Objective FP7-ICT-2009-5-257448/D-3.1

Future Networks

Project 257448

"SAIL – Scalable and Adaptable Internet Solutions"

D-3.1
(D-B.1) The Network of Information: Architecture and Applications

Date of preparation: **11-07-31**
Start date of Project: **10-08-01**
Project Coordinator: **Thomas Edwall**
Ericsson AB

Revision: **1.0**
Duration: **13-01-31**

Document Properties

Document Number:	D-3.1
Document Title:	**The Network of Information: Architecture and Applications**
Document Responsible:	Dirk Kutscher (NEC)
Document Editor:	Petteri Pöyhönen (NSN), Ove Strandberg (NSN)
Authors:	Bengt Ahlgren (SICS), Matteo D. Ambrosio (TI), Erik Axelsson (KTH), Lars Brown (KTH), Christian Dannewitz (UPB), Zoran Despotovic (DoCoMo), Anders E. Eriksson (EAB), Stephen Farrell (TCD), Jelena Frtunikj (DoCoMo), Massimo Gallo (FT), Björn Grönvall (SICS), Claudio Imbrenda (NEC), Bruno Kauffmann (FT), Dirk Kutscher (NEC), Anders Lindgren (SICS), Henrik Lundqvist (DoCoMo), Aidan Lynch (TCD), Luca Muscariello (FT), Börje Ohlman (EAB), Jean Francois Peltier (FT), Karl-Ake Persson (EAB), Petteri Pöyhönen (NSN), Ove Strandberg (NSN), Patrick Truong (FT), Janne Tuononen (NSN), Vinicio Vercellone (TI), Stefan Weber (TCD)
Target Dissemination Level:	PU
Status of the Document:	Final version
Version:	1.0

Production Properties:

Reviewers:	Pedro Aranda (TID), Björn Levin (SICS)

Document History:

Revision	Date	Issued by	Description
1.0	2011-07-31	Petteri Pöyhönen	Final version

Disclaimer:

Abstract:

This deliverable D.B.1 "Architecture and Applications" describes the initial *NetInf* architecture and corresponding applications. NetInf is an ICN approach, i.e., an approach to networking that is based on the notion of providing access to named information – a different paradigm compared to the host-to-host communication model of today's Internet. The addressing of data in the network introduce additional functionality and they are presented within architectural building blocks. The different architectural building blocks are further studied in 5 application prototype scenarios for the NetInf prototyping work. The prototypes are used to study the potential gains resulted in the use of NetInf concepts and to verify the NetInf design. First results have been submitted as contributions to IETF standardisation and IRTF work, for example elements related to naming, caching and delay tolerant networking.

Keywords:

Network of Information, NetInf, Information-Centric Networking, Internet, Architecture, Applications, SAIL

Executive Summary

This document is a public deliverable of the Scalable and Adaptive Internet Solutions (SAIL) EU-FP7 project [1] and describes the initial *Network of Information (NetInf)* architecture and corresponding applications. NetInf is an Information-Centric Networking (ICN) approach, i.e., an approach to networking that is based on the notion of providing access to named information – a different paradigm compared to the host-to-host communication model of today's Internet.

There are different design options for ICN, and this document presents the specific NetInf design that is centered around a well-defined set of architecture invariants (such as unique naming, location-independence and a strict information-centric service model) and puts particular emphasis on supporting multi-technology/multi-domain interoperability. These variants provide the necessary orientation for the more detailed architecture elements such as naming, transport, caching, etc. Based on these invariants, an overview of a complete NetInf system that illustrates the relation of the architecture elements in a big picture is presented. The presented architecture building blocks can be used to construct full or parts of the NetInf architecture such as name resolution, forwarding and routing, mobility and security for different use cases and migration paths.

The different architectural building blocks are further studied in 5 application prototype scenarios for the NetInf prototyping work and they are as follows; i) Localised Content Delivery Network (CDN), ii) Event with Large Crowds, iii) Delay Tolerant Video Distribution, iv) Active Information Objects, and v) Local Collaboration. These scenarios represent examples of such cases where NetInf can provide substantial benefits for different stakeholders such as end users, content providers, operators and Internet Service Providers (ISPs) compared to the current deployment practices where the related applications are implemented without NetInf components. In some cases, NetInf can act as an enabler for a large scale application deployment. The prototypes are used to study the potential gains resulted in the use of NetInf concepts and to verify the NetInf design.

Finally, this documents describes recent and on-going work on NetInf migration and standardisation activities – important activities in conjunction with architecture research to ensure an effective and eventually successful technology development. Some NetInf elements are applicable to current Internet technology development and standardisation topics related to ICN (such as Internet Engineering Task Force (IETF) work in Decoupled Application Data Enroute (DECADE) and Peer to Peer Streaming Protocol (PPSP) working groups), and other elements are core ICN technologies that are candidates for genuine ICN standardisation that have also been submitted to the IETF and Internet Research Task Force (IRTF).

Contents

1 Introduction

Information-Centric Networking (ICN) is one of the most promising directions for the Internet to ensures its reliability and scalability also in the future and it can be implemented in different ways. The approach Network of Information (NetInf) of the Scalable and Adaptive Internet Solutions (SAIL) project is centered around a well-defined set of architecture invariants (such as unique naming, location-independence and a strict information-centric service model) and puts particular emphasis on supporting multi-technology/multi-domain interoperability.

This document reports the current results of the on-going work in the SAIL NetInf. These results focus on two main areas: i) architectural work and ii) use case and application scenario work. NetInf is leveraging the 4WARD results on naming and name resolution as elements for an overall Information-Centric Networking (ICN) framework that can be applied to native all-ICN networks but can also be deployed on-top of existing network infrastructures to ensure feasible migration paths. In addition, compared to what was done in the 4WARD project, SAIL NetInf adds new concepts like Delay-Tolerant Networking (DTN). It also takes into account developments elsewhere in ICN research (e.g., Content Centric Networking (CCN), Data-Oriented Network Architecture (DONA) and Publish-Subscribe Internetworking Routing Paradigm (PSIRP)). A particular focus is put on the possibility of inter-connecting different NetInf domains, each with their own object retrieval mechanisms and policies in order to better support network heterogeneity, accommodate different stake holders and to enable possibilities for co-existence with and migration from other, existing networks.

This support for heterogeneity requires a careful approach towards architecture specification, since a monolithic architecture is likely to be too rigid to enable a successful evolution and adoption of NetInf as a general Internet-scale technology. Therefore the choice is to explicitly cater for heterogeneity and evolvability through a framework of architecture invariants that ensures general interoperability but without comprising heterogeneity, evolvability and innovation. The main research objectives in the architecture area are to define the common ICN concepts and terminology by which information-centric networking mechanisms for naming, routing, cache management and transporting the Information Objects (IOs) are created.

Prototypes and demonstrations are integral part of the NetInf work and those are and will be used to study viability of the chosen ICN invariants, common real-life application scenarios and deployment and migration aspects. This work is one of the key elements on the road for ensuring readiness of ICN for deployment and standardisation with documented experimentation, simulation and measurement results.

The work in the SAIL project is taking other ICN approaches into consideration. There are several relevant projects and studies that have influenced the work like 4WARD, CCN, DONA and PSIRP. The 4WARD project has special impact as the basic NetInf design originated in 4WARD. There has been more focus on simplification, actual protocols and migration in SAIL and the work is evolving to make components of NetInf available in the Internet.

The rest of this document is structured as follows. Chapter 2 introduces the central concepts and terminology for understanding ICN related mechanisms introduced later. Additionally Chapter 2 also summarises other main stream research results in the field of the ICN and explains the main differences between the 4WARD and SAIL NetInf concepts and designs. Chapter 3 describes the developed information-centric networking mechanisms and explains their relationships. Chapter 5 concludes the use case and scenario work done and introduces the set of application scenarios that will be used in the prototyping. Finally, Chapter 6 summarises the first year results, lists the prototyping activities and their status and describes the standardisation and migration efforts and plans.

2 ICN Concepts and Related Works

In this chapter, we discuss the general ICN concepts independent of a specific architectural approach (Section 2.1) and provide a summary of the most relevant ICN research activities and concepts (Section 2.2).

2.1 General Concepts and Terminology of Information-centric Networks

This section defines some common terminology and building blocks that are used in multiple ICN architectures. Consequently, not all building blocks described here necessarily have to be part of an ICN architecture. A previous version of this overview has been published in [2].

2.1.1 Information Objects

In the following, the terms data and information as well as the term *content* are used interchangeably unless explicitly stated.

The ICN concept focuses on the information itself and not on the storage location of this information. To highlight this differentiation, we introduce the concept of an *IO*. An IO represents the information itself independent of its storage location and physical representation. An IO can have multiple different *representations*, i.e., unique bit patterns representing the same information, e.g., different encodings. Finally, there can be multiple different *copies* of each representation, e.g., stored on a server, a client node, or in a cache. Consequently, an IO refers to the group of all representations and corresponding copies of the same information. In addition, there can be *metadata* associated with the IO, i.e., data about the represented information, e.g., encoding-related information or security-related information. This metadata is implicitly associated with all representations and copies in the subtree below the IO. Figure 2.1 illustrates the relation between an *IO*, its *representations*, and *copies* of the representations. Please note that the IO as introduced here is just an abstraction that is independent of whether an ICN architecture is actually implementing this abstraction as a concept.

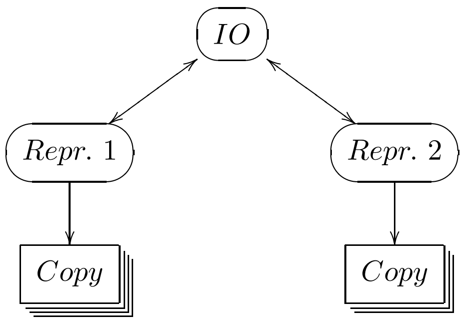

Figure 2.1: An Information Object (*IO*) with two representations (*reps*), each with multiple *copies*

There are multiple different ways to implement the IO concept. For example, one ICN incarnation could implement the IO concept as a dedicated data structure containing all *locators* (i.e., addresses) pointing to all respective copies. Such an implementation could additionally store the metadata as part of an IO. If IOs are represented by a dedicated data structure, an *information model* is required that defines the syntax and semantics of IOs. In a different ICN incarnation, the IO concept might not be represented by any dedicated data structure at all. Here, the copies might be self-contained and include the metadata.

2.1.2 Naming and Security

Naming information plays an important role in the ICN concept. In today's Internet and related applications, mainly the storage location of information is named. For example, files are names using Uniform Resource Locators (URLs) relating to a network node and file structure. IP addresses are used to name the interfaces of the respective storage nodes. In ICNs, the information itself is named, i.e., the Information Objects is named via *location-independent Object IDentifiers (IDs)*. Note that *IDentifier* and *name* are used interchangeably in this document.

Naming is closely related to security in several ICN architectures [3, 4]. In today's Internet architecture, security is an add-on to the original architecture and is mainly based on trusting the source of information via authentication and securing the data channel via encryption. In the ICN concept, security cannot be bound to the storage location as the network and/or user should benefit from any available copy. Consequently, new *information-centric security* concepts are required that base security on the information itself. A popular approach followed by several ICN architectures is to integrate security aspects and the naming concept, i.e., the object IDs. The following five general technical security goals are defined in the ICN context:

- Confidentiality: Only eligible entities (i.e., users or systems) can read secured information, i.e., IOs and corresponding metadata.
- Data integrity: It is possible to identify accidental or intentional changes to IOs and the corresponding metadata. This is possible as the name is constructed in a way that provides name/data integrity. This property is also referred to as *self-certification*.
- Accountability: The owner/creator of information can be authenticated and/or identified. There is explicit differentiation between:
  - Owner continuity/pseudonymity: Binds the information securely to a virtual entity, represented, e.g., by a pseudonym or a public/private key pair.
  - Owner identification: Binds the information securely to a real-world entity, e.g., a person's unique identity or an institution.
- Availability: The IOs and corresponding metadata published in the network have to be available and accessible for (authorised) entities.
- Controlled access: Access (i.e., read, write, execute) to IOs and/or corresponding metadata can be restricted to authorised entities.

2.1.3 Application Programming Interface

ICN architectures typically have *information-centric Application Programming Interfaces (APIs)* that are location-independent. The APIs have a *pull-based* interaction pattern, i.e., the user/application requests IOs based on object IDs. Several approaches integrate a *publish/subscribe* component or *event service* component. The resulting major primitives of an information-centric API are of the type *REQUEST(ID)* and *PUBLISH(ID)*. The REQUEST(ID) primitive might involve a direct request of a fully or partly specified object ID, or a subscription for a certain type of information. The PUBLISH(ID) primitive involves some kind of publication and/or registration of new information in the network, depending on the specific architecture.

2.1.4 Routing and Transport

As the object IDs are location-independent, it is not possible to use common topology-based routing and forwarding algorithms based on these object IDs. There are two general approaches in ICNs to handle routing. They strongly depend on the properties of the object name space, mainly if the names are aggregatable or not.

The first approach uses a *Name Resolution System (NRS)* that stores bindings from object IDs to topology-based locators. They point to corresponding storage locations in the network, i.e., the NRS translates the object IDs into corresponding topology-based addresses. This approach has three conceptual routing phases: (1) routing the request message (i.e., REQUEST(ID)) to the responsible NRS node where the object ID is translated into one or multiple source addresses, (2) routing the request message to the source address(es), and (3) routing the data from the source(s) to the requester. All phases can potentially use different routing algorithms. For example, if the object name space is not aggregatable, a *name-based routing* method might be used for the first phase. The second and third phase might use, e.g., a common topology-based routing algorithm like IP if the NRS translates the object IDs into topology-based addresses. There are multiple alternatives to loosely or tightly integrate these three phases in an ICN architecture.

The second general approach, Name-based Routing (NbR), does not perform any kind of name resolution but directly routes the request message from the requester to one or multiple data sources in the network based on the requested object ID. The routing algorithm used for this approach heavily depends on the properties of the name space again. After the source has received the request message, the data is routed back to the requester, equaling the third phase in the NRS-based approach. This last phase is called the *data transport* phase. The data transport in ICNs can rely on one or multiple data sources.

2.1.5 Caching

To ensure efficient network utilisation and improve data availability, several ICN architectures make heavy use of data *caching*. There are two major caching approaches: *caching at the network edge* and *in-network caching*. Caching at the network edge includes, e.g., user nodes like in P2P networks. In-network caching describes the caching of data within the transport network, e.g., on the forwarding path in network routers, or in conjunction with the Name Resolution System.

2.1.6 Additional Services

Depending on the actual ICN approach, additional components that are external to today's Internet architecture might become part of an ICN network architecture. For example, persistent data *storage* might be closer integrated with the network architecture, including a closer integration with caching and name resolution. Likewise, information *search* might be closer integrated than in today's Internet architecture, especially when metadata about IOs is stored in the ICN.

2.2 Related Work and State of the Art

The information-centric approach of the future networking is being explored by a number of research projects such as DONA, CCN, PSIRP, Postcards from the Edge and Layered Naming Architecture (LNA). Even though most of the projects differ regarding their architecture specification they all focus on a identifier-location split and make use of paradigms such as publish-subscribe.

