the information about a session is propagated in the network. Therefore not only the intermediate forwarding nodes, over which the session has been initially established, share the knowledge about the session but also other nodes in the network recognize this session as well. This enables requests corresponding to a given session to reach the service provider by using multiple alternative paths. Moreover, multipath routing brings a set of benefits such as fault-tolerance, network resource utilization, security, and privacy, as all requests do not have to be forwarded over the single path.

To summarize, single path session support means that only the initial path may be used to reach the service provider of a given session, while multipath support means that multiple alternative paths can be explored to reach the service provider of a given session. In the next subsections, we explain in detail the problems of the single path information propagation and propose solutions to enable multipath session support.

4.2.1 Introduction

In the previous section (cf. Section 4.1), we presented and evaluated our session support for SCN. Our session mechanism allows a service consumer to establish a session with a service provider through a two-way handshake.

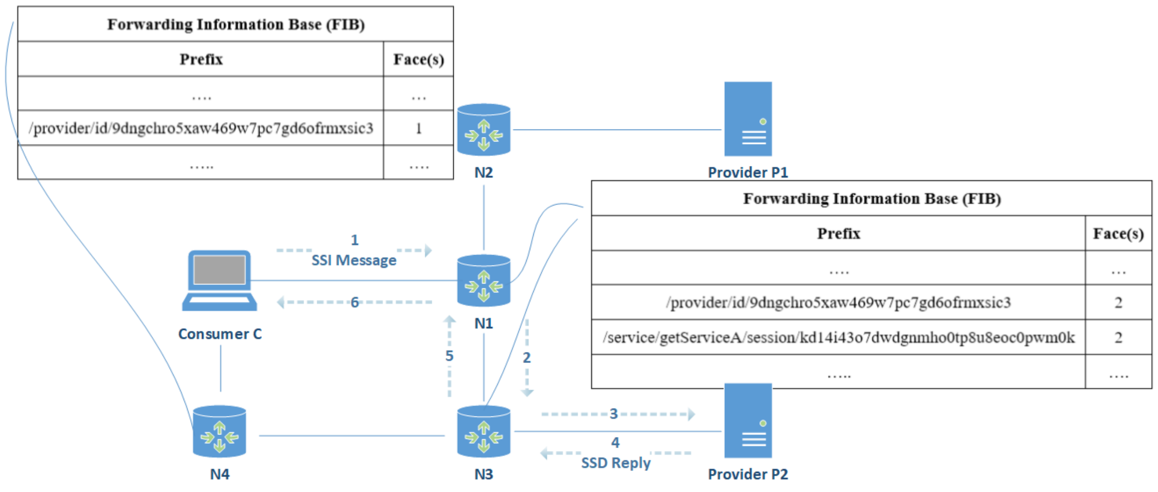
A session is unfortunately only recognized at intermediate routers along a single path, through which the session has been established. Alternative paths from the service provider to the service consumer are not explored, meaning that the session breaks down when link failures or overloads occur on the single path. It is worth noting that alternative paths from the service provider to the service consumer offer multipath capabilities for better load balancing and recovery from link and node failures. Therefore this work aims to integrate multipath routing into the SCN session support.

In Section 4.2.2, we present three designed mechanisms that enable multipath session support for SCN. The first mechanism consists of Bloom filter-based propagation of session identifiers, the second proposal is based upon the dissemination of service provider identifiers, and the third approach uses piggybacking to exchange service provider identifiers. Each mechanism has its advantages and disadvantages, which are elaborated in detail in the remaining part of this section. Before starting the description of the mechanisms, the next section explains the essential components and introduces an example of the designed session support that demonstrates the problem and importance of multipath session support.

Problematic of Single Path Session Support

Sessions allow two communicating parties to establish a semi-permanent communication channel for message exchange. A session is identified by a session identifier, which in our design emerges as a concatenation of two random values generated by the service consumer and the service provider (cf. Section 4.1), and offers three connection phases, i.e., session establishment, session use, and session termination. This section briefly elaborates on the session support mechanism providing an example that illustrates the problems of single path message interchanges. Furthermore, it describes important aspects of multipath session management.

Figure 4-7 illustrates an example topology composed of two service providers P1 and P2, one service consumer C, and three intermediate routers N1, N2, and N3. Figure 4-7 depicts the two-way handshake realized through the exchange of SSI (i.e., 1-3) and SSD (i.e., 4-6) messages. During the forwarding phase of the SSD reply, intermediate nodes add a FIB entry (i.e., in the FIB table), which contains the session identifier. Consumer C can start sending session requests upon receiving the SSD reply from provider P2. A consumer can start using a session when the session is established, meaning that the service consumer successfully received the SSD reply to its initial SSI message. To use a session, the consumer creates Interest requests that use the same naming convention as the FIB entries. The intermediate nodes can properly forward this name prefix, while the two-way handshake appropriately configured the FIB entries among the forwarding routers for a given session, as already explained in Section 4.1. However, the forwarding table entries for a given session are only known by the intermediate nodes, through which the session has been established; other nodes do not possess any information about the establishment of such a session. Hence, only the nodes, over which the session has been established, are able to forward the session requests to the corresponding service provider. Please notice that the migration of a given session to another node is not possible, because the remaining nodes in the network do not store any information about the established sessions.



Forwarding Information Base (FIB)	
Prefix	**Face(s)**
....	...
/provider/id/9dngchro5xaw469w7pc7gd6ofrmxsic3	1
.....	....

Forwarding Information Base (FIB)	
Prefix	**Face(s)**
....	...
/provider/id/9dngchro5xaw469w7pc7gd6ofrmxsic3	2
/service/getServiceA/session/kd14i43o7dwdgnmho0tp8u8eoc0pwm0k	2
.....	....

Figure 4-7: Session Establishment

4.2.2 Multipath Session Support Strategies

Introduction

The previous section described why session support is path related and the importance of multipath session support. This section, in turn, presents three multipath approaches for SCN session management; the approaches are listed below. The central goal of the three strategies is to provide alternative paths in sessions, i.e., multipath support for established sessions.

1. The first mechanism is based upon the propagation of session identifiers using Bloom filters.
2. The second design is based on the propagation of service provider identifiers.
3. The third design uses piggybacking for the dissemination of service provider identifiers.

Session Information Propagation using Bloom Filters

The first multipath session management uses Bloom filters to propagate a set of session identifiers in the network (i.e., the element set consists of session identifiers). Bloom filters disseminate session identifiers currently maintained in the network. The propagation of Bloom filters is realized through the broadcast of regular NDN Interests (i.e., Broadcast Interests) having a name, in which the prefix "/broadcast/sessionids/" is followed by a randomly generated identifier. In our example illustrated in Figure 4-8, P2 can use the following naming structure: "/broadcast/sessionids/ dsa2dj7w1oude7au", where "dsa2dj7w1oude7au" is P2's unique randomly generated identifier. The first two name components indicate a broadcast Interest containing provider's session identifiers. The third Interest name component ensures that Broadcast Interests from distinct providers cannot be discarded by NDN as redundant forwarding due to unique name components (i.e., different random identifiers of different providers). Furthermore, loop-free Interest propagation is guaranteed by the Interest's Nonce field [94].

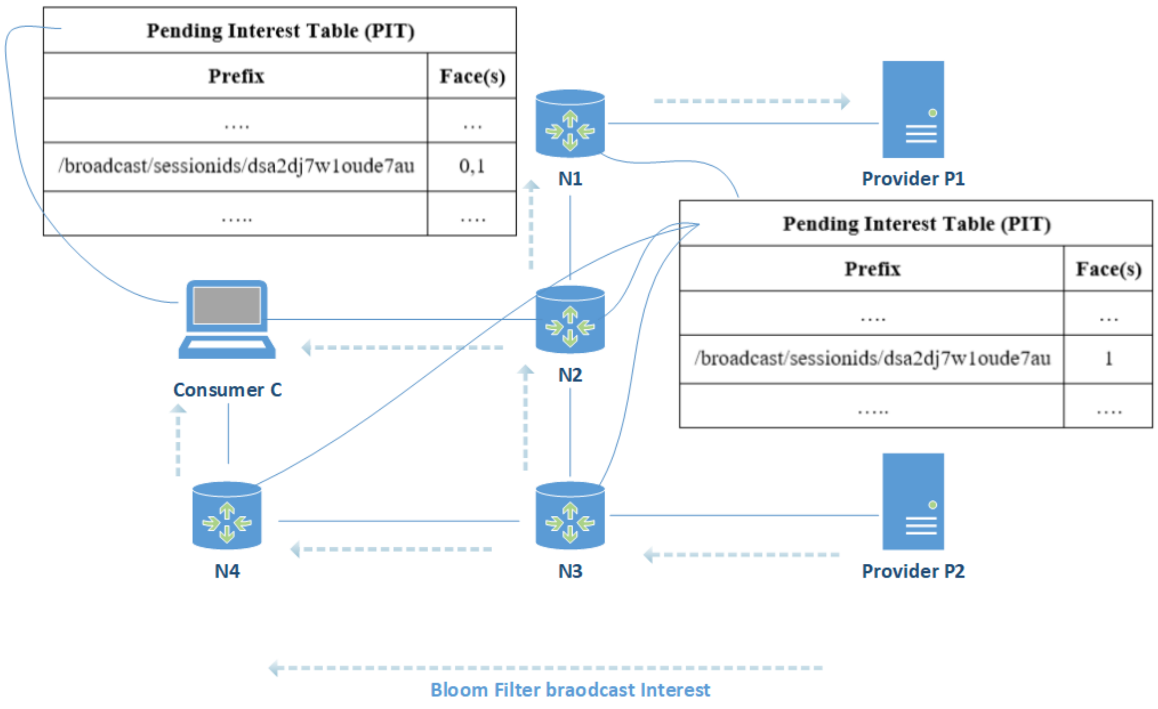
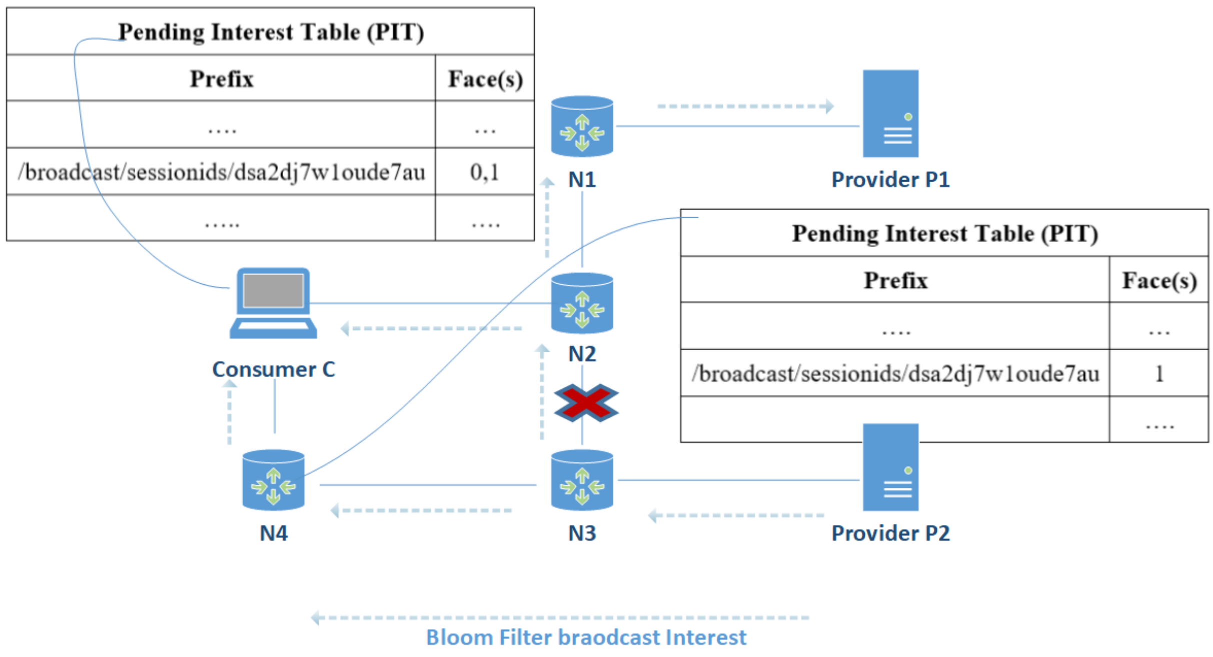

Pending Interest Table (PIT)
Prefix
Face(s)
....
...
/broadcast/sessionids/dsa2dj7w1oude7au
0,1
.....
....
N1
Provider P1
Pending Interest Table (PIT)
Prefix
Face(s)
....
...
/broadcast/sessionids/dsa2dj7w1oude7au
1
.....
....
Consumer C
N2
N4
N3
Provider P2
Bloom Filter braodcast Interest

Figure 4-8: Bloom filter broadcast

When a node receives Broadcast Interests, it forwards them over all faces except the Interest's face of arrival (i.e., incoming face). Similar to regular Interest messages, Broadcast Interest messages are stored in PIT tables. As regular PIT entries, they are removed from the PIT table when their timeout expires. When a node receives two distinct Broadcast Interests originated from different service providers, it adds two separate entries in its PIT table, i.e., one for each distinct Interest. Figure 4-8 shows an example of the propagation of session identifiers using Bloom filters. In this example, provider P2 propagates a Bloom filter containing session identifiers. The identifiers are propagated in the network (dashed arrows) because routers forward Interests containing Bloom filters further also adding entries among PIT tables.

Figure 4-8 shows two PIT tables belonging to consumer C and intermediate node N2. Both tables contain the broadcast entry added upon receiving an Interest request. The PIT table of consumer C has two entries in its face column, because it received Interests from two different faces. In our example, consumer C can reach provider P2 through N2 and N4, which can enhance session management with multipath. In Figure 4-9 consumer C and provider P2 have established a session through nodes N2 and N3. All sessions are then advertised by provider P2, which periodically broadcasts its session identifiers (cf. Figure 4-8). In the case of the link failure between N2 and N3, consumer C may use the alternative path via N4. When a predefined number of session Interest retransmissions fails along a given path, consumer C and the intermediate router N4 find an alternative path by checking the session identifier in the received Bloom filters stored in PIT tables. Intermediate router N3 knows about the session, because the session was set up over N3.



Pending Interest Table (PIT)	
Prefix	Face(s)
….	…
/broadcast/sessionids/dsa2dj7w1oude7au	0,1
…..	….

Pending Interest Table (PIT)	
Prefix	Face(s)
….	…
/broadcast/sessionids/dsa2dj7w1oude7au	1
	….

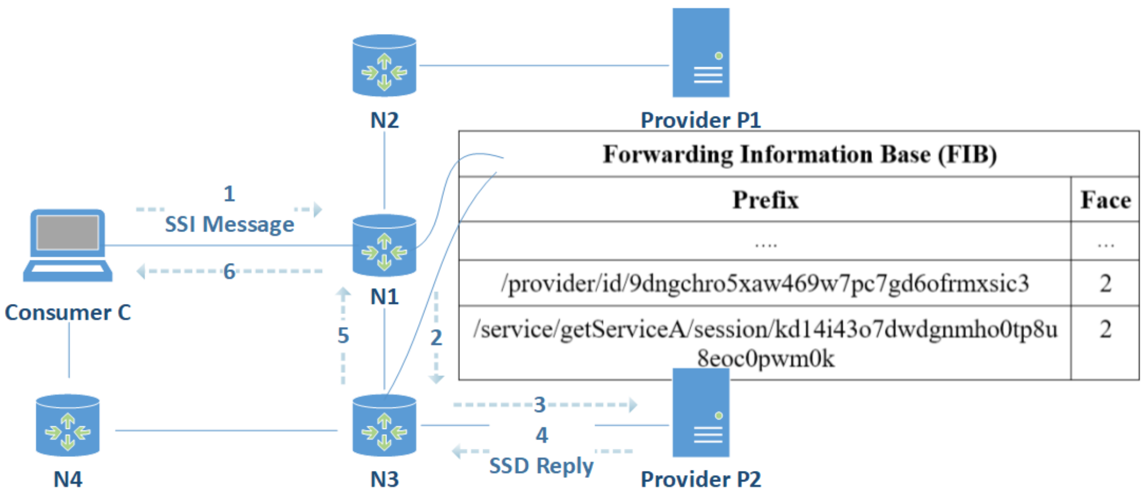
Figure 4-9: Link failure between N2 and N3

Bloom filters do not have false-negative errors. However, false-positive errors may occur. In the case of false-positive matches, the Interest is forwarded through one or multiple wrong faces, and then the Interest is discarded by recipient nodes. Please notice that Bloom filters drastically compress the size of information broadcast in the network and, therefore, increase the scalability of the protocol with an increasing number of nodes in the network.

Service Provider Identifier Propagation

Our first multipath session management is based upon the propagation of session identifiers using Bloom filters. There is, however, a session identifier for each created session. Therefore, each provider has to propagate multiple session identifiers in the network.

The second derived mechanism uses provider identifiers instead of session identifiers. Propagating provider identifiers, instead of session identifiers (i.e., as in the first strategy), has two main advantages. The first advantage is that there is only one identifier for a given provider. This means that only one identifier per provider needs to be propagated in the network. We equip each service provider with a randomly generated unique identifier composed of 32 characters. This identifier is propagated through the network by Interest messages. Broadcasting provider identifiers instead of session identifiers requires, however, the adaptation of the initial session management (cf. Section 4.1), while the request forwarding is based upon session identifiers. It requires us to perform changes in the presented session support mechanism integrating provider identifiers into the forwarding scheme.