The **Data-Oriented Network Architecture (DONA)** [5] project involves a major redesign of the Internet's naming and name resolution infrastructure. DONA names are organised around owners. Each owner is associated with a public-private key pair, and each service or any other named entity (host, domain, etc.) is associated with its owner. Names are in the form $P : L$ where P is the cryptographic hash of the owner's public key and L is a label chosen by the owner, who ensures that these names are unique. The information objects in DONA are published into the network by the sources. Nodes that are authorised to serve data register to the resolution infrastructure. Once a given content is registered, requests can be routed to it. Register commands have a Time To Live (TTL), when the registration expires it needs to be renewed. Resolution Handlers have a hierarchical structure. Requests are routed by name in a hierarchical fashion. The resolution infrastructure routes requests by name and tries to find a copy of the content closest to the client. DONA's anycast name resolution process allows support for network-imposed middle boxes (e.g., firewall and proxies). In purely data-oriented operation, IOs are routed back along the same path of the request. In DONA, caching is inherent in the architecture meaning that any cache can respond to a request and serve the relative IO[2].

Content Centric Networking (CCN) [4] is a project, which proposes a change in the current network architecture by developing a new approach that enables content to migrate wherever it's needed. CCN introduces the concept on which the addresses used to establish communication refer to content rather than to location. The main idea here is that when a user asks for a specific IO, the request for the IO is routed towards the location in the network where that IO has been published. On the way towards the source the caches of the traversed nodes are checked for copies of the requested IO. As soon as an instance of IO is found it is returned to the requester along the path the request came from. All the nodes along that path cache a copy of that IO in case they get more requests for it. The CCN naming scheme is hierarchical in order to achieve better routing scalability through name prefix aggregation. The names are rooted in a prefix, unique for each publisher. The publisher prefix makes it possible for clients to construct valid names for data that does not yet exist, and publishers can respond with dynamically generated data. The granularity of the names is very fine: single chunks (packets) are named. As mentioned above, CCN routes the request for data towards the publisher, and makes use of any cached copies along that path. Copies can also be found by local search. It should be noted that in CCN atomic objects are single packets, so it is possible that only a part of a bigger object is cached.

The **Publish-Subscribe Internetworking Routing Paradigm (PSIRP)** project also proposed a redesign of the current Internet from host-centric approach, to a new information-centric approach. To achieve this goal, PSIRP moves away from the current sender-oriented Internet model towards a more receiver-oriented model, on which the publish-subscribe paradigm plays a central role. Here the IOs are published into the network by the sources. Receivers can then subscribe to IOs that have been published. The publications and subscriptions are then matched by a Rendezvous system and the matching procedure results in a Rendezvous ID (RID). Then the RIDs can be resolved (within a scope) to a forwarding identifier that can be used to forward data objects. PSIRP uses two primary name spaces, rendezvous identifiers and forwarding identifiers. RIDs (together with scope identifiers) name rendezvous points which are used to establish contact between publishers and subscribers. There is no name space for IOs per se, but the name of the rendezvous point can be used as one. In PSIRP, caching is limited to the scope of the rendezvous point for the identifier associated with an object. Within that scope an object can be cached in multiple caches.

A more detailed technical comparison between CCN, DONA, PSIRP and 4WARD NetInf concepts can be found in [2].

As stated in [6], the Future Internet Design (FIND) project **Postcards from the Edge** [7] develops cache-and-forward architecture based on the computation and storage capacities performed at every node, regardless of it being a core router in the backbone, an edge access point or even a mobile host. Even though the project focuses on optimising large file transfer as a separate service, it also acknowledges the need for the traditional best-effort delivery service for other kind of services such as Voice over IP (VoIP), video and audio streaming as well. To achieve this the project proposes a transport layer service that operates in a hop-by-hop store and forward manner. Despite a hop-by-hop transport of opportunistic delivery this FIND project addresses the naming issues, in order to provide location information for mobile terminals. Prototype implementation and experimental validation of key architectural innovations are major goals of this project.

The architecture in the **Layered Naming Architecture (LNA)** [8] project has three levels of name resolution. Such three levels of name resolution are, (1) from user-level descriptors to service identifiers, (2) from service identifiers to endpoint identifiers and (3) from endpoint identifiers to IP addresses. LNA introduces that services and data become the main objects rather than hosts, that mobility and multihoming can be handled in an easy way and that middle boxes (such as firewalls and Network Address Translations (NATs)) can be gracefully accommodated and no longer violate IP semantics.

The **4WARD** project [9] developed the 4WARD *Network of Information (NetInf)* architecture [10]. It can handle information at different abstraction levels making it possible to refer to information independently of its encoding. The naming scheme [3] provides both self-certification and name persistency. It has a flexible name resolution and routing framework for information in which multiple mechanisms can be plugged in, for example, the global hierarchical Multilevel DHT (MDHT) scheme. The service interface is inspired from publish/subscribe. Information is published, but subscribers ask for named content instead of receiving publication events.

> "SAIL emphasises deployment of the solutions in the medium term, which means that it must be possible to integrate the solutions into current technology"
>
> – *SAIL "Description of Work"*

SAIL NetInf is, to some extent, continuing where 4WARD NetInf finished. It is however re-evaluating some design decisions in 4WARD based on results and experience from other projects, mainly the ones mentioned above, and from the DTN community. 4WARD started as a clean-slate project, whereas SAIL emphasises deployment of the solutions in the medium term, meaning that the solutions are integrated into current technology. As an example of both re-evaluation and deployment emphasis, SAIL has simplified the 4WARD naming scheme and defined an URL-compatible syntax (see Section 4.1) to ease application development and migration. SAIL furthermore addresses several topics which 4WARD did not or only partly covered. Examples are content transport, interdomain issues, congestion control and cache management. SAIL is in general devoting more resources to protocol development and prototyping, leading to standardisation and deployment.

The **Content Delivery Networks (CDNs)** provide better scalability, reliance, and performance over a single web server solutions. A CDN is an infrastructure of network elements operating at layer(s) 4+ and is specialised for the efficient distribution and delivery of (digital) content. CDNs rely on content replication where the content is brought close to the users. A CDN relies on both content-delivery and request-routing infrastructures, where users' content requests are routed to the most appropriate edge servers that are then serving users. The request-routing infrastructure is updating its knowledge based on the content-delivery states, i.e., what content is stored in which CDN caches, to ensure the most efficient routing under any circumstances. Some CDNs for instance are using the Domain Name System (DNS) for load balancing purposes. Typically, CDNs also have accounting support in order to support content-specific billing. In a CDN, how the caches are populated is normally decoupled from the normal users' content-delivery and there can be many different technical solutions of doing that. For commercial CDNs, their customers typically need to have an agreement with their CDN provider. CDNs are used for different kinds of content varying from static content to dynamic content including traditional web content and streaming audio and video. The authors of [11] describe a taxonomy of different kinds of CDNs including both commercial and free CDN products. From an ICN point of view, there are significant differences compared to the CDNs like in the ICN environment, the content is decoupled from its location also infrastructure wise, i.e., a user does not necessarily publish the content via any specific infrastructure, thus the publication can be available via many different ways/networks/technologies. Secondly, NetInf IOs due to their self-certifying characteristic provide security even if they are transferred over non-trusted parties/infrastructures. In the CDN environment, a similar security level is achieved by trusting the CDN provider and the infrastructure. The Internet Engineering Task Force (IETF) is forming a new Content Delivery Network Interconnection (CDNI) Working Group (WG)[12], which will focus on studying how to interconnect different CDN infrastructures.

2.3 Research Motivation

In general, ICN is motivated by the fact that accessing named content (such as web and Peer-to-Peer (P2P) network resources) has become a dominant interaction model in today's Internet – without being necessarily well supported. The increasing demand for and deployment of CDN highlights a need for leveraging a certain concentration in content interest distribution by caching such content closer to the user. One of the objectives of the NetInf in SAIL is the research and development of novel networking technologies using proof-of-concept prototypes to lead the way from current networks to Information-Centric Networks.

CDNs can be characterised as an overlay solution, provided by a small number of players – for a limited number of applications (mostly Hypertext Transfer Protocol (HTTP)-based web access), and for a limited number of resources (typically depending on contracts between content and CDN providers). ICN is generalizing the principle of enhancing scalability of content distribution through replication and in-network caching and is making it available as a first order mechanism – for all applications and for all types of content.

There are different design options for ICN, and this document presents the specific NetInf design that is centered around a well-defined set of architecture invariants (such as unique naming, location-independence and a strict information-centric service model) and puts particular emphasis on supporting multi-technology/multi-domain interoperability.

In general, *Future Internet* research is often associated to so-called *clean-slate* approaches, i.e., radical approaches that are not constrained by backward compability requirements. The ICN concept can be seen as one product of such efforts. However, in order to have any realistic chance of deployment, it is important to advance the state of research from 'basic' research on concepts to practical development of systems that consider requirements from different domains: core technical requirements, deployment-oriented requirements such as migration paths from current technology, but also socio-economic requirements, including economic and legislatory requirements.

Still, ICN has hard unresolved research challenges, such as how to achieve scalability of name resolution and name-based routing systems in global ICNs. The SAIL NetInf work aims at addressing both: providing a platform for important research on ICN research topics as well as providing a technical solution that can be gradually deployed in today's infrastructure without requiring a 'flag-day' transition, providing support for different relevant communication scenarios and applications.

3 NetInf Architecture

The SAIL Network of Information is based on three main foundations: 1) the idea of unique naming of information objects without imposing a hierarchical naming structure (similar to the approaches developed by 4WARD and DONA); 2) receiver-oriented transport as in CCN; and 3) a multi-technology/multi-domain approach than can leverage different underlying network services and employ different name resolution/name-based routing and transport mechanisms.

The SAIL NetInf is leveraging the 4WARD results on naming and name resolution as elements for an overall Information-Centric Networking framework that can be applied to native all-ICN networks but can also be employed on-top of existing network. Particular focus is put on the possibility of inter-connecting different NetInf domains, each with their own object retrieval mechanisms and policies in order to better support network heterogeneity, accommodate different stake holders and to enable possibilities for co-existence with and migration from other, existing networks.

This support for heterogeneity requires a careful approach towards architecture specification – as one monolithic architecture is likely to be too rigid to enable a successful evolution and adoption of NetInf as a general Internet-scale technology. Therefore we have chosen to explicitly cater for heterogeneity and evolvability and developed a framework of architecture invariants that ensures general interoperability based on common architecture elements without compromising innovation, flexibility and applicability to a wide range of scenarios and networks.

Section 3.1 describes the set of architecture invariants that provide the necessary orientation for the more detailed architecture elements such as naming, transport, caching etc. Based on these invariants, an overview of a complete NetInf system is presented in Section 3.2 that illustrates the relation of the architecture elements in a big picture. The following sections Section 4.1 through Section 4.9 then explain these building blocks in more detail. The architecture building blocks are intended as instantiations for different architecture elements such as name resolution and routing, allowing for investigating (deploying) different alternatives in different contexts.

3.1 Architecture Invariants

In addition to the objectives for the NetInf architecture described above, NetInf should support the following types of applications: content distribution, interactive applications (interactive web services, person-to-person communication), and machine-to-machine communication. NetInf should also support communication over network topologies which are dynamic due to mobility events, as well as communication over delay-tolerant networks. When developing the NetInf architecture, the objectives and use cases for the NetInf architecture [13] have been analysed. This analysis has resulted in a number of high-level design decisions which are reflected in the key invariants of the NetInf architecture described below.

High-level View of the Architecture: The following statements describe the NetInf architecture at an abstract level.

- NetInf enables access of named objects, and defines a naming scheme for these objects. The NetInf naming scheme is designed to make objects accessible not only via NetInf protocols, but also via other ICN protocols.
- The NetInf layer performs routing and forwarding based on IO names.

- IO names are independent of the location of the object in the network topology.
- The NetInf layer forwards a message between end-points of the network. The message includes a source and a destination identifier from the NetInf name space. This is in analogy with the source and destination address in an IP packet.
- A multi-domain NetInf network may include domains with challenged networks, such as DTNs.

Name Resolution & Routing: NetInf may perform routing based on IO names. Routing based on flat IO names may not scale in a global NetInf network. Therefore, the global NetInf network may use a NRS to map IO names to locators of an underlying network, in order to take advantage of global routing and forwarding in this underlying network. An example of such an underlying network is the current Internet. Likewise, the NRS may provide indirections of IO names into other IO names known to the routing and forwarding plane, or to locators of an underlying network. Such IO names representing network entities are mapped in fast forwarding tables into next hop addresses. This allows for scalability, because the routing and forwarding plane has only to cope with the network topology, and not with the location of single IOs. The underlying network can be any interconnection of heterogeneous L2 or L3 subnetworks. There is (at least) one NRS for the global NetInf network (interdomain name resolution). There may be name resolution systems which are local to a domain or a host (intradomain name resolution).

Support for Challenged Networks: In some cases (e.g., in a challenged network domain) it is not possible to resolve IO names to locators of an underlying network at the source. However, the resolution may be performed by an intermediate NetInf node along the path to the destination (late binding). In this way, challenged network domains may act as NetInf domains with their own routing and forwarding strategies.

Forwarding: Forwarding is based on a stack of one or more identifiers from the NetInf name space. Each identifier is derived from the name of a network entity along the forwarding path. The identifier stack thereby describes a forwarding path across a sequence of network entities. The network entities may or may not be adjacent. Each NetInf message carries two identifier stacks, one for the destination (Destination ID Stack (DIS)), and one for the source (Source ID Stack (SIS)). Network nodes forward NetInf messages based on the DIS. They may also push/pop identifiers on/from the top of the message identifier stacks. The use of stacks reduces the need for forwarding state in the network. Compact Identifiers (CIDs) may be used in order to limit the message overhead and header size, and for efficient lookup and forward operation.

Mobility and Multihoming: In a global NetInf network, mobility and multihoming is based on dynamic updates of the bindings in the NRS between the IO names and the forwarding identifier stacks. Alternatively, mobility and multihoming may be based on dynamic updates of the NetInf layer routing information, so that the NetInf routing system announces the current location of IOs.

Transport: NetInf interconnects a variety of networks with different address spaces, different network technologies, and different owners. There is a Convergence Layer (CL) which adapts the NetInf layer to different types of underlying networks on a per hop basis. Examples of such underlying networks are TCP/IP networks, Ethernet, etc. The NetInf transport abstraction is chunk-based.

API: The NetInf API is IO-oriented as opposed to a classical channel-oriented and host-oriented API such as the socket API for TCP/IP. This means that in the NetInf API, IOs are addressed directly by their names.

3.2 Architecture Overview

Based on the general ICN concepts as introduced in Section 2.1 and the NetInf architecture invariants described in Section 3.1, an overview of NetInf's architecture elements is provided.

The chosen approach is not to specify a monolithic, comprehensive architecture but rather collect a set of important elements, from which specific networks can be instantiated.

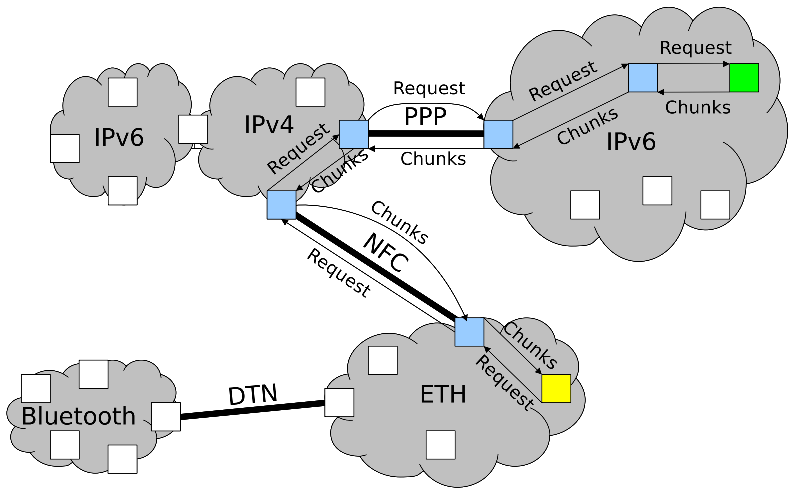

Figure 3.1: Example of NetInf deployment in heterogeneous networks

A global NetInf network can have different network domains with different specific ways of mappings, so a complete system can comprise different routing and name resolution approaches, as well as different underlying network technologies (link layers, network layers, transport protocols) as depicted in Figure 3.1. NetInf may be operated as an overlay over existing networks in some deployments but may eventually also be operated natively. The set of architecture elements is designed to support this diversity and interdomain operation.