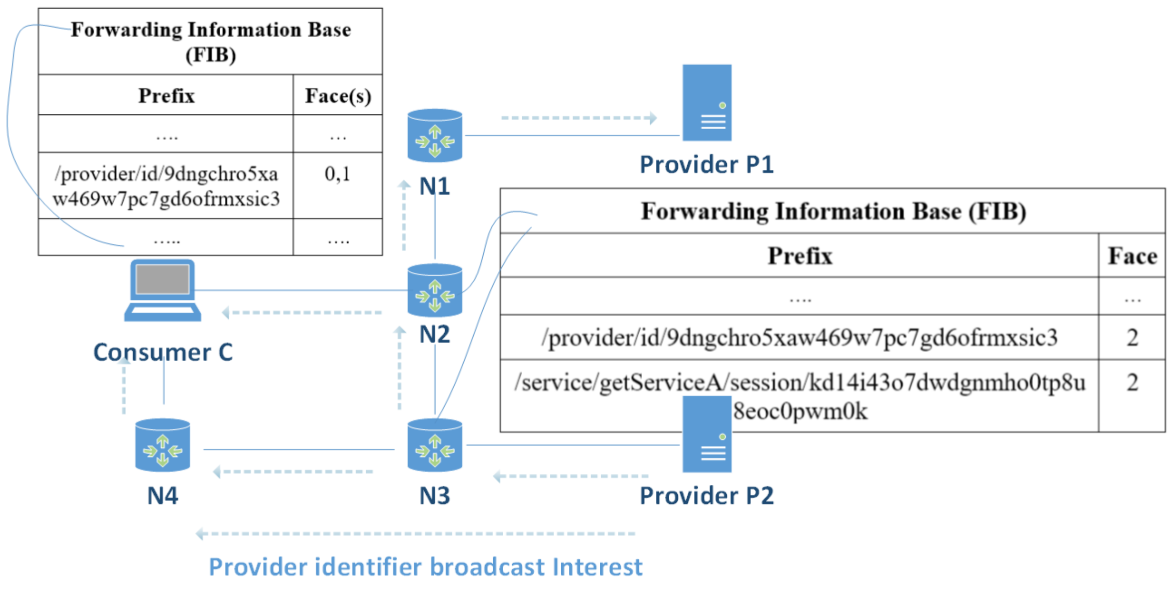


Forwarding Information Base (FIB)

Prefix	Face
....	...
/provider/id/9dngchro5xaw469w7pc7gd6ofrmxsic3	2
/service/getServiceA/session/kd14i43o7dwdgnmho0tp8u 8eoc0pwm0k	2

Figure 4-10: Session Establishment

Figure 4-10 illustrates a regular SSI/SSD handshake between consumer C and provider P2 (cf. Section 4.1). Typically, the SSD reply possesses a session identifier. In addition to previous fields, we include the provider identifier within the SSD reply. Upon SSD forwarding, the intermediate nodes add two FIB entries in their FIB tables, as shown in Figure 4-11. The first entry contains the provider identifier. It starts with the prefix "/provider/id" which holds the keywords "provider" and "id", it is then followed by the provider identifier extracted from the SSD content. The second entry is the session identifier extracted from SSD.



Forwarding Information Base (FIB)

Prefix	Face(s)
….	…
/provider/id/9dngchro5xaw469w7pc7gd6ofrmxsic3	0,1
…..	….

Forwarding Information Base (FIB)

Prefix	Face
….	…
/provider/id/9dngchro5xaw469w7pc7gd6ofrmxsic3	2
/service/getServiceA/session/kd14i43o7dwdgnmho0tp8u8eoc0pwm0k	2

Figure 4-11: Provider identifier broadcast

Once a session has been established, consumers can use the session by sending service requests using the following Interest naming convention /service/[service-identifier]/session/[provider-identifier]/[session-identifier]. In our above example, the Interest name used by the service consumer C to utilize the newly created session would look as follows:

“/service/getSearviceA/session/9dngchro5xaw469w7pc7gd6ofrmxsic3/kd14i43o7dwdgnmho0tp8u8eoc0pwm0k”

The name contains two keywords “service” and “session”. The “service” keyword is followed by the service name, while the “session” keyword is followed by the session and provider identifiers. This information enables us to forward the request to the appropriate service provider by using session or provider identifiers. The session identifier is only known by the intermediate nodes, through which the session is established. The provider identifier is broadcast with Interest messages to the entire network using a Provider Identifier Broadcast Interest (PIBI) message. Therefore, the provider identifier is recognized by all nodes in the network.

Figure 4-11 shows FIB entries in consumer C and intermediate nodes N2 and N3 after broadcasting PIBI messages. We continue using the example from the previous paragraph. Intermediate nodes N2 and N3 have two FIB entries created upon the forwarding of the SSD reply. The first entry enables the request to be forwarded using the provider identifier. The second entry allows for regular session forwarding (cf. Section 4.1). Consumer C, as well as intermediate nodes N4 and N1, maintain the identifier prefix added to their FIB tables, which was created upon receiving the PIBI message.

This strategy extends the initial session support mechanism, explained in Section 4.1. We integrate service provider identifiers (SPI). SPIs are broadcast with PIBI messages to the whole network and are used for FIB population by the nodes in the network. Additionally, compared to the regular session support mechanism, the service requests do contain not only the session identifier, but also the provider identifier. This offers alternative paths for service session request forwarding. For example, in Figure 4-11, when node N2 is down, the request from consumer C can be forwarded using the service provider identifier through node N4. The introduction of service provider identifiers lowers the message overhead dramatically, since a provider identifier is rarely changed.

Service Provider Identifier Piggybacking

The third strategy is an alternative specification of the second strategy. We do not use PIBI messages to broadcast SPI. Instead, we only rely on piggybacked SPI using SSD messages. This strategy does not create any message overhead, because it piggybacks information. Table 4-5 compares the three proposed mechanisms.

	Bloom Filter broadcast of session identifiers	**Provider identifier broadcast**	**Provider identifier piggybacking**
Multipath	Yes	Yes	Yes
Based-on	Session identifiers	Session and provider identifiers	Session and provider identifiers
Broadcasted information	Bloom filter containing session identifiers	Provider identifiers	-
Scalability	Low	High	Very High
Stability	High	Very High	Low
Advantages	Broadcasting by using Bloom filter, do not use provider identifiers	Provider identifier information only has to be broadcast in long periods	No message overhead
Disadvantages	Message overhead	The initial session support mechanism has to be extended, and provider identifiers introduced	Session information propagation may be slow. The initial session support mechanism has to be extended, and provider identifiers introduced

Table 4-5: Comparison of the three multipath mechanisms

4.2.3 Evaluation of the Multipath Strategies

This section presents an evaluation of our three multipath strategies for SCN session support.

Evaluation Scenario

The three strategies are evaluated in a network topology composed of 100 nodes. The created topology is shown in Figure 4-12. It resembles the topology of the Internet at a small-scale with high connectivity hub nodes and single connectivity leaf nodes.

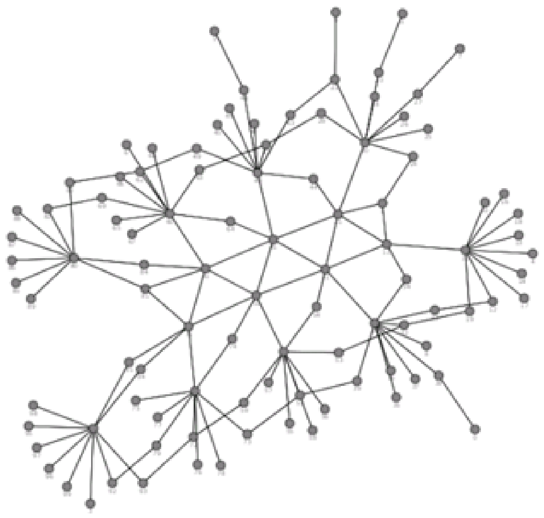


Figure 4-12: Evaluation topology

We have selected three leaf nodes as service consumers and one leaf node as a service provider. The service consumer nodes establish consecutive sessions with the service provider. Each session is used by sending 48 service Interest requests. Finally, the session is terminated. In total, 50 Interest messages are sent for each session: 1 Interest to establish a session, 48 Interests that use the session, and 1 Interest to terminate the session. We have set four evaluation scenarios where each service consumer sends 100, 200, 300, and 400 service Interest messages that need to be satisfied with the responses of the service provider. We have simulated link failures by randomly selecting links located in between the service provider and service consumers. The link failure time is randomly selected using a uniform distribution in the range [1 s, T], where T is the simulation execution time. The chosen link location is randomly selected among links connecting service consumers and the service provider. We do not, however, select links that divide the network into two disconnected components. The request frequency is set to one Interest per second. In the case of the first mechanism, the Bloom filters of size 1450 bits are broadcast every 10 seconds, but only if new sessions are created. In the second mechanism, the identifier information is broadcast at the request time of the instantiation of the first session. The third mechanism uses piggybacking with a piggybacked payload of size equal to 16 bytes, hence, there is no broadcast. For the four evaluation scenarios, in total, there are 300, 600, 900, and 1200 messages sent in the network to create, use, and terminate sessions. Each experiment is repeated ten times. The mechanisms are compared on a statistical basis.

Furthermore, we have evaluated the multipath session support mechanism and the initial single path session support scheme in terms of Interest loss rate. To ensure a fair evaluation of the Interest loss rate, we choose a random multipath topology containing 5 nodes. The topology is shown in Figure 4-13, it contains one service provider (P) and one service consumer (C) at its right and left extremities. The service consumer can reach the service provider over three different paths. The service consumer sends 2000 service session Interest requests in 10 seconds. We have repeated the evaluation ten times and calculated the ratio between the number of requests sent by the service session consumer and the number of requests that have reached the service session provider, i.e., the Interest loss rate. Intuitively, we can say that the multipath capable strategy should provide a better Interest loss rate.