For NetInf, two fundamental conceptual ways of interacting with the network can be distinguished: *publishing* IOs to the network (`PUBLISH(name,IO)`) and *requesting* IOs from the network (`REQUEST(name)`), as depicted in Figure 3.2. These messages are decoupled: due to the inherent in-network caching, the node that has published an IOs first does not necessarily have to be the one that receives and answers `REQUEST` messages.

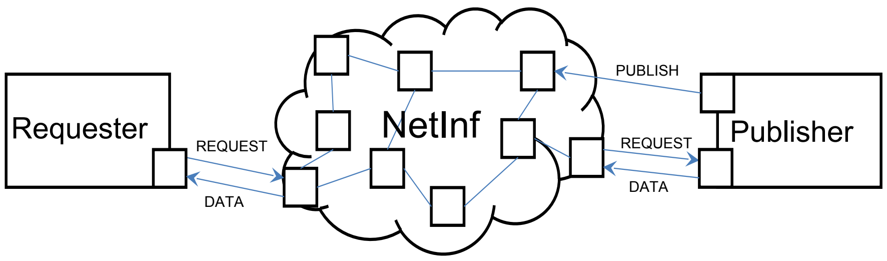

Figure 3.2: High-level NetInf architecture overview

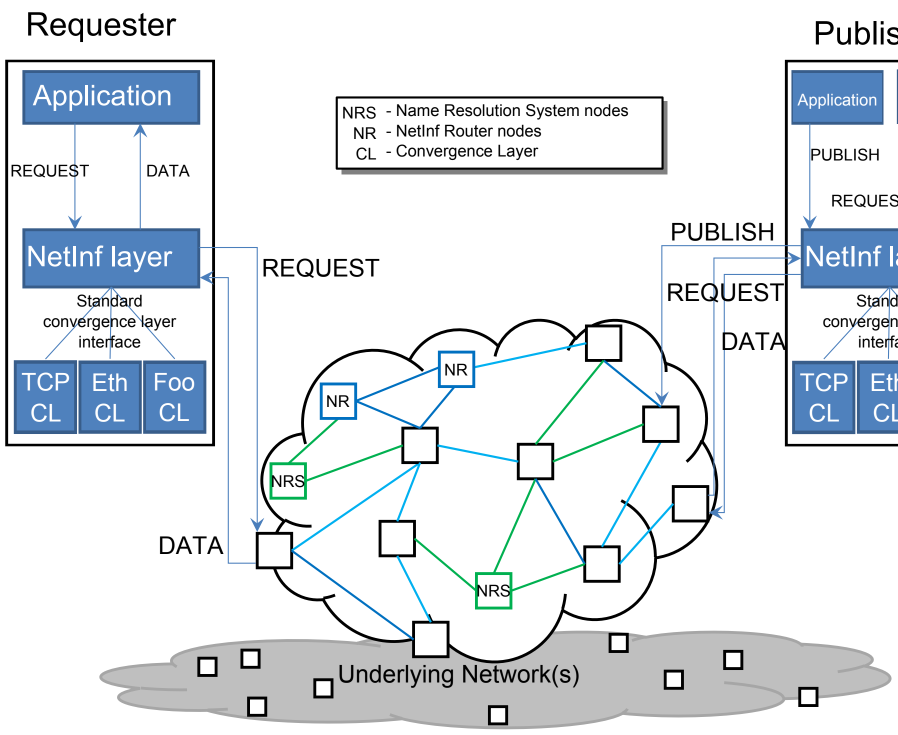

Figure 3.3: NetInf architecture overview

Figure 3.3 puts some of the mentioned architecture elements into context. The NetInf network is depicted as a network of name resolution system nodes and NetInf router nodes. There is an underlying network that can potentially provide packet-level routing and forwarding and that provides nodes that can be addressed by locators. The NRS can provide a resolution from NetInf names to such locators. The API and the conceptual node platform architecture is depicted by using two different node types – a *requester* and a *publisher*. The NetInf layer on those nodes provides a very simple interface to applications and it is utilising a corresponding interface provided by convergence layers, which map the NetInf protocol to specific underlying networks (this concept is explained in detail in Section 4.6).

It should be noted that in Figure 3.3's model the application instance of the requester does not 'talk' to the application instance of the publisher. Instead it is rather that the publisher application publishes an IO in the network (through registration in an NRS, through disseminating copies in a network environment, or through advertisement in a NbR system), and that the requester application requests this object from the network.

Naming In NetInf, there is no stringent relationship between the **IO name** and the location of the resource in the network. Names serve two purposes: 1) identifying objects, independent of their location in the network and 2) acting as keys to name resolution and routing mechanisms. Syntactically, NetInf names may provide some structure, but the network does not require names to carry network-topology nor organisation-structure related information – NetInf names are hence not hierarchically structured.

In the WWW architecture, Uniform Resource Identifiers (URIs) are typically used in a similar way (when considering cacheable resources), and application developers are already used to the idea of resolving different kinds of URI differently, and with pluggable application code. Hence it is decided to base the **naming format** for NetInf applications on the URI concept.

Such NetInf URIs may essentially be used in requests, or may be resolved (similar to how DNS

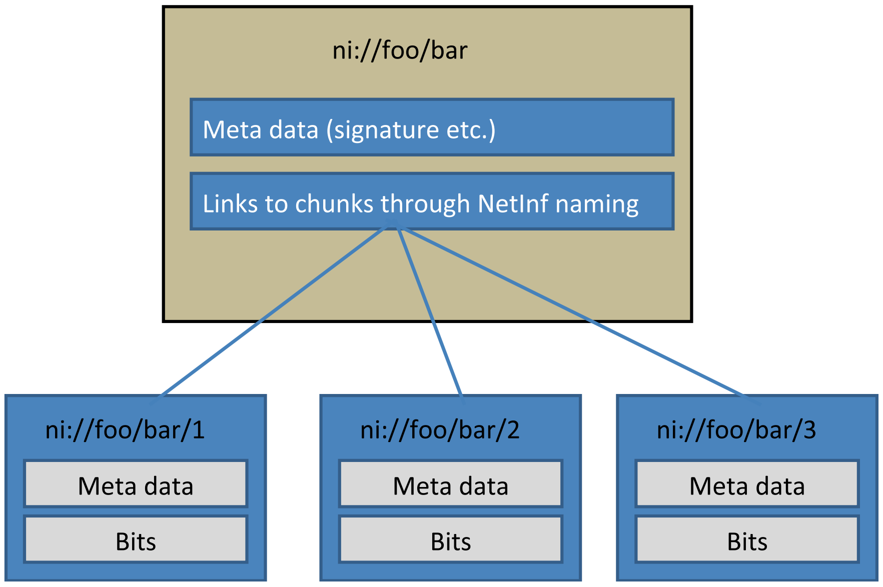

Figure 3.4: NetInf Object Model

names are resolved, or via other means) in order to carry out some operation on an IO. The kinds of operation may be similar to HTTP's GET and PUT methods. The details of NetInf's naming syntax and semantics are described in Section 4.1.

Object Model NetInf objects that are transferred in the network provide a certain structure that is conceptually depicted in Figure 3.4. In general, NetInf objects provide a name, meta information (e.g., for trust assertions) and, optionally, the actual bits representing the content and/or references to objects parts (*chunks*) or indirections such as locators (names in a different name space) for the object.

Those chunks generally provide the same structure, but may require different mechanisms for securing name-content binding and its verification by NetInf nodes.

The metadata that is conceptually part of the object may also be used without the actual object itself. For instance it can be part of the information that an NRS stores about an object to facilitate object name lookup and object selection.

NetInf Messages and Interfaces NetInf nodes employ a small set of Protocol Data Units (PDUs) to request and forward information objects. Fundamentally, node-to-node communication is based on a `REQUEST` message scheme that is parametrised with information qualifying the requested IO. This scheme also affects the design of APIs that are offered to applications. Both the PDUs and the API are described in the following.

A NetInf network can provide different elements. A **NetInf node** is an entity in the network that can receive `REQUEST` messages and that can assist in retrieving the corresponding object, for instance by answering the request from its cache, by forwarding the request to a next-hop NetInf node, by resolving the NetInf name to a lower layer locator etc.

In a multi-domain setting, i.e., a network of NetInf networks, there would be gateways between those networks that would also be NetInf nodes (with interfaces to two domains).

These nodes use a single PDU – the `REQUEST` message – to communicate, but there are different ways to specify the desired object, and there are different responses. The different variants are discussed in the following:

```
REQUEST(NetInfName, [Hint], [Qualifier]) => (NetInfIO | NetInfName | [Hint] )
```

The `NetInfName` parameter identifies the requested IO (see naming section). The optional `[Hint]` parameter is a list of location hints that may be used by the next-hop node to process the request. The optional `[Qualifier]` parameter is a list of request qualifiers that can be used to further specify the request, e.g., by providing selectors for certain elements of the requested IO. The request can either deliver the requested IO itself – it can also redirect to another object (specified by a NetInfName), or it can yield a list of Hints, e.g., lower-layer locators.

It should be noted that this conceptual description does not specify the actual messages (and their parameters) on the wire. Several optimisations are possible, for example, there could be one syntactic format to represent both `NetInfNames` and `Hints`.

This basic request scheme describes different NetInf networks with different functional components - that are all NetInf nodes according to the above description. For example, a **Name Resolution Service** node would be a NetInf Node that accepts requests with either `NetInfName`, `HumanReadableName`, or `SearchExpression` parameters and that returns a list of `Hints`.

A **NetInf Router** in a name-based routing scenario would be a NetInf Node that accepts requests with a `NetInfName` parameter and then decides how to forward the request so that it will eventually yield a `NetInfIO` as a result that can be delivered to the previous-hop NetInf Node.

A **NetInf Cache** is a NetInf Node that also accepts a `NetInfName` parameter and uses this as a key to look up the object in its object store. Of course, NetInf Cache functionality can be present in a NetInf Router, as well as in a Name Resolution Service node. In the latter case, the Name Resolution Service node would either return the IO (if it was found in the cache), or the list of `Hints` if does not have a cached copy.

This fundamental model is reflected by bf APIs that fundamentally shape the way that higher layer functions and applications can interact with the network. Enabling different instantiations of NetInf systems with specific (for instance) name resolution mechanisms and providing the necessary evolvability to future innovations however requires some flexibility – which are discussed in Section 4.9.

Object Replication and Caching Caching of IOs by NetInf Nodes is a key feature of NetInf. NetInf IO naming and name-to-content binding mechanisms enable object replication, which is the basis for caching.

There may be multiple copies of an information object, and in this case the NetInf network may be able to map the name of an information object to the locations of the different copies. NetInf may also support these mappings when an information object moves between different locations in the network topology, and when new copies are created, e.g., in caches.

Principally, we can distinguish between on-path caching and off-path caching. As described above, the general *NetInf Node* concept can be leveraged to describe different instantiations such as *Name Resolution Service* systems, *NetInf Routers*, and *NetInf Caches*. A *NetInf Router* with built-in caching functionality can provide **on-path caching**, i.e., it can cache objects by forwarding corresponding `REQUEST` messages and by caching the retrieved results when forwarding it to the previous-hop node. Subsequent requests for the same object could then be answered from the cache.

NetInf also supports another form of caching – **off-path caching**. When NetInf is run on a network with lower-layer locators, such as today's web, objects copies could be present at different locations in the network, e.g., in HTTP proxy caches, CDN caches etc. In such networks, routing and forwarding would be done below the NetInf layer, but still it would desirable to obtain a requested object from cache that is topologically located close to the requestor. *NetInf Caches* would thus register a cached copy in *Name Resolution Service* node, and when this node is later receiving `REQUEST` messages for that particular object, the *NetInf Cache's* locator will be returned in the list of `Hints`.

Finally, in networks with peer-to-peer communication interactions, such as DTN networks with opportunistic object dissemination, **peer-caching** could be employed, i.e., nodes generally provide storage functionality and are able to exchange information object on contacts.

It should be noted that the different caching variants can also be combined in a specific network, which enables NetInf to provide CDN-like administratively controlled pro-active caching (through off-path caching) as well as dynamic, on-demand caching in *NetInf Routers*.

Networks typically provide transmission of fragments with a maximum size limit (Maximum Transmission Units (MTUs)), and also, applications are often not always interested in complete IOs but only fragments – which is addressed by NetInf's chunk concept. Fundamentally, NetInf can provide on-path and off-path **caching** of chunks, and caches can be used to enable pro-active caching (i.e., distributing object copies to caches before they are actually requested) and re-active caching. NetInf provides cache management functions that can, for instance, leverage knowledge from name resolution services to enhance cache hit probability – which are described in Section 4.7.

Furthermore, maintenance and operation of caches are important topics for caching and network operation, e.g., for enabling operators to set certain performance and quality-of-service goals in advance that are implemented through NetInf network management. This requires a consistent management of network, caching and computing resources, which is also one of the topics of the work in the SAIL Network Management Theme that is described in [14].

Protocol Overview The fundamental protocol depicted in Figure 3.3 allows for the transmission of requests for IOs and corresponding responses across a network of NetInf nodes and across networks of different NetInf domains, with independent specific approaches towards name resolution and forwarding.

In general, NetInf supports name resolution based operation (when a NetInf node obtains lower-layer locator information for a given IO from an NRS and, in a second step, uses this information to obtain the object) and name-based routing approaches where a NetInf node sends a request to the network, and the network routes that request appropriately so that the requested object can be returned. These two approaches are also called *two-step* / *one-step* approach.

NetInf allows for combining these two approaches by employing a single protocol for requesting objects and/or requesting information about the location of objects. In other words, when requesting an object, a requestor must be prepared to receive either the object itself, an indirection to another object, or lower-layer locator information (which are conceptually also treated as indirection).

In the following, first the *protocol* is conceptually described, before the interactions with next-hop nodes and NRSs are explained in more detail.

The fundamental NetInf Protocol is based on a `REQUEST` messages that can have different qualifier types for a requested IO (NetInfName, HumanReadableName, SearchExpression, etc.).

Any node, including NRSs, must be prepared to receive and to reply to such requests. A NRS node would, in most cases, try to look up the provided information (e.g., the NetInfName in its NRS and return hints for the location.

A NetInf Router Node would accept the request, perform routing decisions on the name and forward the request. This forwarding may result in retrieving the actual object – or at least hints where the object can be obtained from.

Request and Response Forwarding NetInf request forwarding (e.g., for name-based routing) is stateless on NetInf nodes, i.e., NetInf nodes do not need to keep state for requests when receiving and forwarding them to be able to forward the corresponding response back towards the requestor.[1]

[1] Convergence layer implementations might require some state, depending on the provided services (e.g., reliable transmission and congestion control).