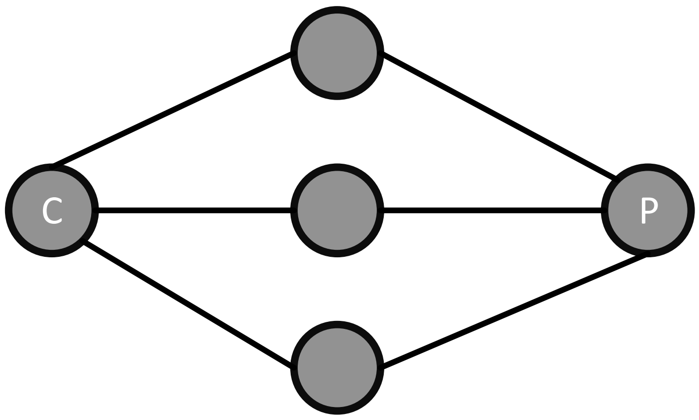


Figure 4-13: Evaluation topology for the Interest loss rate

Evaluation Results

Figure 4-14 compares the three mechanisms and the state-of-the-art session mechanism that does not support session recovery. The mechanisms are evaluated in terms of successfully transmitted session Interests against the total number of session Interests sent in the network. Please notice that Interest messages are used to start, use, and terminate sessions. Circles in Figure 4-14 display the mean number of successfully exchanged session messages. The circles are accompanied with respective 95% confidence intervals for different numbers of Interest messages sent (cf. x-axis). The provider identifier mechanism delivers all service session messages because the provider identifier is broadcast at the beginning of the evaluation execution. Network participants can save and use the provider identifier for future forwarding decisions. The Bloom filter broadcast and provider identifier piggybacking strategies do not deliver all session messages. This is due to the fact that the information about alternative paths is not immediately available after a link failure. The confidence intervals for Bloom filter and provider identifier broadcast mechanisms overlap for different numbers of session messages sent. This suggests that there is no significant difference between the mean delivered session messages for these two mechanisms. The overlap can be explained by the delayed propagation of routing information in the network.

Moreover, all our multipath support developed strategies surpass in terms of delivered messages the session support from Figure 4-1, which is not equipped with any session recovery mechanism. The session support without multipath support satisfies around 50% of Interest requests (cf. the ratio between the number of messages delivered and messages sent shown in Figure 4-1). The newly delivered strategies display much better values with the provider identifier broadcast mechanism reaching 100% for all experiments performed.

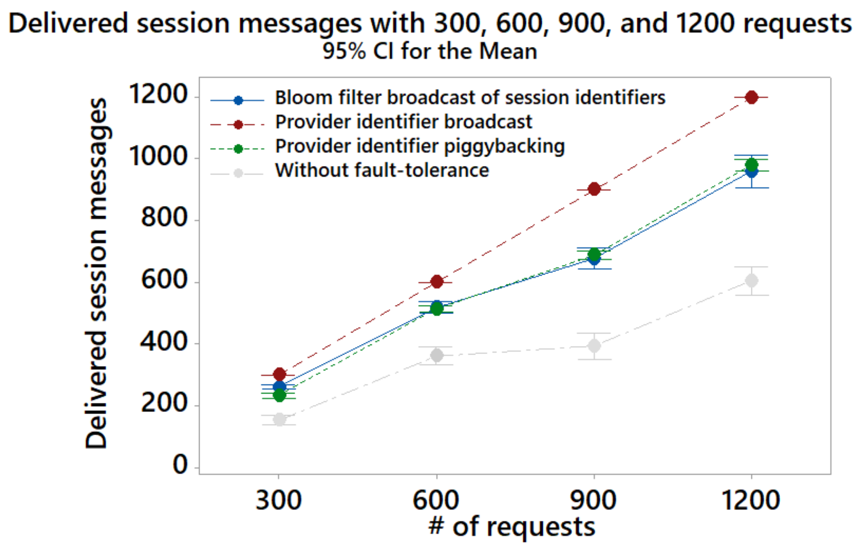


Figure 4-14: Delivered session messages for the different mechanisms and evaluation setups

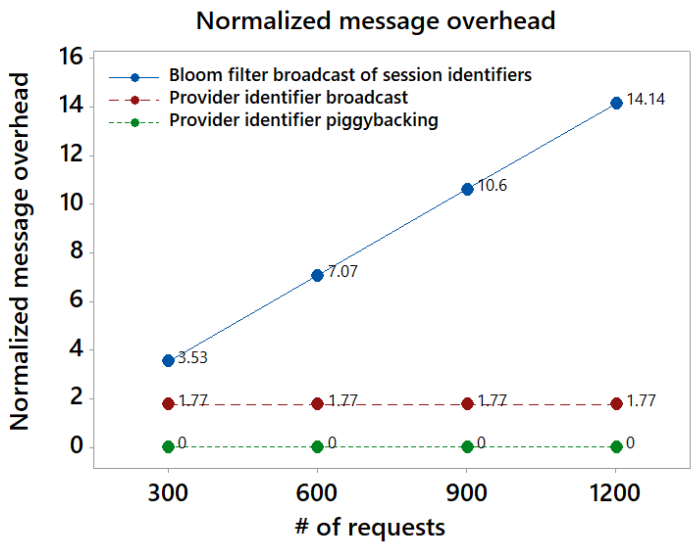


Figure 4-15: Normalized message overhead for the different mechanisms and evaluation setups

Figure 4-15 shows the normalized message overhead (NMO) for the three multipath mechanisms. The message overhead consists of messages sent by the service provider to inform the network about the existing session or provider identifiers, which enable us to use alternative paths. To draw a fair comparison, we compute the normalized message overhead (NMO) by using the formula $NMO = \frac{M}{N}$, where M is the number of transmissions concerning the propagation of session and provider identifier information for a given evaluation setup, and N is the number of distinct nodes that have received the message. Therefore, the NMO cost is expressed in overhead messages per node. Figure 4-15 shows the NMO for the three mechanisms for the different number of service Interests sent. The provider identifier broadcast mechanism has a normalized message overhead of 1.77 and offers a successful delivery of all session messages. The identifier is broadcast at the beginning. Later on, when a link failure occurs, all the messages are redirected successfully by using the provider identifier. The Bloom filter broadcast and piggybacking strategies deliver successfully a similar number of service messages; some messages are lost since there is a delay between the session establishment and the session id or service provider identifier propagation. The Bloom filter strategy has the highest overhead, because it periodically broadcasts Bloom filters, if a new session has been created since the last broadcast. The message overhead of the Bloom filter strategy grows linearly in relation to the number of requests, because of its periodically broadcasting nature. The service provider identifier piggybacking mechanism has no broadcast and reaches a similar ratio of successfully delivered session messages as the Bloom filter-based mechanism. Figure 4-16 shows the mean session recovery time with respective confidence intervals for our three mechanisms. Broadcasting of service provider identifiers delivers the best recovery time, as in the case of a failure, the nodes can immediately switch to an alternative path. The remaining mechanisms suffer from an increased recovery time caused by periodical dissemination of session identifiers in the broadcasting of session identifiers and poor network penetration of SSD messages piggybacked on regular interest messages in the piggybacking method.

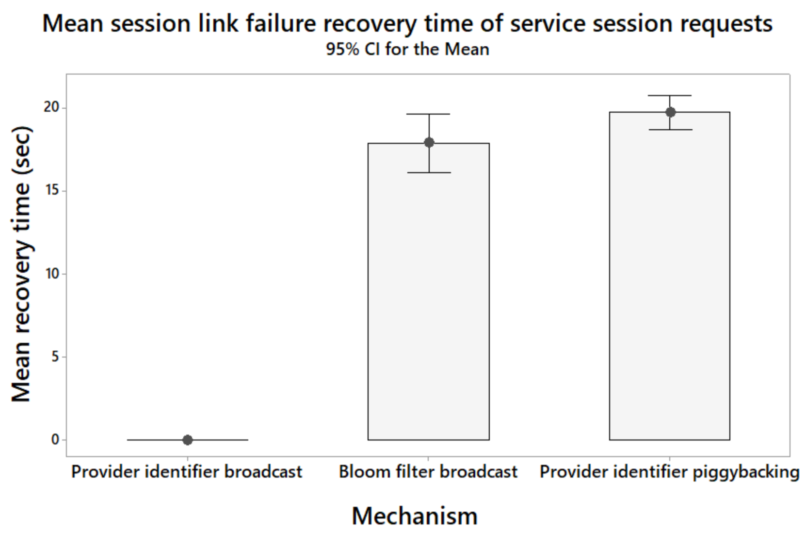


Figure 4-16: Mean session link failure recovery time of all service session requests for the three mechanisms

Figure 4-17 shows the evaluation of Interest loss rate in case of high traffic overhead in our defined scenario of 2000 service session Interest requests sent in 10 seconds. The results confirm our intuition and show decidedly the importance of multipath routing capability in case of high traffic overload for session support. The figure compares the best multipath mechanism against the initial solution without multipath support in terms of Interest loss rate. With multipath support, the Interest loss rate is 0.3% in contrast to 57% loss rate for the solution without multipath support. The significant difference in terms of loss rate endorses the importance of multipath capability for service session requests.

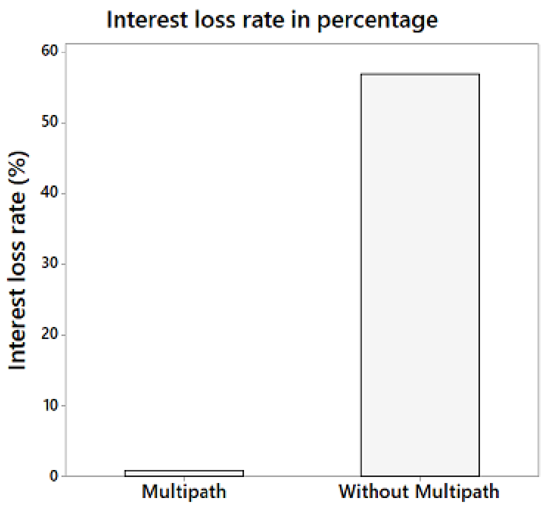


Figure 4-17: Interest loss rate for 2000 service sessions request, the loss rate is 0.3% for Multipath and 57% without Multipath support

4.3 Conclusions

To the best of our knowledge, we have designed the first service session support scheme in SCN. We extended NDN to support services and to prototype our session support. Our session support mechanism is lightweight, and it makes use of the NDN architecture with respect to the forwarding mechanism and storage. This indicates that only a few modifications were required to integrate our solution into NDN. The NDN concept was left unchanged, and a few NDN extensions were provided. Therefore, content and service traffic can be forwarded in a traditional fashion.

We have integrated our service session support mechanism in the ndnSIM simulator and evaluated its performance with common forwarding strategies. Our session support mechanism outperforms the existing strategies in the conducted evaluation.

Furthermore, in this chapter, we presented multipath session management for SCN. The first of the three mechanisms uses Bloom filters for the propagation of session identifiers, the second mechanism relies on the propagation of provider identifiers, and the third strategy uses piggybacking propagating provider identifiers. We have evaluated the presented mechanisms in terms of successful session message forwarding and message overhead in an environment with failing links. However, the solution could also be applicable for load balancing in the SCN network. The Bloom filter forwarding strategy does not rely on provider identifiers, which is an advantage if one does not want to depend on address-based forwarding. However, it displays the highest message overhead among the three mechanisms. The second strategy introduces provider identifier broadcast and has the best results in terms of successful forwarding. The third strategy relies on piggybacking; it does not have any message overhead, but it suffers from a high number of lost messages compared to the provider identifier broadcast mechanism. The evaluation in terms of Interest drop rate shows the benefit of multipath session support capability in traffic overloaded networks.

Chapter 5 Conclusions and Future Work

One of the most prominent designs of a future Internet architecture is NDN. NDN's conceptual model is a shift from the current Internet's host-centric communication model into a content-centric one. The fundamental idea of NDN is that a content requester has only to know the identifier of the content, while the content retrieval is based upon content identifiers. SCN is an extension of NDN putting its focus on services. SCN does not alter the NDN paradigm, but gracefully extends it with the desired service support.

The objective of this thesis was to evaluate the research questions posed at the beginning of this thesis (cf. Chapter 1) that have been answered perfectly as summarized in the table below.

Research Questions and Answers	
Question	***Answer***
How can we extend ICN to support SCN?	We have introduced an SCN scheme enabling service support in ICN, and we have created mechanisms that enable ICN to support short and long arguments as well as sessions.

How to enhance the system performance with mechanisms that improve service request processing time?	Load balancing was the key mechanism identified to improve request processing time. Multipath session interchanges were identified as the key aspects of fault-tolerance in session support to speed up the service request processing.

Table 5-1 Research questions and answers

While answering the research questions, we have developed a layered SCN architecture and its extensions with new communication mechanisms. Furthermore, we have presented SCN session support with multipath capabilities. These provided solutions tackle and solve the problem of efficient service request forwarding and processing through the introduction of load balancing and fault-tolerance.

5.1 Summary of Contributions

This section presents, in a nutshell, the main contributions of the thesis that extend SCN with support for services. Mainly, we focus on two SCN requirements, being efficient routing and processing of service requests and session support with multipath.

5.1.1 L-SCN Architecture

L-SCN is a two-layered SCN architecture. It provides efficient service routing. The L-SCN design splits the network into domains and specifies communication protocols for service provider information propagation. Every Node in a given domain is enriched with substantial knowledge on available resources (e.g., CPU, RAM) and services within this domain, while the communication between different domains is realized through supernodes. L-SCN integrates light-weight data structures to enhance service-related information propagation and protocol overhead. L-SCN does not require a global coordinator, while its approach is distributed, and hence, does not introduce a single point of failure. Our scheme enables the service consumer to use multiple parameters of type short and long, by following the specified naming convention as input parameters were identified as a crucial requirement for services. Service consumers should be able to send parameters tied to a service request that may be freely accessed by the service provider prior to computing. A service input parameter may be of two types, i.e., short or long. The parameter of type short is encoded as a short byte stream, while the long parameter type can be either a registered NDN content object or a locally stored data item on a service consumer. We found out that short parameters may be directly provided in request names, however, long parameters cannot be directly included in requests and require pulling from the network. An NDN content object may be directly fetched by the service provider, with traditional content requests. However, a locally stored content item on a service consumer has to be fetched through a specialized mechanism based upon the content hash provided by the service consumer.

We have extended the L-SCN architecture with additional service provider information propagation mechanisms and a DHT overlay network that enables alternative forwarding decisions in the case no entry is available in the FIB table for the requested service. The DHT-based overlay enhances the forwarding capabilities of nodes in the network by enabling them to query a DHT overlay network and receive additional request forwarding information. Initially, the L-SCN architecture had a consumer-driven information propagation approach, where the consumer had to fetch the information about services and resources from service providers. We have enhanced the information propagation mechanism strategy by introducing two information propagation mechanisms that improve the processing time and lower protocol overhead for service request processing. The two proposed mechanisms are named provider-driven and event-driven mechanisms. The event-driven mechanism propagates service provider information based on an event (e.g., high overload), while the provider-driven strategy periodically disseminates service provider information. According to our studies, the event-driven approach offers the best trade-off between service processing time and message overhead. However, the provider-driven mechanism provides the best results in terms of service request processing time.

Furthermore, we have extended the NLSR protocol with resource availability information propagation, enabling us to rank faces for service request forwarding decisions. Our solution outperforms the existing mechanisms in nlsrSIM in terms of service processing time. The final conclusion is, therefore, that L-SCN outperforms the existing forwarding strategies in ndnSIM.

5.1.2 Session Support for SCN with Multipath Capabilities

Sessions are identified as an essential component of services. Sessions enable us to establish a context between a service provider and consumer. We have presented a session support mechanism for SCN that performs service request forwarding and processing based on an established session identifier. The presented SCN session mechanism has three phases: session establishment, session use, session termination. During the session establishment phase, a two-way handshake is performed to establish the session identifier. First, the service consumer sends a specially constructed Interest message towards the service provider containing its randomly generated value. When the service provider gets the interest, it replies with a data message containing another randomly generated identifier on the provider side. The concatenation of the two randomly generated identifiers eventually forms the session identifier. Furthermore, during the two-way handshake, the intermediate nodes populate their FIB tables to ensure that future requests in a given session can be forwarded between the corresponding service consumer and provider. The actual data exchange in a session is performed through traditional NDN messages. The service consumer can use the session by sending specially crafted service requests containing the session identifier. When the session is not needed, the service provider can terminate the session by a specially formed Interest message. It is verified through simulations in ndnSIM, that the session mechanism provides better results compared to the existing solutions in terms of context provisioning, resulting in faster service processing.

Initially, the developed SCN session support mechanism was not multipath enabled. We, therefore, introduced multipath capabilities into the session support mechanism. In essence, the session identifier was known only to the intermediate forwarding nodes, through which the session has been established. The session could only be used through the path it has been created. Multipath support guarantees that independent paths can be non-simultaneously used to reach the service provider, i.e., a request is forwarded along a unique path to reach the service provider.

Introducing multipath routing capabilities, we investigated three different link-failure recovery techniques aiming to propagate session information in the network, which enables us to enhance the single path interchanges against multipath solutions. The first mechanism is based upon the propagating of session identifiers within the network using Bloom filters. The second design is based upon the propagation of service provider identifiers, while the third proposal uses piggybacking for the propagation of service provider identifiers. It is verified through simulations that multipath support enhances session support with fault-tolerance and better load balancing of service requests, especially in the case of overload. The piggybacking strategy offers the lowest results in terms of message overhead. However, the best outcome in terms of the number of delivered session requests is provided by the provider identifier propagation strategy.

5.2 Future Work

In the previous chapters of the thesis, we have presented a spectrum of contributions that satisfy SCN requirements. The core requirements are SCN architecture, input parameter support service request forwarding, and multipath session support.

It is worth mentioning that L-SCN architecture is composed of domains and supernodes. A stimulating research direction could be the supernode selection process, which aims to enable the automatic selection of supernodes in the network. An ongoing work [95] tackles this subject from a graph theory perspective, however, other techniques [96]–[98] used in P2P networks could be adapted as well. Another research direction in supernode selection could be based on a centralized or hybrid approach, where a centralized controller selects the supernodes based on the node capability and network traffic.