Fundamentally, NetInf does not provide the notion of node identifiers such as IP addresses. Named information objects are the only named entities in the network. When NetInf is operated over networks with named nodes, those identifiers may be used for path/gateway selection, route recording, etc., NetInf therefore supports the notion of *label stacks*, i.e., records of identifiers that can identify next hope nodes, networks (or gateways) or other object identifiers.

Such node identifiers would, for instance, be used to identify nodes on underlays and may not have global significance. Examples may be an DTN Endpoint Identifier (EID), an IPv4 address, and an IPv6 address that would be used to identify gateway nodes between NetInf domains.

When processing a `REQUEST` message, a node that is actually able to answer the request can use the aggregated information from the label stack and inject a response message to the network. The default operation would be to use the label stack to forward the response on a symmetric path back to the requestor, by sending it hop-by-hop to the top-most stack entry. Every node that receives a message would remove its own identifier from the stack and try to forward the request to the node identified by the next stack element.

For both request and response forwarding, nodes do not necessarily have to understand the identifiers on the stack – only the next-hop's identifier.

In NetInf, Labels (and locators) are names, fundamentally just like NetInf names: They are meaningful to some nodes to identify information. A resolution step can yield another name, which can be a locator (an identifier that is intended to be useful in some network or system context).

Consequently, there is no strong requirement about the nature of label stack elements. Elements should be useful for recording a path back to the requestor, but the label stack elements do not all have to be of the same type (e.g., identifiers on a certain layer).

When forwarding responses back to a requestor an intermediate NetInf node may take any stack element, perform further name resolution on it and, if successful send the response in the direction of the resolved names' node (with proper label stack modifications).

When a response arrives at a requestor node, the node must deliver (demultiplex) the message to the requestor application. Some sort of demultiplexing identifier is needed for this purpose, similar to a TCP/IP port number. The name space for this identifier is for further study.

NetInf Transport Services NetInf's support for technology heterogeneity requires a transport abstraction that can provide connectivity as well as transport services such as a reliable transmission over multiple different underlying network technologies and across multiple network domains.

NetInf can employ **convergence layers** as transport abstractions that provide domain- and underlay-specific mechanisms to implement the NetInf protocol. In this model, a locator is essentially an alternative name for an object that can be used to request the object in a network that supports the specific locator syntax and associated protocols.

The different locators represent identifiers in different protocol domains. E.g., `http://n1.foocdn.com` represents an HTTP URI that can be used in an HTTP GET request, whereas `nitp://n2.foocdn.com:80` represent an identifier for another protocol.

Each of these protocols provides the fundamental NetInf `REQUEST` interaction – but every protocol has its own way of doing it. These NetInf request/response implementations for specific protocols are called "convergence layers" (in analogy to bundle sending/delivery service of the DTN Bundle Protocol Convergence Layers as specified by RFC 5050).

Convergence layers thus provide *chunk request and transport services* for a particular network link/domain. NetInf nodes implementing such convergence layers could be gateways between domains – or the equivalent of *performance enhancing proxies*, i.e., they would provide optimised hop-by-hop transport for a challenged network link. One research challenge in this context is to control the interaction between transport services provided hop-by-hop convergence layers (such as retransmission strategies) and transport services in the overall network.

Alternatively, NetInf can employ **end-to-end** transport protocols as in today's Internet which are based on the notion that a requestor requests chunks from identified nodes without intermediaries involved. Different models are possible, including the adoption of traditional transport protocols such as Transmission Control Protocol (TCP) but also the definition of new, receiver-oriented transport protocols.

Details for these different approaches are discussed in Section 4.6.

NetInf Protocol in the Interdomain NetInf can allow the composition of interdomain networks, i.e., networks that connect independent network domains, each with their own mechanisms, policies and underlying network layers. Each of such networks would support the general NetInf Protocol, but it is assumed that name-based routing would be largely local to a domain (where the routing protocol is used), and that an NRS resolution would also be limited to resolve to locators of a local domain.

Consequently, gateways have to bridge between those domains. Fundamentally the same label stack approach applies to interdomain communication: gateways are NetInf nodes and add their identifier (in the destination domain) to request messages.

A particular challenge for NetInf and its non-hierarchical naming approach is to make the name resolution and routing system *scale to a large number of nodes* (i.e., appropriate for global deployment). Another challenge is to provide good performance in terms of latency for setting up communication sessions and network utilisation. This is discussed in detail in Section 4.2 and Section 4.4.

This ability to inter-connect different network domains is also intended to support *inter-provider relationships*, which is one of SAIL's general research themes that is discussed in more detail in in [14].

Since NetInf is oblivious to object location from an application point of view, one could assume that **mobility** is non-issue. For example, "moving" an IO could be equivalent to (re)-registering in to a name resolution service or advertising it to the routing system (ignoring propagation and convergence latency for now). Similarly, a mobile requestor would simply issue new object retrieval requests at its new point of attachment. However, performance, timeliness and robustness objectives impose requirements on the network architecture that are discussed in Section 4.5.

One of NetInf's propositions is that information can be uniquely identified and thus easily be replicated and cached in the network so that requests can be answered from caches instead of origin servers, which has some consequences on how **transport** fundamentally works. A genuine characteristic of transport in NetInf (and some ICN approaches) is that transport interactions can be fundamentally *receiver-driven*: there is no single origin server that is responsible for sending object chunks at an appropriate pace to a receiver in a host-to-host connection. Instead a receiver is requesting object (chunks) from the network, potentially leveraging path and cache diversity. Consequently, achieving objectives such as congestion control, flow fairness and flow stability requires a new approach to transport protocols, which are described in Section 4.6.

Security With NetInf's independence on object network locations, well-established **security** practices such as host-to-host connection-based security cannot be applied, so NetInf requires specific approaches for some security objectives. In NetInf, rather than authenticating the peer from whom one receives an IO, the IO (bytes) returned are authenticated to ensure that those bytes are that were requested. This can, for example, be achieved via inclusion of cryptographic values in names.

Additional application layer security services will be required for NetInf applications – and this continues to require confidentiality and data-integrity and various forms of accountability in an ICN context. However, as with the binding of the name and the IO bytes, these security services will need to be embedded into the IO itself – NetInf will make much more use of IO-level security

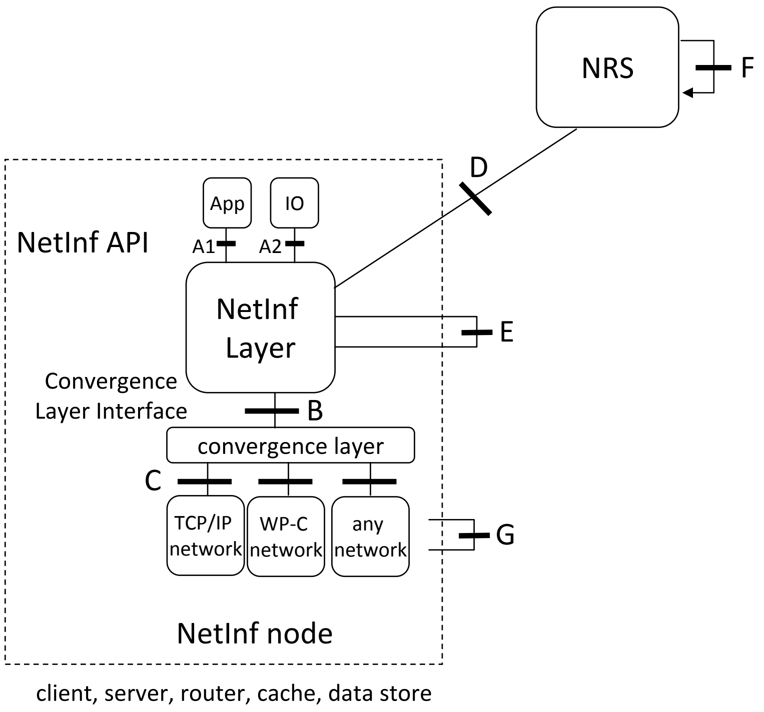

Figure 3.5: NetInf reference model

when compared to current applications (such a Session Initiation Protocol (SIP)-based VoIP signaling solutions) that can get-by with Transport Layer Security (TLS) for proxy-proxy connections. Security threats to NetInf design are described and corresponding potential security mechanisms are discussed in Section 4.8.

3.3 NetInf Reference Architecture

Figure 3.5 shows a more detailed view of the NetInf architecture and introduces a *reference model* with a set of *interfaces* and *reference points* that can be used to describe different elements in a NetInf system. This model exhibits all *possible* reference points and interfaces and can be used to describe different architecture elements and different instantiations of the NetInf architecture. A specific instantiation may use only a subset of the interfaces in the reference model. It does not mandate a particular way to implement or deploy a NetInf network in the prototyping activities [2].

The following interfaces are depicted:

- A: NetInf API. Different subsets A1 and A2 are used by applications and IOs respectively. The NetInf API is described in Section 4.9.
- B: Convergence Layer Interface. Offers NetInf-optimised transport services. Described in Section 4.6.
- C: Control and data interface between the Convergence Layer and underlying packet networks. Described in Section 4.6.
- D: Registration of networked objects (files, Radio Frequency ID (RFID) tags, application instances, etc.) and their metadata in NRS. Name resolution queries to NRS. Described in Section 4.2.

[2]The depicted reference architecture will be updated based on experience from prototyping.

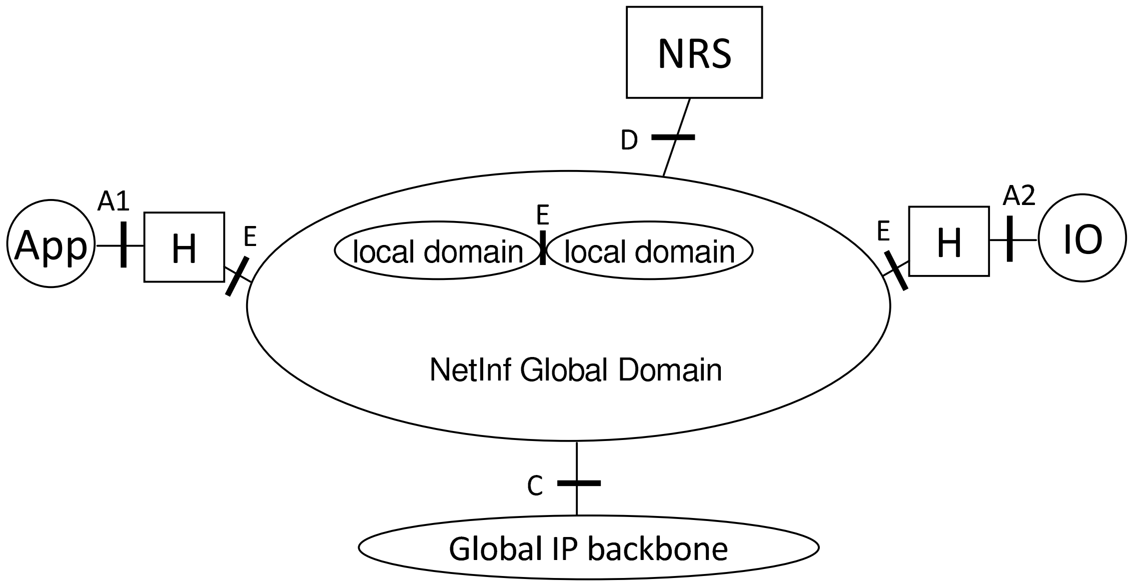

Figure 3.6: A network instantiation of the NetInf architecture

- E: NetInf data transfer and signalling messages, as well as NetInf network routing announcements. Described in Section 4.4 and Section 4.6.
- F: Inter-domain operation of NRS. Described in Section 4.2 and Section 4.3.
- G: Data transfer and routing announcements between underlying networks. Beyond the scope of the NetInf architecture.

The NetInf architecture as described by the reference model above can have different instantiations depending on the use case, requirements, performance optimisation, etc. Figure 3.6 shows an instantiation example of the NetInf architecture. Reference point *E* is used between local domains within the global domain, and also between hosts and network domains. Reference points *A1* and *A2* belong to the NetInf API. Reference point *C* is between the NetInf Convergence Layer and an underlying IP network.

3.4 Summary

The NetInf architecture that has been described in this chapter is based on the fundamental invariants that are introduced in Section 3.1, aiming at providing a framework for an information-centric networking system that can leverage existing network infrastructure but also migrate from an overlay approach to a global, native NetInf network. One of the guiding invariants is the desire to make as little assumptions of underlying (and neighbouring) networks as possible, allowing for domain-specific instantiations of NetInf's architecture invariants and for interworking between those domains.

Naturally this approach requires some flexibility and evolution possibilities. Similar as the Internet today, NetInf is expected to leverage different specific approaches for routing, name resolution, transport protocols etc., which prohibits any fixed system architecture. The chosen approach is to base the system on a overall model with fundamental invariants, such as naming, object model, general network protocol/interface abstractions etc., and to allow research and experiments for different specific approaches.

This research work requires implementation (running code) and experimentation, which can be seen as important activities to validate, but also improve the architecture invariants. Therefore the NetInf invariants can be seen as current guiding elements that will be refined based on prototyping experience. In Chapter 4, we present more specific architecture elements. Some of these elements comprise different specific approaches (e.g., for routing) – as instantiations of the overall architecture.

4 NetInf Architecture Elements

This chapter presents specific architecture elements as instantiations of the overall architecture described in Chapter 3. For some elements, different specific approaches (e.g., for routing) are presented, which does not imply that all variants will be required or will be used together in one network. Such variants are rather intended to illustrate different instantiation options which are under current investigation and might (for example) be used in different network domains.

Section 4.9 presents the NetInf API that applications would use to interact with the network. Section 4.1 provides details on the NetInf naming concept and format. Section 4.2 describes Name Resolution System approaches for NetInf, Section 4.3 presents how information objects can be searched for based on such approaches. Section 4.4 discusses routing in NetInf, and Section 4.5 shows how mobility can be supported in such networks. Section 4.6 describes NetInf transport services in more detail, and Section 4.7 presents corresponding caching strategies. Finally, Section 4.8 discusses security requirements and mechanisms for NetInf.

4.1 Naming

Naming of information plays a major role in NetInf. NetInf does not demand a hierarchical name space, which has consequences for the other architectural elements like NbR and name resolution as discussed in detail in the corresponding sections.

The NetInf naming concept strongly integrates with the NetInf security concepts. Specifically, the combination of names and related metadata includes information to support *data integrity*, *owner continuity*, and *owner identification*. Details about these security mechanisms are described in Section 4.8 and in [3]. The following section discusses a URI-based naming scheme for naming IOs. A URI-based NetInf naming scheme is used to support and foster application development and simplify migration of NetInf. This scheme has been published as Internet draft in [15].

The naming scheme is meant to be general so that it can serve as basis for several architectures focusing on information, such as P2P systems, caching system, and ICN architectures. More specifically, the scheme also builds a basis for a future NetInf-specific naming scheme that integrates the NetInf-specific naming features as discussed in Section 4.8.

4.1.1 URIs for Named Information

URIs [16] are used in various protocols for identifying resources. In many deployments those URIs contain strings that are hash function outputs in order to ensure uniqueness in terms of mapping the URI to a specific resource, or to make URIs hard to guess for security reasons. However, there is no standard way to interpret those strings and so today in general only the creator of the URI knows how to use the hash function output.

In the context of information-centric networking, there is value in being able to compare a presented resource against the URI that was de-referenced in order to access that resource. If a cryptographically-strong comparison function can be used then this allows for many forms of in-network storage, without requiring as much trust in the infrastructure used to present the resource. The outputs of hash functions can be used in this manner, if presented in a standard way. There are also many other potential uses for these hash outputs, for example, in terms of binding the URI to an owner via signatures and public keys, mapping between names, handling versioning, etc.