Furthermore, our architecture could be enhanced with other SCN techniques, such as service chaining. Currently, the service requester can only request one specific service with one request. With service composition, the service consumer can request a composition (i.e., set) of services with one shot. It is challenging to adequately combine services so that they can be efficiently requested by the service consumer with high performance.

Currently, the information about service providers is communicated to the network through specific mechanisms, which suffer from significant message overhead. An alternative approach would rely only on traffic monitoring of forwarding decisions. Nevertheless, this mechanism puts pressure on intermediate nodes requiring them to process high-load traffic volumes. In such a case, a thick intermediate node would perform considerable amount of work on traffic analysis for improved forwarding.

Furthermore, another interesting research direction is the dynamic selection of the information propagation mechanism depending on momentary network conditions. This would permit us to choose an information propagation strategy that offers the best performance according to the current situation in the network.

From the global perspective, the main goal of NDN is to substitute the IP network, however, to this end, a global-scale, native NDN network has to be first established.

Regarding the SCN session support mechanism, due to its lightweight design, it can be extended with relevant session requirements such as security [99], [100], and privacy [101]. For example, an authentication scheme would allow the service provider and the service consumer to verify the identity of each other to conduct a secured message exchange. Some recent works [102],[103] already drive in that direction. Furthermore, it is eventually worth mentioning that the SCN session support scheme is not designed to support mobility. Hence, mobile mechanisms explicitly designed for mobility use-cases are required [104], [105].

Bibliography

[1] G. Marco and M. Giugni, "SNSF | P3 Research Database | Project 139911," p. 139911, 2011.

[2] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks, 5th Edition*. 2011.

[3] V. Cerf and R. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Trans. Commun.*, vol. 22, no. 5, pp. 637–648, May 1974.

[4] Cisco, "VNI Global Fixed and Mobile Internet Traffic Forecasts," *CISCO*, 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html#~mobile-forecast.

[5] D. Comer and D. L. Stevens, *Internetworking With Tcp/Ip*. 2003.

[6] Cisco, "Cisco Visual Networking Index: Forecast and Methodology, 2008–2013, White Paper," 2009.

[7] P. Loshin, "Internet Protocol Routing," in *TCP/IP Clearly Explained*, 2003, pp. 485–507.

[8] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," Jul. 2017.

[9] G. Xylomenos *et al.*, "A Survey of information-centric networking research," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 2. pp. 1024–1049, 2014.

[10] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, Jul.

2012.

[11] D. Oran, “Information-Centric Networking Research Group (ICNRG),” pp. 11–14, 2012.

[12] D. Wu *et al.*, “Networking named content,” *Commun. ACM*, vol. 55, no. 1, p. 117, 2011.

[13] N. D. N. project team, “Named Data Networking (NDN) - A Future Internet Architecture,” 2010. [Online]. Available: http://named-data.net/.

[14] D. P. Arjunwadkar, “Introduction of NDN with Comparison to Current Internet Architecture based on TCP/IP,” *Int. J. Comput. Appl.*, vol. 105, no. 5, pp. 31–35, Nov. 2014.

[15] M. P. Papazoglou, “Service-oriented computing: concepts, characteristics and directions,” in *Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials (Cat. No.03CH37417)*, pp. 3–12.

[16] Y. Li, Y. Liu, L. Zhang, G. Li, B. Xie, and J. Sun, “An Exploratory Study of Web Services on the Internet,” in *IEEE International Conference on Web Services (ICWS 2007)*, 2007, pp. 380–387.

[17] T. Braun, V. Hilt, M. Hofmann, I. Rimac, M. Steiner, and M. Varvello, “Service-centric networking,” in *IEEE International Conference on Communications*, 2011, pp. 1–6.

[18] T. Braun, A. Mauthe, and V. Siris, “Service-centric networking extensions,” in *Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC ’13*, 2013, p. 583.

[19] D. Mansour and T. Braun, “Survey in Service-Centric Networking,” *Tech. Rep. IAM*, vol. 14–004, 2014.

[20] M. Gasparyan, T. Braun, and E. Schiller, “L-SCN: Layered SCN architecture with supernodes and Bloom filters,” in *2017 14th IEEE Annual Consumer Communications and Networking Conference, CCNC 2017*, 2017, pp. 899–904.

[21] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[22] M. Gasparyan, E. Schiller, A. Marandi, and T. I. Braun, “Communication Mechanisms for Service-Centric Networking,” in *2020 17th IEEE Annual Consumer Communications & Networking Conference (CCNC) (CCNC 2020)*, 2020.

[23] M. Gasparyan, E. Schiller, A. Marandi, and T. I. Braun, “Enhancing SCN with Routing Mechanisms,” in *IEEE/IFIP Network Operations and Management Symposium, NOMS 2020 (Submitted)*, 2020.

[24] A. K. M. M. Hoque *et al.*, “NLSR: Named-data link state routing protocol,” in *ICN 2013 - Proceedings of the 3rd, 2013 ACM SIGCOMM Workshop on Information-Centric Networking*, 2013, pp. 15–20.

[25] L. Wang, V. Lehman, A. K. M. Mahmudul Hoque, B. Zhang, Y. Yu, and L. Zhang, “A Secure Link State Routing Protocol for NDN,” *IEEE Access*, vol. 6, pp. 10470–10482, 2018.

[26] “GitHub - named-data/NLSR: Named Data Link State Routing.” [Online]. Available: https://github.com/named-data/NLSR.

[27] M. Gasparyan, T. Braun, and E. Schiller, “IaDRA-SCN: Intra-domain routing architecture for service-centric networking,” in *2018 IEEE International Conference on Communications Workshops, ICC Workshops 2018 - Proceedings*, 2018, pp. 1–6.

[28] M. Gasparyan, G. Corsini, T. Braun, E. Schiller, and J. Saltarin, “Session support for SCN,” in *2017 IFIP Networking Conference, IFIP Networking 2017 and Workshops*, 2018, vol. 2018-Janua, pp. 1–6.

[29] M. Gasparyan, A. Marandi, E. Schiller, and T. Braun, “Fault-Tolerant Session Support for Service-Centric Networking,” in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019.

[30] S. Ratnasamy *et al.*, “A scalable content-addressable network,” in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM ’01*, 2001, vol. 31, no. 4, pp. 161–172.

[31] P. Felber, P. Kropf, E. Schiller, and S. Serbu, “Survey on load balancing in peer-to-peer distributed hash tables,” *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 473–492, 2014.

[32] D. D. Clark, W. Lehr, S. Bauer, P. Faratin, R. Sami, and J. Wroclawski, “The Growth of Internet Overlay Networks: Implications for Architecture, Industry Structure and Policy,” *33rd Res. Conf. Commun. Inf. Internet Policy, Arlington, Virginia*, pp. 1–50, 2005.

[33] Z. Zhu and A. Afanasyev, “Let’s ChronoSync: Decentralized dataset state synchronization in Named Data Networking,” in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, 2013, pp. 1–10.

[34] I. Aad, C. Castelluccia, J.-P. P. Hubaux, C. Castclluccia, and J.-P. P. Hubaux, “Packet Coding for Strong Anonymity in Ad Hoc Networks,” in *2006 Securecomm and Workshops*, 2006, pp. 1–10.

[35] M. Raya, P. Papadimitratos, I. Aad, D. Jungels, and J. P. Hubaux, “Eviction of misbehaving and faulty nodes in vehicular networks,” *IEEE J. Sel. Areas Commun.*, vol. 25, no. 8, pp. 1557–1568, Oct. 2007.

[36] A. Marandi, T. Braun, K. Salamatian, and N. Thomos, “BFR: A bloom filter-based routing approach for information-centric networks,” in *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, 2017, vol. 2018-Janua, pp. 1–9.

[37] A. Marandi, T. Braun, K. Salamatian, and N. Thomos, “Pull-based Bloom Filter-based Routing for Information-Centric Networks,” *2019 16th IEEE Annu. Consum. Commun. Netw. Conf. CCNC 2019*, pp. 1–6, Sep. 2018.

[38] I. Stoica *et al.*, “Chord: A scalable peer-to-peer lookup protocol for Internet

applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, Feb. 2003.

[39] L. Zhang *et al.*, "Named data networking," *Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[40] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," *Univ. California, Los Angeles, Tech. Rep*, vol. 4, 2012.

[41] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2 . 0 : A New Version of the NDN Simulator for NS-3," *University of California, Los Angeles, Tech. Rep. NDN-0028*, pp. 1–8, 2015.

[42] A. Afanasyev, S. Mastorakis, I. Moiseenko, and L. Zhang, "ndnSIM Documentation — ndnSIM documentation," 2017. [Online]. Available: https://ndnsim.net/current/.

[43] G. F. Riley and T. R. Henderson, "The ns-3 Network Simulator," in *Modeling and Tools for Network Simulation*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34.

[44] Ns-3, "ns-3 | a discrete-event network simulator for internet systems," 2018. [Online]. Available: https://www.nsnam.org/.

[45] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, p. 181, 2007.

[46] V. Dimitrov and V. Koptchev, "PSIRP project -- publish-subscribe internet routing paradigm," in *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies - CompSysTech '10*, 2010, p. 167.

[47] N. Fotiou, D. Trossen, and G. C. Polyzos, "Illustrating a publish-subscribe Internet architecture," in *Telecommunication Systems*, 2012, vol. 51, no. 4, pp. 233–245.