Many such uses can be based on "wrapping" the object with metadata, e.g., including signatures, public key certificates, etc.

Therefore the "ni" URI scheme that allows for, but does not insist upon, checking of the integrity of the URI/resource mapping is defined.

Hash-function outputs however are not human memorable, and cannot easily be used to construct a hierarchical name space, which some protocols and applications may require. The URI scheme therefore also allows for human-readable strings to be used with, or instead of, the hash function output strings.

It is expected to be beneficial for applications to be able to map between human-readable URIs and URIs that allow for validation of integrity the URI/resource mapping. However, in order to keep the proposed scheme simple and more broadly applicable, all considerations for how to map between URIs and for how to access resources using these URIs are to be specified elsewhere. Here, a form of URI that can be used hopefully in many different contexts is defined.

The URI scheme defined here could be thought of as being similar to URLs, but with the ability to verify the URL/resource mapping. However, these URIs can actually be envisaged to be used in applications where the resource is not located at a particular place in a network topology, but can rather be cached in many places.

Syntax definitions here are specified according to Augmented Backus-Naur Form (ABNF) [17].

4.1.2 Hash Strings

How the outputs from hash functions are handled is specified next.

Hash outputs are binary values that MUST be base64 encoded with line feeds, spaces, and terminating "=" characters removed. These values MUST be immediately preceded with a hash algorithm identifier and a separator character (":"). For example, the start of such a value might look like: "sha256:NDc0NzgyMGVmOGQ3OGU0..."

Hash values MAY be followed by a function identifier, naming the function to be used to verify that hash. If the function identifier is omitted, then the application needs to know how to verify the URI/resource mapping if that is desired.

In many cases the input to the hash function will be the actual resource itself as presented by whatever protocol uses the name and this is the default when the function identifier is omitted. This is what would be in the body of an HTTP response, were the URI used in an HTTP GET message and were the object returned in a 200 OK HTTP response with no fragmentation.

The function identifier allows for cases where the resource is actually presented with additional information (e.g., metadata) or is wrapped in other encoding. One way in which this is expected to be used is when the resource is presented with an accompanying digital signature. In that case the signature could be presented along with the resource and the hash function could be calculated over some combination of the resource and signature, or, just over the signature bits. (Note that the signature bits themselves are not part of the name in this example.)

Since the hash value needs to be verified against the resource, and since sometimes this will involve the resource being wrapped in some other format that allows inclusion of metadata or security data, it may be the case that the protocol that presents the resource identifies it as having the "wrapped" type. In order to support applications that require typing for the resource itself (as opposed to its "wrapped" form) a hash value is allowed to be accompanied with an "inner" type that identifies the type of the resource, rather than the wrapper. This is done using Multipurpose Internet Mail Extension (MIME) types appended after the function identifier.

Note that the "/" character from the MIME type MUST be percent encoded in order to conform to the ABNF below. That is, "application/jpeg" will be presented as "application%2fjpeg".

The "default" function-identifier, which is the only one defined here, is denoted with the string "id" and means that the resource when returned can be directly fed into the hash function without any canonicalization required, so this is the “identity” function. Of course, the hash based comparison may fail if some middle box or access protocol has re-encoded the resource in some way.

The “id” function identifier can be used if an “inner” MIME type should be added to the name.

Hash algorithm identifiers and function identifiers are to be registered, in an Internet Assigned Numbers Authority (IANA) registry.

A value encoded as above is called a “hash-string” and it could be defined as follows:

- hash-string = hashalg “:” b64value [“:” function-identifier [“:” mime-type]]
- hashalg = identifier
- function-identifier = identifier
- identifier = ALPHA *(ALPHA / DIGIT / “+” / “-” / “.”)
- mime-type = type %2f subtype

b64value is a string based on the 64-character subset of US-ASCII as defined in [17]. mime-type is based on the Content-Type header field syntax as specified in [18], but using %2f as a delimiter between type and subtype instead of “/”, and without parameters.

It is important to note that implementations are NOT REQUIRED to support any cryptographic operations, that is, as necessary, they need to be able to parse, route, log, and resolve names with any of the above fields, but do not have to perform any cryptographic operations.

Implementations that do support cryptographic operations MUST offer applications a way (e.g., via an API) to compare a ni name with a resource. The set of cryptographic operations to be supported (e.g., the set of supported function identifiers), is an implementation decision and is not further specified. Where an implementation does not support the operation needed to verify a ni object, it MUST return an error distinct from the case where the name-to-object comparison failed, e.g., due to a hash mismatch.

Implementations that create names such as these MUST ensure that it is possible to validate the mapping from the name to the resource, should other implementations choose to do that validation.

Note that not all protocols and applications making use of this URI form will require strong integrity assurances when doing name/resource comparisons. For this reason, it is expected to be relatively common to use truncated hashes in URIs.

4.1.3 URI Scheme

URIs consist of the scheme, an optional authority part and then a “local” part which is a possibly empty sequence of either hash-strings or any other string whose encoding is allowed. As with the local part, the authority part may be either a hash-string or any other string whose encoding is allowed.

The semantics of the authority part are not further defined here, but MUST be specified by any protocol or application making use of these URIs.

The “local” part is intended to contain an identifier for the resource in question that is meaningful in the context of the authority.

Note that where the authority part is omitted and where the local part is not a hash-string, there may be a significant probability for accidental name collisions. Protocols and applications using this URI scheme MUST take care of such collisions, if they matter. Note that this is also true, even if the authority part is present, unless there is some strict authority-part registration scheme in force and where spoofing is hard.

The combination of *authority* and *local-part* is called *hier-part* (i.e., hierarchical part) subsequently. This means that the defined URI scheme *can* support hierarchical names. However, it is important to note that NetInf does not *require* names to be hierarchical. The *hier-part*, nevertheless, allows to also support hierarchical naming schemes with the same URI scheme, which hopefully enables a common URI scheme for different ICN architectures.

One obvious thing to use for the authority part is a Fully Qualified Domain Name (FQDN), possibly with a port number, in which case applications using these URIs could make use of the DNS and TCP. Again though - such uses are outside the scope of this document. In general, there will be no guarantee that the resource can be accessed at that host:port even in that case.

- ni-name = scheme ":" hier-part
- hier-part = "//" [authority] *("/" local-part) ["/"]
- scheme = "ni"
- authority = hash-string — other-string ;(delimiters %-encoded)
- local-part = hash-string — other-string ;(delimiters %-encoded)

4.1.4 Examples

The longer examples in this section flow over lines, but the meaning should be clear enough.

1) ni://tcd.ie/cs8053-exam-2012

Example 1 is quite like a HTTP URL, and simply shows that "normal" URI forms can be used with ni names.

2) ni:///weather-in-dublin-today

Example 2 shows an example of an "intentional" name, where the resource returned will likely change from time to time. This example has no authority part, which presumably would mean that the requester doesn't really care much about the source of the weather information.

3) ni://tcd.ie/sha256:NDVmZTMzOGVkY2JjZGQ0ZmNmZGFlODQ5MjkyZ
DM0ZTg2ZDI5YzllMmU5OTFlNmE2Mjc3ZTFhN2JhNmE4ZjVmMwo

4) ni:///sha256:NDVmZTMzOGVkY2JjZGQ0ZmNmZGFlODQ5MjkyZDM0ZTg
2ZDI5YzllMmU5OTFlNmE2Mjc3ZTFhN2JhNmE4ZjVmMwo

Examples 3 and 4 are the same, one with, and one without, an authority part. In both cases, if only the ni name is known, then there is no knowledge of what was hashed. Some higher layer protocol may of course have this knowledge.

It may be that the authority part of example 3 allows for more scalable name-based routing of a request to get, or do something with, that resource.

5) ni://sha256:NDc0NzgyMGVmOGQ3OGU0MmI2MWYwZjY3MDAzNDJmZTY
0NzhhMGY0OTBhMDRiNzA0YTY0MWY0MzVkODQzZWUxMAo:id:sshpk/thing

The authority for example 5 is a hash-string of a public key as stored in a file by openssh and use of this hash-string is optional and case sensitive. For instance, some protocol(s) making use of this name, might expect that the resource contain a signature verifiable with a public key that matches that hash.

6) ni://tcd.ie/sha256:NDVmZTMzOGVkY2JjZGQ0ZmNmZGFlODQ5MjkyZDM0ZTg2ZDI5Yz llMmU5OTFlNmE2Mjc3ZTFhN2JhNmE4ZjVmMwo:signeddata:application%2Fjpeg

Example 6 is an ni name for a jpeg file that contains a hash of the file contents where the image to be received is expected to be in a Cryptographic Message Syntax (CMS) SignedData wrapper.

Note that the function identifier signeddata could be defined to also accept a Pretty Good Privacy (PGP) or XMLDSIG or other wrapper - what's identified is a function, and not directly a format. The signeddata function is something that would have to be defined elsewhere. That is, another specification would need to be written for each such function.

4.1.5 Security Considerations

Network elements that do attempt to verify the mapping from the name to the resource are doing more work that those that don't, both in terms of CPU, (for the hash and function identifier calculations) and possibly also in terms of network access and/or storage, since they need the resource, and possibly metadata that might have to be separately requested. An example of the latter kind of metadata might be a public key certificate or CRL. This additional load could be leveraged in some kinds of Denial of Service (DoS) attack. Protocols that call for validation of the name/resource mapping SHOULD specify how to handle any such DoS that may be relevant.

4.2 Name Resolution

The Name Resolution function is a key element of NetInf, as explained in the architecture overview (Section 3.2). A NetInf Name Resolution System (NRS) can provide resolution services to NetInf clients and/or be part of a name-based routing system. In the following subsection, a specific NRS design, based on the Resolution EXchange (REX) subsystem of Multilevel DHT (MDHT) ([19]), is used to show how a resolution process, integrated with a name-based forwarding system, can be performed over a global network.

Addressing Information Objects in the Global Network

A Global Name Resolution System (NRS) for NetInf is a great challenge, since the system should be able to identify and locate all the data available in the global network, whether explicitly published in the NRS or not.

One possible approach for that is coupling the NRS based on Multilevel DHT (MDHT) and REX, described in [19], with the high performance, stateless, forwarding system based on ID stacks, described in Section 4.4.1, developed in the context of the Global Information Network (GIN) architecture [20]. The resulting Integrated Name Resolution and Routing System may provide NetInf with the ability of delivering messages to any named object in the global network.

The structure of the GIN object identifiers is used to do that and these identifiers are compact labels composed of an authority part plus a local-part [1], and the *Late Name Binding (LNB)* strategy of MDHT, that allows the resolution process to terminate directly into the target object.

In the following, the process of addressing IOs registered in the NRS of the global NetInf network, with the NRS built according to [19], is described.

- It is assumed that information objects are identified with short size structured identifiers, called Compact Identifiers (CIDs). Any CID is composed of two (short) labels, w/p, where w identifies

[1]N.B. The alignment of the CID naming scheme adopted in GIN with the NetInf naming scheme described in Section 4.1 is still under discussion

the authority of the information object and p, the local-part, identifies a specific object in the domain of its authority. CID structure is defined here in analogy with that of NetInf names (Section 4.1) but CIDs are not equivalent to ni names. CIDs can be obtained algorithmically or via a mapping system from application level names, including ni names.

- Each authority chooses a *primary NRS*, that is, a private or public NetInf domain which provides a NRS service. For instance, in Figure 4.1, the primary NRS for authority w is network Y.
- Each authority publishes the authority ID, e.g., w, into the REX. The REX system is an independent third party NRS, which provides the Global Resolution service in the MDHT Architecture. The authority ID is registered in the REX system and mapped into the ID of the network domain which contains its primary NRS. In this example, a binding $w \rightarrow Y$ is registered into the REX.
- Any NetInf network domain can subscribe a notification service with the REX, in order to be notified about addition/removal/update of (interesting) bindings into the REX system. The (interesting) REX bindings may then be downloaded and installed/cached in the local NRS of that NetInf domain. Transit providers which offer global delivery services need to install all REX entries in their local NRS. Regional providers need to install only REX entries of authorities whose primary NRS is within their customer/peer networks. In the example of Figure 4.1, transit network Z is notified by REX about the binding $w \rightarrow Y$, which is added to the local NRS.
- After the authority registration, objects belonging to that authority can be registered in the network NRSs. The master copy (i.e., the authoritative and original copy) of a new object w/p may be registered in the primary NRS, even when created and held in another network, for global reachability. Additional replicas (i.e., non authoritative copies) of the same object can be registered also in the NRS of the local network where they are available, for local reachability.
- When a NetInf client requests object w/p to its access network, the whole object ID is first searched in the local NRS. If a replica of the object exists in the local network and has been registered, the forwarding system described in Section 4.4.1 is able to deliver the message to the device hosting that object.
- If a match for w/p does not exist in the NRS of a given network, a search is done on the authority ID w. Note that the NRS can be designed so that the search for both w/p and w may be done on the same network node.
- If the authority ID w is not registered in the local NRS, the message should be sent to the upstream provider [2]. In Figure 4.1, this kind of default rule is called ID Default Indirection.
- If the binding $w \rightarrow Y$ is found in the network NRS, the message is sent to network Y, by means of the delivery system described in Section 4.4.1. In the NRS of network Y, primary for authority w, a binding certainly exists for at least one master copy of object w/p, if it exists and has been registered. The delivery system will then forward the message to the destination(s), that is to the server(s) hosting a copy of w/p.
- Within a network domain, MDHT can provide a NRS with full support of locality, i.e., anycast routing. In that case, the delivery system can forward the message to the closest copy (or copies) of the target object available in that domain.

In the following, the process of addressing IOs which are not explicitly registered in the NRS of the global NetInf network is described.

[2]Tier 1 networks with no default indirection will typically discard the message

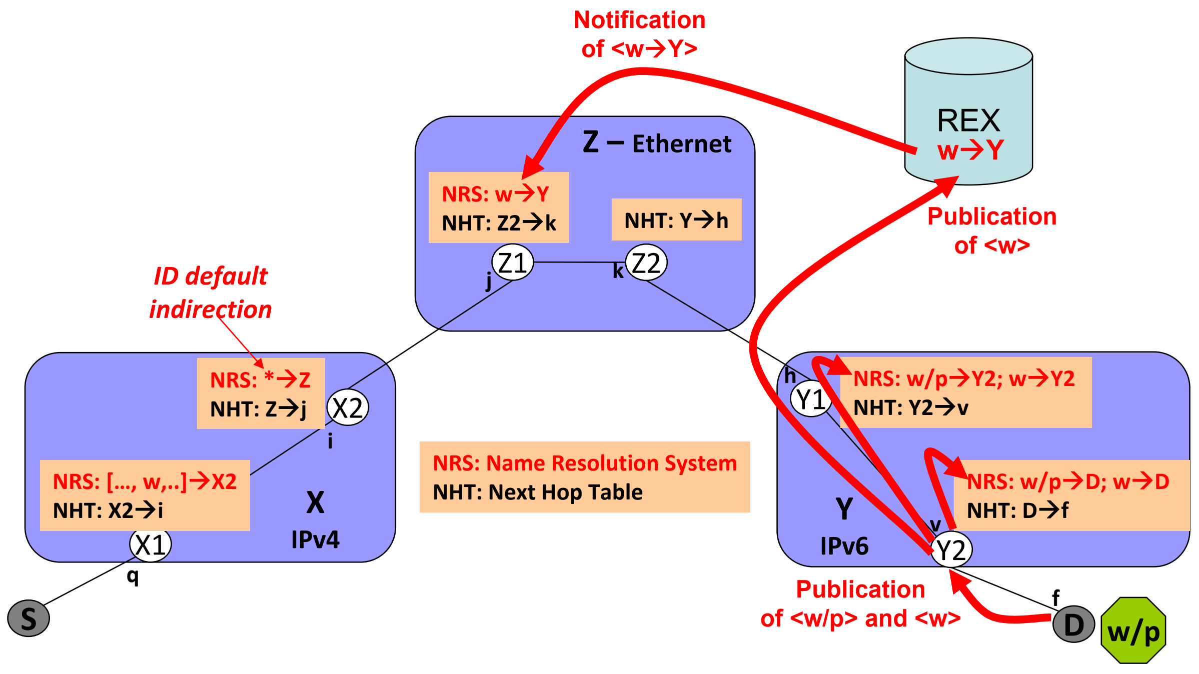

Figure 4.1: Publishing the authority ID (w) and the object ID (w/p) in MDHT for global resolution

- The binding $w \to Y$ is registered in the REX system and notified to the transit networks, as in the above case. The bindings $w \to Y2$ and $w \to D$ are also registered in the NRS of the authority's access network, as shown in Figure 4.1. This bindings allow the delivery system to forward any request for an object belonging to w to its server device D attached to the access node $Y2$. If the server of w is mirrored in more sites or multihomed, more bindings will be added to the NRS.