[48] C. Dannewitz, J. Golić, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *Proceedings - IEEE INFOCOM*, 2010, pp. 1–6.

[49] "CCNx." [Online]. Available: https://wiki.fd.io/view/Cicn.

[50] "Interest Packet — NDN Packet Format Specification 0.3 documentation." [Online]. Available: https://named-data.net/doc/NDN-packet-spec/current/interest.html.

[51] "Interest Packet — NDN Packet Format Specification 0.3 documentation." [Online]. Available: https://named-data.net/doc/NDN-packet-spec/current/data.html.

[52] "NDN Packet Format Specification — NDN Packet Format Specification 0.3 documentation." [Online]. Available: http://named-data.net/doc/NDN-packet-spec/current/.

[53] A. Afanasyev, J. Shi, B. Zhang, … L. Z.-D. C. S., and U. 2014, "NFD developer's guide," *named-data.net*.

[54] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet," in *2007 IEEE International Conference on Communications*, 2007, pp. 1899–1904.

[55] D. Nurmi *et al.*, "The eucalyptus open-source cloud-computing system," in *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGRID 2009*, 2009, pp. 124–131.

[56] "AWS | Amazon EC2 – Service d'hébergement cloud évolutif," *Amazon Web Services, Inc*, 2019. [Online]. Available: https://aws.amazon.com/fr/ec2/.

[57] S. Shanbhag, N. Schwan, I. Rimac, and M. Varvello, "SoCCeR: Services over content-centric routing," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, 2011, pp. 62–67.

[58] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 26, no. 1, pp. 29–41, 1996.

[59] K.-L. Du and M. N. S. Swamy, “Ant Colony Optimization,” in *Search and Optimization by Metaheuristics*, Bradford Company, 2016, pp. 191–199.

[60] T. Stützle and H. H. Hoos, “MAX-MIN Ant System,” *Futur. Gener. Comput. Syst.*, vol. 16, no. 8, pp. 889–914, Jun. 2000.

[61] M. Heissenbüttel and T. Braun, “Ants-based routing in large scale mobile ad-hoc networks,” in *Proceedings of Kommunikation in Verteilten Systemen (KiVS 2003)*, 2003, pp. 91–99.

[62] H. Zhang, X. Wang, P. Memarmoshrefi, and D. Hogrefe, “A Survey of Ant Colony Optimization Based Routing Protocols for Mobile Ad Hoc Networks,” *IEEE Access*, vol. 5, pp. 24139–24161, 2017.

[63] J. Lv, X. Wang, and M. Huang, “Ant Colony Optimization-Inspired ICN Routing with Content Concentration and Similarity Relation,” *IEEE Commun. Lett.*, vol. 21, no. 6, pp. 1313–1316, Jun. 2017.

[64] C. Li, W. Liu, L. Wang, M. Li, and K. Okamura, “Energy-efficient quality of service aware forwarding scheme for Content-Centric Networking,” *J. Netw. Comput. Appl.*, vol. 58, pp. 241–254, Dec. 2015.

[65] Q. Huang and F. Luo, “Ant-colony optimization based QoS routing in named data networking,” *J. Comput. Methods Sci. Eng.*, vol. 16, no. 3, pp. 671–682, Oct. 2016.

[66] J. Lv, X. Wang, and M. Huang, “ACO-inspired ICN Routing Scheme with Density-Based Spatial Clustering,” Springer, Cham, 2017, pp. 112–117.

[67] J. Lv, X. Wang, K. Ren, M. Huang, and K. Li, “ACO-inspired Information-Centric Networking routing mechanism,” *Comput. Networks*, vol. 126, pp. 200–217, Oct. 2017.

[68] S. Srinivasan *et al.*, “CCNxServ: Dynamic service scalability in information-centric networks,” in *IEEE International Conference on Communications*, 2012, pp. 2617–2622.

[69] "The CCNx Distillery software distribution." [Online]. Available: https://github.com/parc-ccnx-archive/CCNx_Distillery.

[70] S. R. Srinivasan *et al.*, "NetServ," in *Proceedings of the 2009 workshop on Re-architecting the internet - ReArch '09*, 2009, p. 37.

[71] "JAR File Specification," *Oracle*. [Online]. Available: https://docs.oracle.com/javase/6/docs/technotes/guides/jar/jar.html#JARIndex.

[72] A. Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad; Buckley, "The Java® Language Specification," *Addison-Wesley*, p. 688, 2014.

[73] E. Nordström *et al.*, "Serval: an end-host stack for service-centric networking," *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, pp. 7–7, 2012.

[74] D. Mansour, T. Braun, and C. Anastasiades, "NextServe Framework: Supporting Services over Content-Centric Networking," 2014, pp. 189–199.

[75] Q. Wu *et al.*, "SOFIA: Toward service-oriented information centric networking," *IEEE Netw.*, vol. 28, no. 3, pp. 12–18, May 2014.

[76] C. Tschudin and M. Sifalakis, "Named functions for media delivery orchestration," in *2013 20th International Packet Video Workshop, PV 2013*, 2013, pp. 1–8.

[77] C. Tschudin and M. Sifalakis, "Named functions and cached computations," in *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, 2014, pp. 851–857.

[78] I. Król Michałand Psaras, M. Król, and I. Psaras, "NFaaS," in *Proceedings of the 4th ACM Conference on Information-Centric Networking - ICN '17*, 2017, pp. 134–144.

[79] A. Madhavapeddy and D. J. Scott, "Unikernels: Rise of the Virtual Library Operating System," *Commun. ACM*, vol. 57, no. 1, pp. 61–69, Aug. 2014.

[80] I. Baldini *et al.*, “Serverless computing: Current trends and open problems,” in *Research Advances in Cloud Computing*, Singapore: Springer Singapore, 2017, pp. 1–20.

[81] “Named Data Networking: Motivation &amp; Details - Named Data Networking (NDN).” [Online]. Available: https://named-data.net/project/archoverview/.

[82] B. Kaliski and M. Robshaw, “Message authentication with MD5,” *CryptoBytes, Sping*, 1995.

[83] H. Handschuh, “SHA Family (Secure Hash Algorithm),” in *Encyclopedia of Cryptography and Security*, Springer US, 2005, pp. 565–567.

[84] A. Rahman, R. Ravindran, D. Kutscher, and D. Trossen, “Deployment Considerations for Information-Centric Networking (ICN).” 2019.

[85] “Get NFD Connected - Named Data Networking (NDN).” [Online]. Available: https://named-data.net/2015/01/06/get-nfd-connected/.

[86] “overlay network Archives - Named Data Networking (NDN).” [Online]. Available: https://named-data.net/tag/overlay-network/.

[87] “Named Data Networking Forwarding Daemon (NFD).” [Online]. Available: http://named-data.net/doc/NFD/current/INSTALL.html.

[88] A. Jangam and D. Sidhu, “nlsrSIM: Porting and simulation of named-data link state routing protocol into ndnSIM,” in *DIVANet 2017 - Proceedings of the 6th ACM Symposium on Development and Analysis of Intelligent Vehicular Networks and Applications, Co-located with MSWiM 2017*, 2017, pp. 39–46.

[89] “GitHub - 3rd-ndn-hackathon/ndnSIM-NLSR.” [Online]. Available: https://github.com/3rd-ndn-hackathon/ndnSIM-NLSR.

[90] A. Medina, A. Lakhina, I. Matta, and J. Byers, “BRITE: an approach to universal topology generation,” in *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*,

pp. 346–353.

[91] A. L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science (80-.).*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[92] E. Kalogeiton, T. Kolonko, and T. Braun, “A multihop and multipath routing protocol using NDN for VANETs,” in *2017 16th Annual Mediterranean Ad Hoc Networking Workshop, Med-Hoc-Net 2017*, 2017, pp. 1–8.

[93] C. Anastasiades, J. Weber, and T. Braun, “Dynamic Unicast: Information-centric multi-hop routing for mobile ad-hoc networks,” *Comput. Networks*, vol. 107, pp. 208–219, Oct. 2016.

[94] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, “On the role of routing in named data networking,” in *Proceedings of the 1st international conference on Information-centric networking - INC ’14*, 2014, pp. 27–36.

[95] S. A. Marandi, V. Hofer, M. Gasparyan, T. Braun, and N. Thomos, “Bloom Filter-based Routing for Dominating Set-based Service-Centric Networks,” in *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2020*.

[96] V. Lo, D. Zhou, Y. Liu, C. GauthierDickey, and J. Li, “Scalable supernode selection in peer-to-peer overlay networks,” in *Proceedings - Second International Workshop on Hot Topics in Peer-to-Peer Systems, HOT-P2P 2005*, 2005, vol. 2005, pp. 18–27.

[97] J. Liang, R. Kumar, and K. W. Ross, “Understanding kazaa.” submitted, 2004.

[98] A. Durand, M. Gasparian, T. Rouvinez, I. Aad, T. Braun, and T. A. Trinh, “BitWorker, a Decentralized Distributed Computing System Based on BitTorrent,” 2015, pp. 151–164.