- Assume that a message is sent to the target w/p from an external network. The message cannot be resolved in that network, so it is sent, by default indirection, to an upstream provider. The upstream provider, notified by the REX, can resolve the authority ID w into the network ID Y. Then it can forward the message to that network by normal interdomain routing, as shown in Section 4.4.1.

- In network Y, it is not possible to resolve the target ID w/p. However, it is possible to resolve the authority ID w, which is mapped into the server ID D attached to node Y_2. The delivery system can then forward the message to the original server D, which has a master copy of the desired object or can redirect the message to another server.

- Note that the main server of authority w may be not the only source of object w/p. Cache nodes on the path between any requester and the authority servers can have a copy of the object to serve.

4.3 Search based on IO names and metadata

With the ubiquitous and pervasive presence of all types of content in ICN, another important component in the global NetInf architecture concerns information usability: how to provide an efficient, scalable and accurate ability for users to locate information objects matching users' interest from multiple sources or caching nodes in the network. The objective is to put receivers (or content providers) in touch with senders (or users/consumers) on good terms, to distribute, index, search and retrieve information in a large scale and highly dynamic networking environment. For receivers (or consumers), the problem consists of finding resources or content which closely reflect their interest, without knowledge if there exist senders that may have the queried content and where they may be located. This issue is all the more problematic since more and more information objects are continuously and massively disseminated in the network. At the opposite side, for senders that have objects to distribute, the main concern is how to efficiently advertise information into the network without knowledge of receivers that may be interested in these objects, so that the most users possible can receive the advertisement (in particular, users that are most likely to want those advertised objects).

4.3.1 Towards a Native Content Search Support

Resource discovery in a distributed network has always been a challenging issue, which consists in finding an efficient and scalable method of routing complex queries within the decentralised environment. Different solutions have been proposed in the literature, especially in P2P systems, and can be classified into two categories: unstructured and structured resource indexing and discovering. For example, the first category such as Gnutella uses blind search techniques to forward search queries to all neighbouring peers and could support a large range of queries including partial keyword search, but the bandwidth cost of searching grows exponentially with the number of connected users. This overhead issue makes unstructured searching as an unscalable distributed system, and inspired the development of more structured architectures based on Distributed Hash Tables (DHTs), which are more scalable and can generally support exact matching. Various proposals add a layer over a DHT structure to support partial matching search, but they often fail to support complex queries requiring an interpretation of resource semantic description.

The current Web is discovered by crawlers such as Yahoo!Slurp, Msnbot (Bing) or Googlebot. Web crawlers are bots designed to collect resources (web pages, images, videos, documents, etc.) from the Web for later indexing by a search engine. To scan for new resources, they proceed by recursively visiting the hyperlinks found from a set of seeding URLs. This approach is challenged by scalability and is a waste of energy (Google has to maintain one million servers for indexing the Web). However, a large volume of resources are beyond this recursive exploration, for example hyperlinks created on demand, dynamic or contextual pages are not found by a bot. This set of unexplored resources, known as the deep Web, is estimated as several orders of magnitude larger than the surface Web, which by contrast represents the part of the Web indexed by standard search engines. Since they are missing the deep Web, search engines are therefore searching only 0.03% of the pages available on Internet [21]. In parallel, it is worth noting that Google and Twitter need to handle more than 88 billion and 19 billion search queries per month respectively.

The Global Name Resolution System (see Section 4.2), as depicted in Figure 4.1, consists of different levels of indirections of NRS. This section proposes an extension of the Global Name Resolution System to provide an efficient and scalable content search within the NetInf layer, as a native network built-in process and not as an overlay or external application. The proposed approach consists in enabling content search directly within any NRS on a resolution path in the global network, so that the complexity of search is decentralised and shared out over the different levels of the Global Name Resolution System within the same processes for advertising and requesting content.

4.3.2 NRS-based Content Search

A search engine mines content available in the network and summarises the semantic properties of any discovered content into descriptors. When a content search is performed, the descriptors extracted from the query meta-info are compared to the descriptors of the search engine's index using a similarity measure. The search function returns then a list of information objects which match the query object. Search engines should be sublinearly scalable with growth in network size and efficient with a low query traffic volume and a high query success rate.

In the ICN context, information object metadata are used as descriptors to provide an overlay of semantic description of the information objects in the network, which then allows us to integrate content indexing and searching within the NRS by associating similar patterns between advertised metadata and search queries. In other words, there is a set of advertised descriptors per published IO, called advertised pattern, and from any search query, a set of query descriptors, called query pattern, is extracted. The set of advertised patterns are then distributed and indexed over the nodes which implement the DHT representing the network Dictionary of the NRS. Searching for any content consists thus in routing the search query to the most appropriate NRS nodes that are most likely to have the advertised patterns similar to the search query (based a similarity measure).

Each NRS node typically maintains a NRS lookup table in which an IO ID is mapped into one or several network IDs. To integrate content searching within the NRS, there is a need to extend this table to take into account IO metadata. Instead of using IO IDs as hash inputs to get the keys for DHT routing, it is proposed to use the advertised patterns. Content advertisements (i.e., publications) and search queries are then processed as follows:

1. Metadata are extracted from an advertised IO or a search query so as to semantically describe the IO or the search.

2. A pattern is created to store these metadata. As such, there is a pattern per published IO (called advertised pattern) and a pattern per search query (called query pattern). Note that the advertised pattern of a published IO is constructed with its name (i.e the IO ID) and its metadata. This allows us to retrieve information object by name as well as by content-based search.

3. Target NRS nodes, where to index content (i.e., to store advertised patterns) or where resolving a search query can occur, are identified by means of routing in the DHT.

4. Upon reception of a search query, the NRS node performs a lookup in its table by comparing the query search pattern against the advertised patterns (using a similarity measure) to find the set of most similar IOs as results for the search.

Search query resolution within the NRS aims at making the search query pattern converging to some advertised patterns by forwarding the query to the target NRS nodes which maintain the advertised patterns within a certain distance based on a similarity measure. As such, search queries for content in an information-centric network are actually equivalent to noise in a communication channel leading to errors during the transmission. By reusing error detection and correction techniques, the NRS can reconstruct the original data, i.e., find advertised patterns that match a search query. Routing a search query in the NRS consists then in finding a set of codewords such that the Hamming distance from the query pattern to each codeword is at most a similarity-based error. These codewords attempt to correct the query pattern in the error-correcting code and they actually correspond to the IOs that most closely match the search query.

4.3.3 Advertising and Searching for an IO in the Network

To publish an Information Object (Figure 4.2), a provider first computes the advertised pattern by using the IO identifier and its associated metadata. The provider then registers to its primary NRS via an advertisement message. The advertisement message contains the advertised pattern and additional useful information for the Name Resolution System: msg = <Object_ID, Metadata,

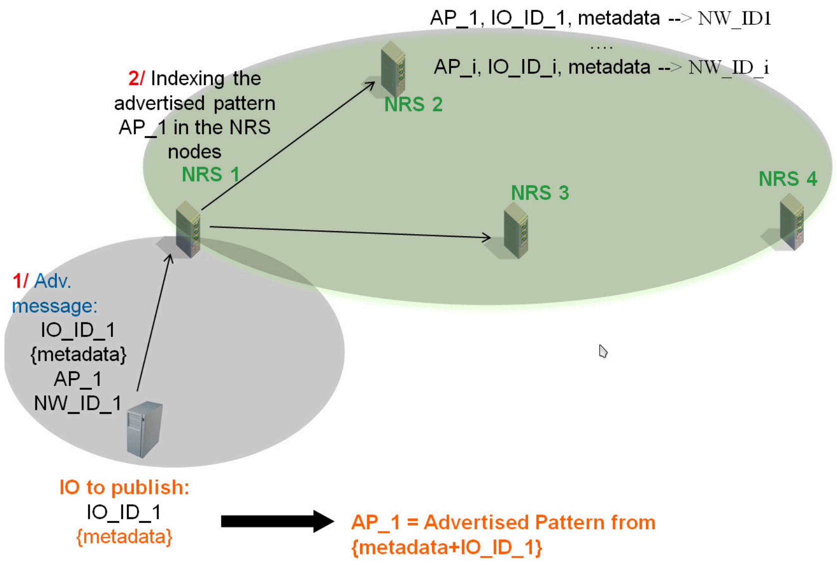
AP_1, IO_ID_1, metadata --> NW_ID1
....
AP_i, IO_ID_i, metadata --> NW_ID_i
2/ Indexing the advertised pattern AP_1 in the NRS nodes
NRS 1
NRS 2
NRS 3
NRS 4
1/ Adv. message: IO_ID_1 {metadata} AP_1 NW_ID_1
IO to publish: IO_ID_1 {metadata}
AP_1 = Advertised Pattern from {metadata+IO_ID_1}

Figure 4.2: Advertising and indexing an IO in the NRS

advertised pattern, NW_ID>. The primary NRS finally indexes the advertised IO at the appropriate NRS nodes.

Figure 4.3 describes the procedure for searching for content from the network. A universal message REQUEST(object ID) is needed to support content retrieval both by name and by querying a search. From meta-information encoding the search query, a query pattern which will be sent to the local NRS within the domain is computed. The query pattern is then routed to the target NRS nodes by using codewords-based routing. When receiving the query pattern, each target NRS node looks up in its NRS table to find the entry whose advertised pattern is similar to the query pattern: IO identifiers and associated metadata from the matching entries are then sent to the client that has performed the search.

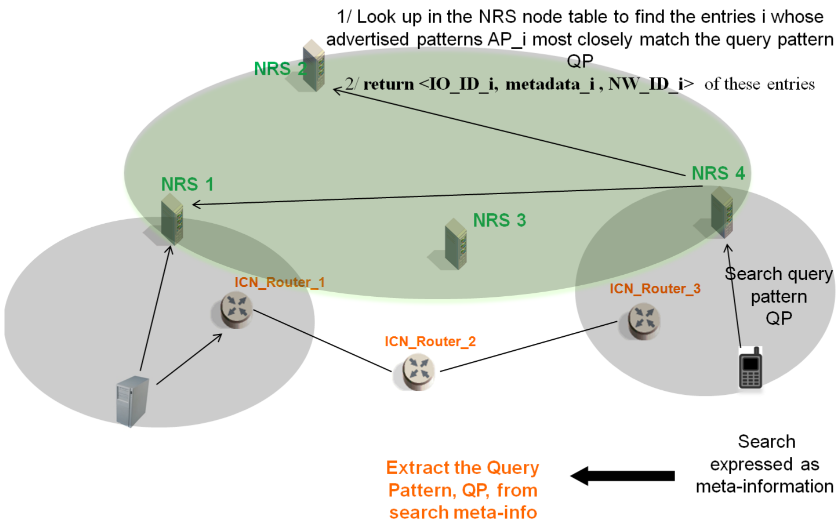
1/ Look up in the NRS node table to find the entries i whose advertised patterns AP_i most closely match the query pattern QP
2/ return <IO_ID_i, metadata_i , NW_ID_i> of these entries
NRS 2
NRS 1
NRS 3
NRS 4
ICN_Router_1
ICN_Router_2
ICN_Router_3
Search query pattern QP
Search expressed as meta-information
Extract the Query Pattern, QP, from search meta-info

Figure 4.3: Searching for content from the NRS

4.4 Routing and Forwarding

In NetInf, messages are routed between IO sources and requesting clients based on IO names. Routing by IO names may involve the Name Resolution function, which provides mappings between IO names and IO location information.

Alternatively, routing can be done without the support of an NRS. The NetInf routing system then announces the location of an IO via a routing protocol, similarly to the announcement of IP prefixes via an IP routing protocol. NetInf routers use the forwarding state set up by means of these routing announcements to forward request messages to an IO. For such a scheme to scale to the size of a global network, either compact routing schemes with routing stretch higher than 1 are needed, or IO names should be aggregatable based on the network topology. In the latter case, however, IO names would have location-dependent prefixes like in CCN. Clearly, this is in contrast with the NetInf assumption of location-independent names.

Routing scalability is less of an issue in smaller networks, where routing information for IO names can be announced without aggregation. Likewise, scalability of a broadcast approach is less of an issue in smaller networks. For example, a request for an IO can be broadcasted to all entities storing IOs without need for any routing state based on IO names.

The key motivation for using NRS as a component in the routing process is that it allows for leveraging the routing system of an underlying network, such as an IP network. Less routing state is then needed in the NetInf network. The underlying network and its address space can be optimised for aggregation and scalability. By doing so, routing issues do not need to be considered in the IO name space. An NRS-based approach therefore allows for a strict separation between IO names and IO location.

Similarly to the TCP/IP routing architecture, the NetInf architecture does not specify one specific routing mechanism. To allow for development of the routing technology, as well as the need for local optimisation of routing mechanisms, the NetInf architecture is open for different routing mechanisms. However, the NetInf architecture defines invariants with regard to routing and forwarding based on NRS as described in Section 3.1. The two routing mechanisms described in Section 4.4.1 and Section 4.4.2 are based on these invariants. Also, a routing mechanism based on a rendezvous system is described in Section 4.4.3 PSIRP. The relation between this mechanism and the architecture invariants is for further study.

Routing without the use of an NRS, for example in a DTN, is for further study.

4.4.1 A Routing and Forwarding System Based on ID Stacks

This section describes a forwarding system, adopted in the GIN architecture [20], which uses stacks of IDs to address information objects in an ICN network. This kind of mechanism is illustrated here because it is a candidate for the NetInf forwarding system.

In GIN, packets have a Source ID Stack (SIS) and a Destination ID Stack (DIS) written in writable headers. The first packet between a requesting entity and the target destination is sent across a resolution path (i.e., a path through resolution nodes which ends in the end-system having a copy of the target object). As the first packet crosses the network, the Source and Destination ID stacks are dynamically built in the resolution nodes. After the first packet has reached its target, following packets between the two communicating entities can be delivered on a topological shortest data path.

The GIN forwarding system is able to deliver packets by name over heterogeneous L2 and L3 networks (e.g., Ethernet, MPLS, IPv4, IPv6).

4.4.1.1 Requirements for the GIN Forwarding System

In GIN information objects are routed by name. That means that the routing decision is taken by examining the name or an ID of the target IO in the header of a packet. The forwarding operation should however be very fast. Forwarding of IP packets in current state-of-the-art routers is performed in about 50 microsec. An ICN forwarding system should have comparable performance. Therefore, the forwarding process should be preferably stateless and based on a simple lookup operation in a forwarding table, like in a router switch. For that reason, it would be reasonable to use IDs instead of application names to identify the target destinations. ID means a label identifying an IO. An ID used to address an IO should be quite short to keep the overhead of the packet header small.

Another related requirement is that the forwarding tables in the nodes of an ICN global infrastructure should have a limited size. Current IP routers can easily manage forwarding table with several hundreds of thousands entries. On high-end routers, it is possible to handle more than one million routes. A higher number of entries in the forwarding table could be expensive in terms of both costs and routing performance (e.g., convergence speed). The forwarding system should also be able to scale as the network grows beyond the current boundaries. Therefore, the size of the forwarding tables should grow very slowly with respect to the size of the global network infrastructure. Besides, the amount of control traffic (i.e., routing updates) should also be manageable and the routing process should guarantee the stability of the network and a high speed of convergence in case of recovery from node/link failures.

In ICN, the number of addressable entities is very high, because any IO can be potentially addressed in the forwarding system. Actually, the GIN forwarding system addresses any IO in the global network, a practical unlimited number of entities, while keeping IO IDs independent from the network locations where they can be found. Note that matching both requirements of 1) small forwarding tables and 2) almost unlimited number of topological independent addressable entities, is very challenging.

Supporting the delivery of packets over heterogeneous underlying networks is also another important requirement of GIN and NetInf. In fact, a main goal is to design a new ICN forwarding system able to deliver packets to destination by name (i.e., ID), across different underlying networks, which can be L2 (e.g., Ethernet, MPLS) or L3 (e.g., IPv4, IPv6). Such name-based networking system enables communication between IOs which are attached on different and heterogeneous network segments and which cannot normally communicate without some kind of interworking function. Besides, the ability of supporting communication over any underlying technology could facilitate the deployment of the new ICN network and facilitate the interworking between IPv4 and IPv6.

The name-based routing and forwarding system should also exploit the locality principles: first of all, if a copy of an object is available in the neighbors of a requesting client, the forwarding system should be able to address that copy first (i.e., *content locality*), with an anycast routing behavior. Besides, the resolution process and the data path should be kept within the network area which contains both the destination and the source (i.e., *resolution locality* and *routing locality*).

4.4.1.2 System Overview

GIN packets are forwarded in a GIN node by a combined process of next hop forwarding and resolution operations, which obeys to the following rule:

> Try to forward a packet in the Node Next Hop Table (NHT) first. If it is not possible, resolve the target destination of the packet in the NRS of the Node.

The following two functions are combined in the same infrastructure GIN node:

- The Name Resolution function provides just *indirections*: $ID \rightarrow ID_1, ID_2, ...ID_n$, that is, it maps an IO identifier into a list of one or more IO identifiers. In particular, IDs are mapped into IDs of topology objects (nodes, networks, hosts). In the following, it is assumed that an ID is composed of two fields, the authority part, which identifies the principal of an information set, and the local part, which uniquely identifies an information object within the domain of a given authority.
- The routing and forwarding system maps, in the NHT, the IDs of a node/network/local hosts into the underlying next hop address, and provides the encapsulation protocol needed to transfer the packet to the next hop.

The NRS and NHT are then built with different mechanisms.

The NRS is built with a registration protocol. Clients register in the provider NRS the IDs of IOs located in their devices/networks. They also register the authority part of the ID in a independent third party system (i.e., REX [19]), so that each authority can be mapped into the network ID where all IOs of that authority are registered (i.e., the *primary NRS* for that authority). These authority bindings are distributed and cached in the NRSs of the transit networks. A possible NRS architecture which embeds the topology hierarchy in order to support locality features (i.e., anycast routing) and implements the REX concept is the MDHT system (please, see [19] and Section 4.2 for more details).

The NHT is built with a routing protocol. Traditional intradomain/interdomain routing protocols (i.e., protocols based on distance/path vector or link state routing technology) can be used to advertise IDs of infrastructure nodes and networks and calculate the next hop information to fill in the NHT. Such protocols can guarantee fast convergence under topology changes and high performance. The approach is like the one currently in use in Internet to build the IP forwarding tables. The only difference is that nodes and networks IDs are advertised in the routing protocols, instead of IP prefixes and addresses.

”ID stacks” are used to dynamically address any IO in the network. Each GIN packet contains a couple of ID stacks: the DIS addresses the target IO, while the SIS addresses the source of the packet.

The DIS in a packet header is initialised to a target IO ID. The packet is delivered to the first GIN infrastructure node, which tries to forward the packet based on the ID on top of the DIS.

If the top ID is that of the current node, that ID is popped from the stack, and the next ID in the stack is used for fast forwarding in the node NHT. If a match is not found in the NHT, the packet is sent to the node NRS , which tries to resolve the top ID in the local NRS repository (called Dictionary). This even is called *Resolution Fallback*. Then, the following events may occur:

- If the top ID is resolved (i.e., an indirection is found into a new ID), the new ID is pushed on top of the DIS. If more indirections are found, a field in the packet header can suggest the right forwarding strategy: for example, the ’best’ ID is chosen and pushed on top of the packet DIS, or the packet is replicated in multiple packets, one for each valid indirection.
- If the top ID cannot be resolved (i.e., no match exists in the Dictionary), the ID of the next candidate resolution node (or network) is pushed on top of the DIS. This can be performed, as an example, by the MDHT finger table.

In both cases, the packet is handed again to the NHT forwarding function for a new lookup.

In general, a packet is said to be sent through a *resolution path* if, in the forwarding process, one or more nodes in the path require an ID in the DIS of the packet to be resolved in some node NRS. The resolution path is a slow path because packets must be routed through resolution nodes, which may be not on the shortest path, and because the resolution process cannot be performed in hardware like the NHT forwarding. In fact, the Dictionary DB may have a relevant size (up to 10^9 bindings with a TB memory). Assuming SSD technology, each Dictionary lookup may take about 0.5 msec ([19]). On the contrary, a packet which requires only NHT lookup operations to be forwarded to its destination, is said to be delivered on the *fast data path*. In fact, it is assumed that the NHT can be implemented with the same CAM and DRAM technology with which IP forwarding tables are built in high-end routers. In that case, only one memory access, i.e., a few microseconds, is required for each NHT lookup.

An example of a resolution path through heterogeneous networks is shown in Figure 4.4. One packet is sent by client S to its (IPv4) access network X to retrieve an IO with $ID =' w/p'$, where w represents the authority of the object, and p is the local part which uniquely identifies the object in the domain of w. Access node $X1$ is unable to forward or resolve for w/p, so it sends the packet to the next resolution node $X2$ responsible for the label range $[w]$, which includes the label w/p (*ID range indirection*). In that node, again, it is not possible neither to forward, nor to resolve for w/p, so the packet is sent, to the (Ethernet) transit network Z via *default ID indirection*. Here, it can be assumed that REX bindings of IO authorities have been registered in the distributed NRS of the transit provider. So there is a binding for w into network Y, which is the primary NRS for w. By simple NHT forwarding, the packet is sent then to the (IPv6) network Y, where a binding for w/p exists which points first to the exit node $Y2$, then to the server device D. Note that all entries in the NHTs can be built by means of traditional intradomain/interdomain routing protocols. Note also that only local underlying addresses are used in the NHT, that is the NHT forwarding process does not require to know routes towards remote addresses. In the figure, the DIS of the packet is shown hop by hop along the resolution path, with the push() and pop() operations. Note that the depth of the destination ID stack is only 2 IDs until the target destination.

The SIS of a GIN packet is initialised with the ID of the source entity (e.g the ID of the source end system). The source delivers the packet to an infrastructure Access Node, which pushes its ID on top of the SIS. In theory, any node on the path can add its ID in the SIS. Besides, if the packet is sent to an external network, the exit gateway can add the ID of the access Autonomous System (known in the global NHT, e.g., by Border Gateway Protocol (BGP)) on top of the SIS. In practice, however, only the intermediate nodes/networks which require to be on the return path should add their ID on top of the SIS and not all intermediate nodes/networks. When the packet reaches the target destination, the SIS can be put in the DIS of a response packet to be sent back to the source. Note that, in general, only a few IDs are really needed in the SIS to identify the return path. In most of the cases, three IDs are enough to route GIN packets back to any source in the Internet:

- the Source ID (known in the NHT of the local access node as a directly connected route);
- the Access Node ID (known in the NHTs of the nodes in the access Autonomous System (AS), by means of an intradomain routing process);
- the ID of the Access AS (known in the NHTs of the nodes of transit ASs, by means of an interdomain routing process).

For the example in Figure 4.4, the SIS of the packet sent from client S may hold only three IDs, i.e., [X|X1|S]. Such a SIS describes a hierarchical path to node S, connected to access node $X1$ into network domain X. If no other intermediate node on the resolution path is put in the SIS, the return data path coincides with the shortest path. Besides, such data path is also a fast path, that is, data sent on this path can be forwarded with lookups only in the fast NHTs. Note that most of the traffic can be sent through the fast data path, after the first packet of a data session has reached its destination on the resolution path.

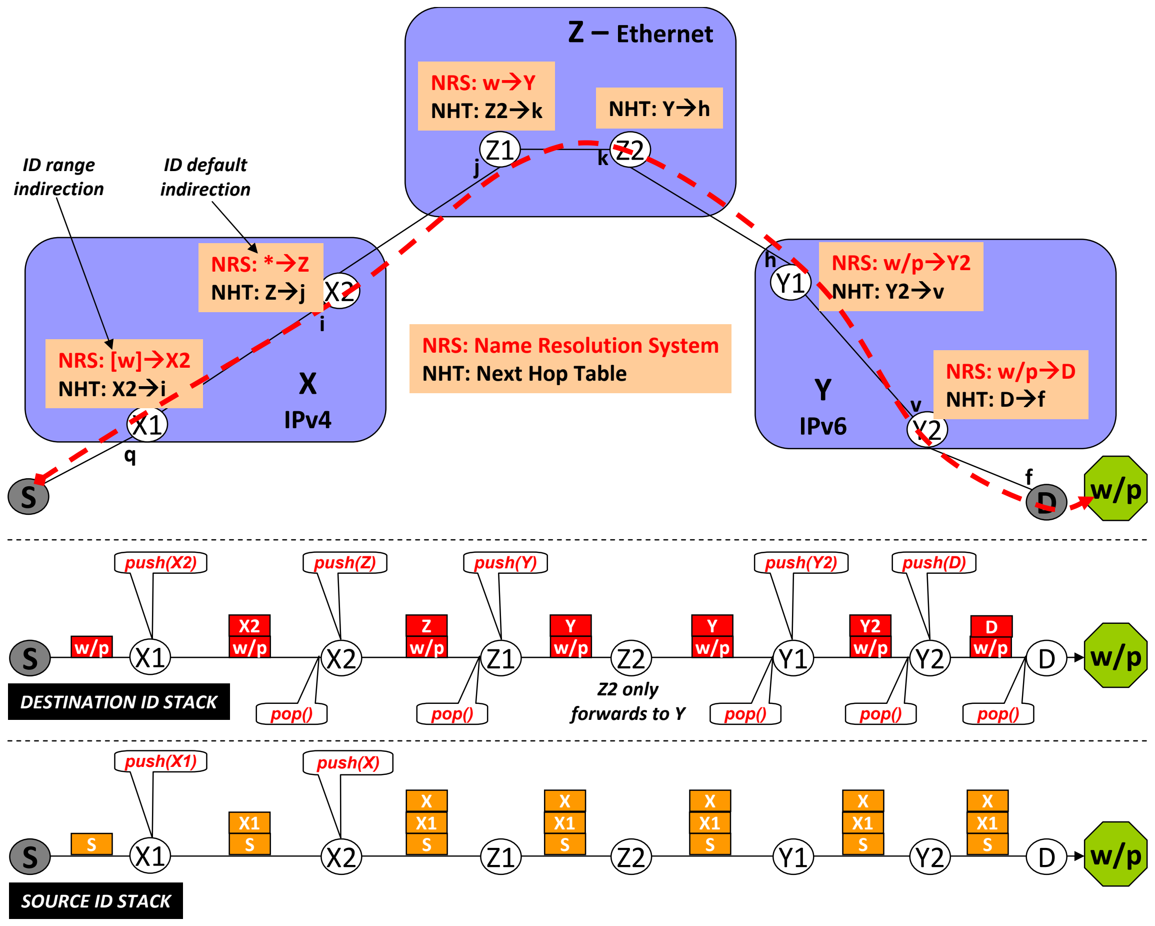

Figure 4.4: Routing a GIN packet with ID stacks over heterogeneous networks

The combined forwarding/resolution process causes a GIN packet to be routed by name to the target destination. An interesting aspect of this approach is that no universal addressing scheme is needed in the underlayer. Besides, network entities (infrastructure nodes, network domains, hosts) are labelled with topology independent IDs at the name networking layer. Packets can then be forwarded over a global network composed of heterogeneous L3 or L2 networks and most of the data traffic can be routed on an optimal shortest fast path. The system can be engineered to satisfy all the requirements described in the previous section.

4.4.2 Combined Routing and Name Resolution

This section outlines a NetInf routing mechanism that can be employed when there is a global Name Resolution System in combination with a global underlying network, such as a global IP backbone. The basic mechanism is based on Late Locator Construction (LLC) and is described in the 4WARD deliverables [9]. Here it is described how it can be adapted so that it adheres to the high-level NetInf architecture invariants described in Section 3.1. The reason for providing an outline of the LLC mechanism here is that it is a concrete example of an information-centric routing mechanism which has been evaluated by means of simulation, and which can be adapted to the NetInf architecture as described below. LLC has good scalability characteristics with regard to the amount of router state as a function of the number of networked IOs, and is able to handle mobility and multihoming of IOs as well as hosts and moving networks [9]. However, the LLC

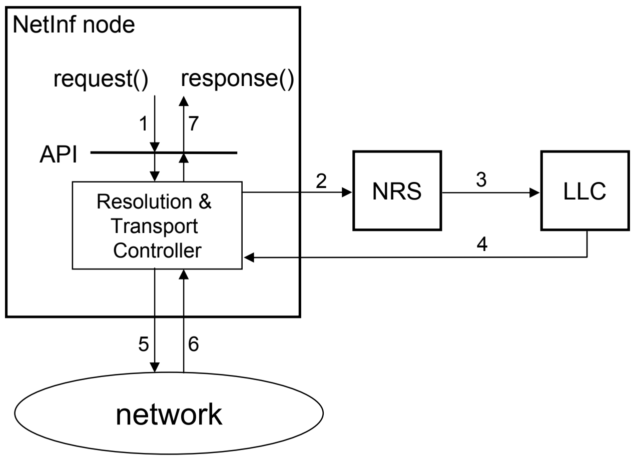

Figure 4.5: Building a label stack using LLC-based routing

routing mechanism is not applicable to Delay Tolerant Networks, where a Name Resolution System cannot always be reached, and where there is not necessarily a global backbone network.

Section 3.1 on NetInf architecture invariants describes a source and a destination label stack in the packet header, which is used for forwarding purposes. The entries in this stack represent a sequence of named network entities, such as network domains, gateways, hosts, etc. This sequence of network entities defines a forwarding path.