[99] H. Dai, Y. Wang, J. Fan, and B. Liu, “Mitigate DDoS attacks in NDN by interest traceback,” in *2013 IEEE Conference on Computer Communications Workshops*

(INFOCOM WKSHPS), 2014, pp. 381–386.

[100] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang, "Interest flooding attack and countermeasures in Named Data Networking," in *2013 IFIP Networking Conference*, 2013, pp. 1–9.

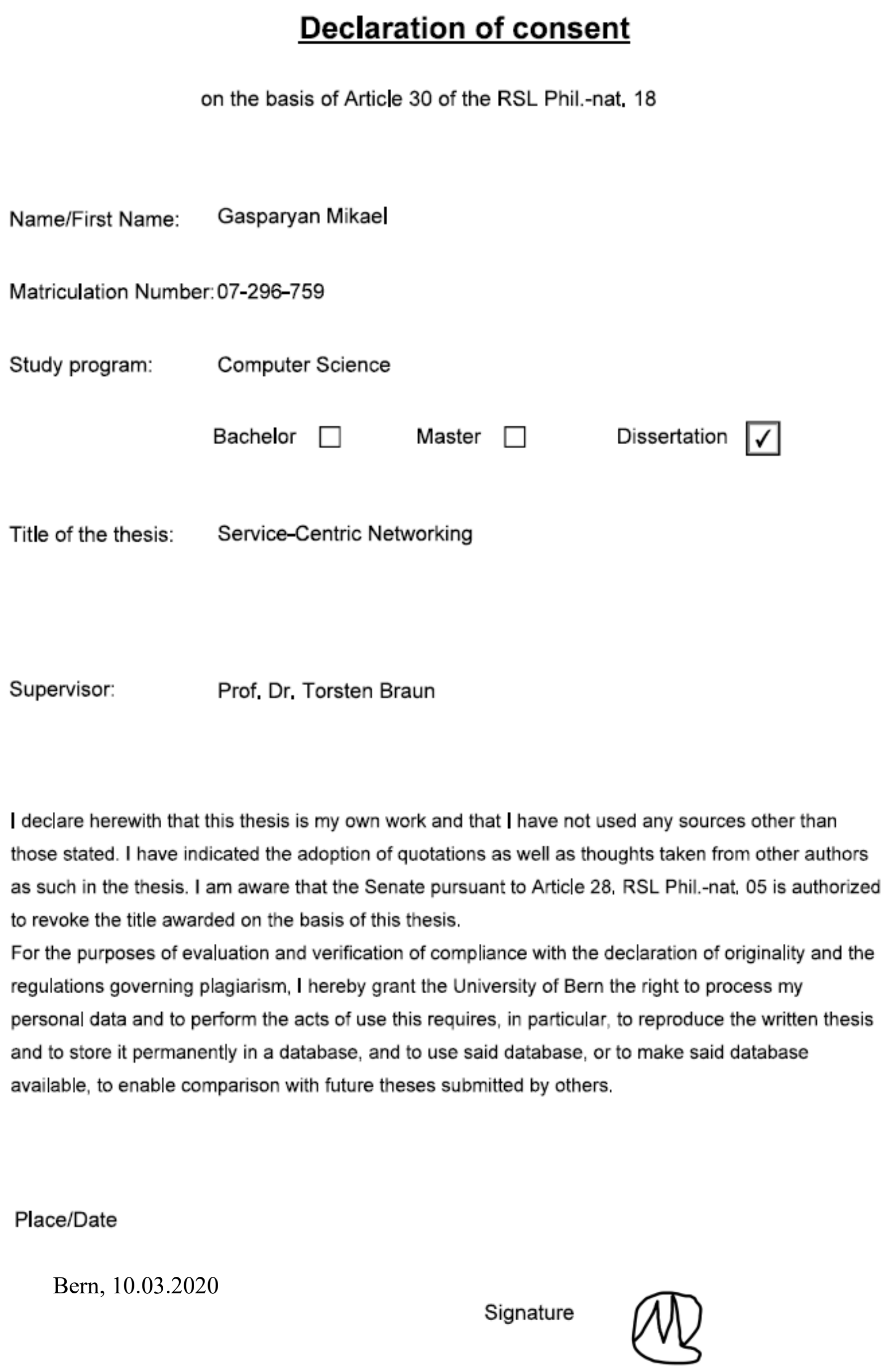
[101] R. Tourani, S. Misra, T. Mick, and G. Panwar, "Security, Privacy, and Access Control in Information-Centric Networking: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 1. pp. 556–600, 2018.

[102] M. Król, K. Habak, D. Oran, D. Kutscher, and I. Psaras, "Rice: Remote method invocation in ICN," in *ICN 2018 - Proceedings of the 5th ACM Conference on Information-Centric Networking*, 2018, pp. 1–11.

[103] I. Aad, T. Braun, and D. Mansour, "Authentication and Trust in Service-Centric Networking," in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, 2016, pp. 563–566.

[104] Y. Zhang, A. Afanasyev, J. Burke, and L. Zhang, "A survey of mobility support in Named Data Networking," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2016, pp. 83–88.

[105] E. Schiller, N. Nikaein, E. Kalogeiton, M. Gasparyan, and T. Braun, "CDS-MEC: NFV/SDN-based Application Management for MEC in 5G Systems," *Comput. Networks*, vol. 135, pp. 96–107, Apr. 2018.



Declaration of consent

on the basis of Article 30 of the RSL Phil.-nat. 18

Name/First Name:	Gasparyan Mikael
Matriculation Number:	07-296-759
Study program:	Computer Science
	Bachelor ☐ Master ☐ Dissertation ✓
Title of the thesis:	Service-Centric Networking
Supervisor:	Prof. Dr. Torsten Braun

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 28, RSL Phil.-nat. 05 is authorized to revoke the title awarded on the basis of this thesis.

For the purposes of evaluation and verification of compliance with the declaration of originality and the regulations governing plagiarism, I hereby grant the University of Bern the right to process my personal data and to perform the acts of use this requires, in particular, to reproduce the written thesis and to store it permanently in a database, and to use said database, or to make said database available, to enable comparison with future theses submitted by others.

Place/Date

Bern, 10.03.2020

Signature [illegible]

MIKAEL GASPARYAN

+41 76 307 43 17
gasparyan@inf.unibe.ch

Rue du Mont-Noble 30
CH-3960, Sierre

EDUCATION

PhD	Computer Science, University of Bern, Switzerland	May 2020
MS	Computer Science, University of Fribourg, Switzerland	April 2015
BS	Information Technology, HES-SO Valais, Switzerland	September 2012

PUBLICATIONS

[1] Marandi, Sayed Ali; Hofer, Vincent; **Gasparyan, Mikael**; Braun, Torsten; Thomos, Nikolaos, Bloom Filter-based Routing for Dominating Set-based Service-Centric Networks, in IEEE/IFIP Network Operations and Management Symposium, NOMS 2020.

[2] **Gasparyan, Mikael**; Schiller, Eryk Jerzy; Marandi, Sayed Ali; Braun, Torsten. Communication Mechanisms for Service-Centric Networking. In: 2020 17th IEEE Annual Consumer Communications & Networking Conference (CCNC).

[3] **Gasparyan, Mikael**; Marandi, Sayed Ali; Schiller, Eryk Jerzy; Braun, Torsten (2019). Fault-Tolerant Session Support for Service-Centric Networking. In: IFIP/IEEE International Symposium on Integrated Network Management (IM).

[4] **Gasparyan, Mikael**; Braun, Torsten; Schiller, Eryk Jerzy (2018). IaDRA-SCN: Intra-domain routing architecture for Service-Centric Networking. In: IEEE International Conference on Communications Workshop on Information-Centric Edge Computing and Caching for Future Networks (ICECC) (p. 1-6). IEEE 10.1109/ICCW.2018.8403718

[5] Schiller, Eryk Jerzy; Nikaein, Navid; Kalogeiton, Eirini; **Gasparyan, Mikael**; Braun, Torsten (2018). CDS-MEC: NFV/SDN-based application management for MEC in 5G Systems. Computer Networks, 135, p. 96-107. Elsevier 10.1016/j.comnet.2018.02.013

[6] **Gasparyan, Mikael**; Corsini, Guillaume; Braun, Torsten; Schiller, Eryk Jerzy; Saltarin de Arco, Jonnahtan Eduardo (2017). Session Support for SCN. In: IFIP Networking 2017 workshop on Information-Centric Fog Computing (ICFC). IFIP Open Digital Library 10.23919/IFIPNetworking.2017.8264879

[7] **Gasparyan, Mikael**; Braun, Torsten; Schiller, Eryk Jerzy (2017). L-SCN: layered SCN architecture with Supernodes and Bloom Filters. In: The 14th Annual IEEE Consumer Communications & Networking Conference (CCNC) (p. 899-904). IEEE 10.1109/CCNC.2017.7983252

[8] Durand, Arnaud; **Gasparyan, Mikael**; Rouvinez, Thomas; Aad, Imad Rafic; Braun, Torsten; Trinh, Tuan Anh (2015). BitWorker, a Decentralized Distributed Computing System based on BitTorrent. In: Wired/Wireless Internet Communications. The 13th International Conference on Wired & Wireless Internet Communications. Lecture Notes in Computer Science: Vol. 9071 (p. 151-164). Cham: Springer 10.1007/978-3-319-22572-2_11