The LLC label stack does not use the push and pop option mentioned in the high-level description of the label stack in Section 3.1. The label stack thus does not change as a packet is forwarded between the source and the destination. As a consequence, the destination can return a packet to the source by reversing the order of the labels in the label stack. In principle, a push and pop mechanism for the label stack could be considered also for LLC. However, a new label stack for the reverse path would then have to be constructed by the LLC system. This construction of a separate label stack for the reverse path is also needed when asymmetric routing is desired.

Since a label stack defines a forwarding path, a routing mechanism is needed to build such a stack. The high-level NetInf architecture is open with regard to the routing mechanism, and thus does not prescribe a specific mechanism, or set of mechanisms. For example, the routing mechanism described in Section 4.4.1 fits in the NetInf architecture, as well as the LLC routing mechanism. However, even though the two mechanisms fit in the NetInf architecture, they do not necessarily interoperate. This is similar to the TCP/IP architecture, which does not prescribe a specific routing mechanism, but where the interoperation solution between different mechanisms is specific for each pair of mechanisms.

The LLC routing mechanism uses the Name Resolution System and the label stack of NetInf architecture. A NetInf node sends a request for the resolution of an IO name, and a label stack is returned describing a path to the IO. The combination of the NRS and the LLC performs this resolution, see Figure 4.5. When an application invokes the REQUEST() method of the NetInf API (see API description in Section 4.9) in order to retrieve a specific IO, the name of the IO is resolved using the NRS, see steps 1 and 2 in Figure 4.5. The NRS maps the IO name to an address of an IO-specific register in the LLC system (step 3 in the figure). This register is the entry point of a procedure to build a destination label for the IO based on topology information stored in the LLC system. In parallel, a source label stack for the requesting application is also built. The LLC system returns the source and destination label stacks to the NetInf node (step 4), which uses them when sending a REQUEST message to the network (step 5). The source and destination label stacks are also used when returning the requested IO (steps 6 and 7).

The LLC routing mechanism assumes a global backbone network. Attached to this network, there

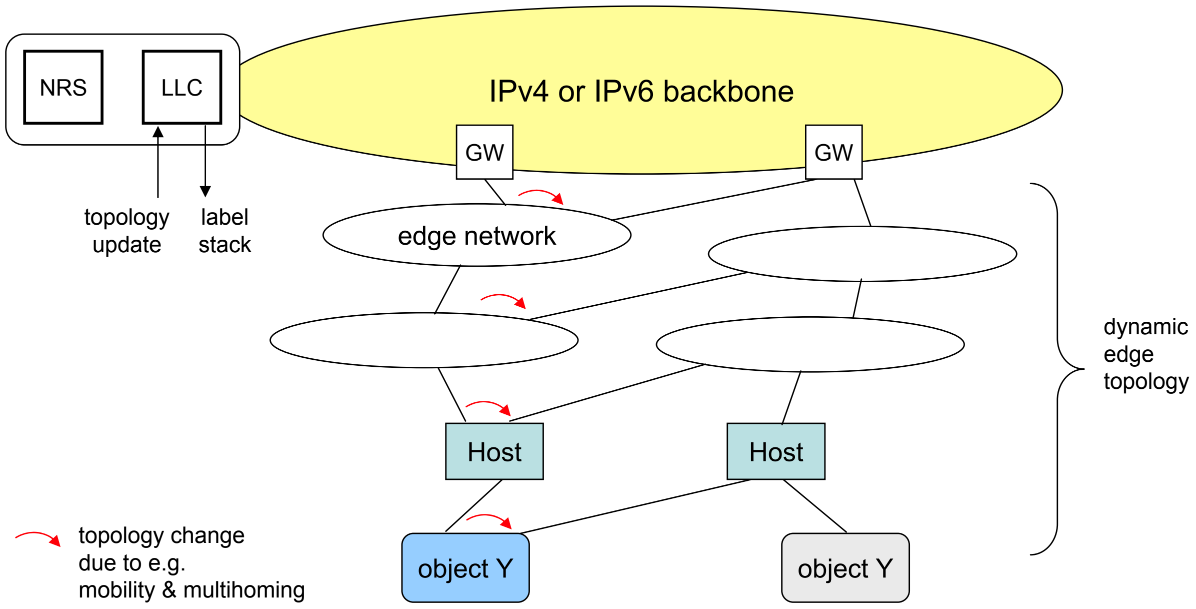

Figure 4.6: Routing across a dynamic edge topology using LLC

are a number of edge networks, see Figure 4.6. For example, the global backbone network may be the current Internet backbone, and the edge networks may be based on heterogeneous network technologies. The source label stack describes a forwarding path from the source IO across one or several edge networks to the backbone network. The destination label stack describes a path from the backbone network across one or several edge networks to the destination IO. The first entry in the destination label stack is the name of the edge gateway of the destination edge network. The name of this gateway is resolved into an address which is routable across the backbone network. The combined information in the source label stack, the destination label stack, and the routing system of the backbone network thus defines the end-to-end path. If the source and the destination are in the same edge network, a shortcut path can be defined, which does not traverse the edge gateway.

The LLC system is updated dynamically with topology and policy information for network entities and IOs, as well as with topology information for new copies of an IO. The forwarding path can thereby adapt to changes in the network topology, as well as to changes in the routing policy. This allows for handling of mobility and multihoming, as well as policy-based routing. Also, multiple copies of an IO can be registered in the LLC system. The LLC system may thus build several label stacks representing different copies of an IO. The LLC system or the requesting NetInf node can then select one of the label stacks based on some performance criteria. Such performance criteria are for further study.

The details of the LLC mechanism and a performance evaluation are reported in the 4WARD deliverables cited above. As mentioned earlier, the LLC routing mechanism does not support DTN. There is therefore a need for a complementary solution, where different routing protocols are used in the global network and in the DTN domains. This is for further study. Also, it should be noted that the NRS and the LLC system can be distributed in a hierarchical fashion similarly to the MDHT approach described in Section 4.4.1.

4.4.3 Publish/Subscribe Rendezvous Function

The rendezvous function is the control point of the publish/subscribe based ICN architecture in the PSIRP [22]. In its essence, the function is responsible of matching interests of a content provider (publisher) and a content consumer (subscriber). This subsection briefly describes main characteristics of the PSIRP rendezvous concept, which has been tested, evaluated and proven to be adequate in the global scope during the PSIRP project.

4.4.3.1 Rendezvous identifiers

In the rendezvous concept, information is identified with globally unique and flat crypto-identifier (RIDs). For example, a RID can be a hash of the content it identifies. Scalability for the rendezvous name space is provided by introducing the scoping of information, where each information object is published under at least one scope. A scope is identified in the rendezvous system with another flat crypto-identifer, known as Scope ID (SID). For instance, a SID can be a hash of the public key of the publisher. In other words, a scope can be seen as a reference to a collection of information and the same information can belong to several collections.

4.4.3.2 Rendezvous architecture

The basic building block of the rendezvous system is a server/router (the rendezvous node). A set of rendezvous nodes form a tree topology rendezvous network. Globally scalable rendezvous system is established, when a set of rendezvous networks is interconnected via their root rendezvous node(s). In the reference architecture interconnection is based on the Canonical Chord (CANON) DHT interconnection overlay.

Every scope in the rendezvous system must have one or more indirection points (also known as the scope home rendezvous point). Routing of control signals in the rendezvous layer is done based on the scope: A subscriber feeds a control signal to the rendezvous system through its first-hop rendezvous node and the rendezvous system routes the signal towards the scope specific indirection point(s) (if it/they exist). Indirection points are created as a logical entities in a rendezvous node(s) as part of the publication registration procedure: Publisher sends a registration signal to the trusted rendezvous node(s) for each publication.

A registered scope become reachable in a rendezvous network through process where a rendezvous node advertises its known scopes by sending the related reachability state(s) to its rendezvous peer(s) and rendezvous service provider(s). The service provider(s) further propagate this state to their rendezvous peers and providers. In the end, the process results that a root rendezvous node(s) retain all registration state(s) advertised by their peers and (transitive) customers.

4.4.3.3 Rendezvous Interconnection overlay

As stated above, root rendezvous nodes hold registration state of scopes of their rendezvous network. Scopes in other rendezvous networks are reached via an interconnection overlay formed by the root rendezvous nodes of each rendezvous network. The overlay is used to store <scope identifier, pointer> tuples, mapping the identified scope to the nearest rendezvous node responsible for it. Immediately upon encountering such a pointer in the overlay, the communication request is forwarded to the responsible rendezvous node using the forwarding path stored with the pointer. The destination rendezvous node can be hosted by any network reachable either on the underlying network or via the interconnection overlay. The overlay structure is virtual and does not require specific support from non-participating networks, such as upstream transit providers. However, the overlay depends on the participants having a common agreement on their relative position in the virtual structure. This requires a degree of mutual trust, and therefore entails obligations between the participants.

This concludes the brief description of rendezvous concept that has been developed in the PSIRP project.

4.5 Mobility

Native support of mobility and multihoming is an important feature of any new networking paradigm. Dynamic updates of the network layer routing information can be used to support both mobility and multihoming. However, such a mechanism has shown, in current Internet, severe limitations in terms of scalability of the routing system (convergence times and the size of the forwarding tables are impacted by the routing updates). In NetInf, mobility may be supported by leveraging the NRS element, which can be an integral part of the routing and forwarding system. This section first illustrates how native mobility is supported in an integrated name resolution and forwarding system like GIN (Section 4.5.1). Then, an alternative technical approach is outlined for the combined routing and name resolution mechanism based on LLC (Section 4.5.2).

4.5.1 Native Support of Mobility in ICN

In ICN, mobility should be supported natively. This requirement, however, needs to be clarified. *Native support of mobility* in ICN means the following:

- Mobility is supported seamlessly for any moving object (i.e., hosts, networks, information objects).
- Mobility mechanisms are embedded in protocols at the ICN networking level.
- Routing of data is optimised under mobility, that is, data is delivered over shortest or minimal cost network paths between moving objects (no triangular routing).
- Scalability of routing architecture is not affected by the mobility support. In particular, mobility support scales with the size of the network and the number of moving objects.
  - To be consequent, the routing state in forwarding tables should not be increased after an object move and the routing/resolution updates for moving objects should be handled in constant time.

Here, a possible approach for native support of mobility in the context of Global Information Network (GIN) [20] is proposed. Such an approach is based on the delivery system described in Section 4.4.1. In that system, a stateless delivery protocol is used to forward messages by name over heterogeneous networks, by addressing objects with ID stacks in writable packet headers. The forwarding system is only concerned with the routing among (fixed) infrastructure nodes. As a matter of fact, mobile objects attached to the fixed network are not advertised in the core routing protocols. Instead, mobility of objects is managed into the Name Resolution System.

The consequences of the above approach are the following:

- Infrastructure routing is stable and routing convergence is not impacted under mobility. Mobility does not affect scalability of the forwarding system, that is, the size of the forwarding tables is not impacted by the number of moving objects. Forwarding tables are updated only for infrastructure changes, not for relocations of connected objects. Scalability of the routing and forwarding system only depends on the size of the (fixed) infrastructure network.
- Since object mobility requires binding updates in the NRS, scalability of the overall architecture depends on the scalability of the NRS. In a scalable NRS, binding updates should be performed in almost constant time for relocations of connected objects, as the number of moving objects increases.

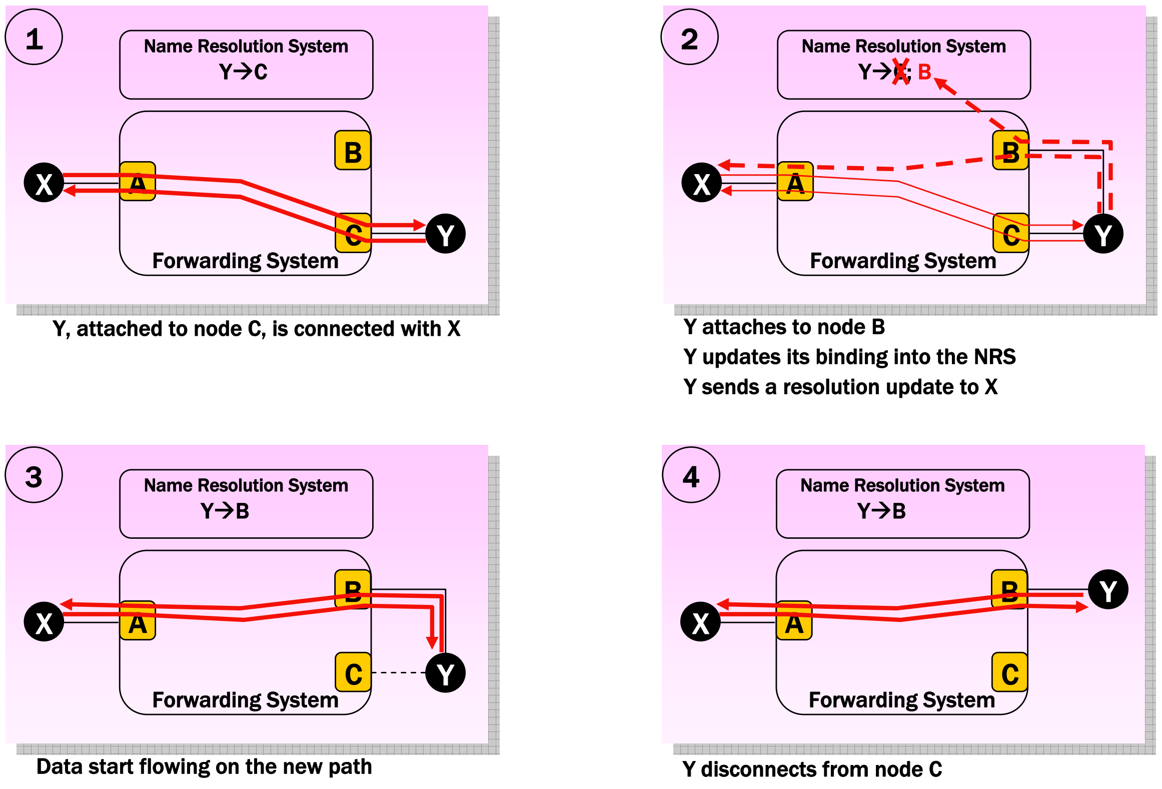

Figure 4.7: Setup before break

The MDHT system [19] can be adopted in this context, since MDHT has been specifically designed to implement the NRS of an ICN infrastructure network with high scalability (in terms of objects managed in the system), good latency performance and locality support. Information Objects (networks, hosts, data objects) are registered in the MDHT Dictionary. A binding record maps an object ID into a list of locations and other associated metadata and control data (policies, filters, etc.). A moving object registers the new location into the Dictionary. Outdated bindings for an information object are removed from the Dictionary:

- By the owner of the object, before/after a move to another location, or before disconnecting from the network;
- By the network, on binding expiry, or after detecting unreachability of the object.

A NRS built with MDHT may adopt a *Late Name Binding (LNB)* strategy for name resolution. Under LNB, the resolution process terminates into the destination or close to it. LNB may facilitate soft handover procedures to achieve *seamless connectivity*. In that case, the ultimate resolver is the target destination. The resolution request is forwarded to the destination(s) and contains the source address. The destination replies directly to the requesting source with its address. Then, a direct path can be established between the requesting source and the destination. When a terminal moves to another access point, it can update the remote peer(s) with its new location before breaking the old connection. This soft handover procedure, called Setup Before Break (SBB), is shown in Figure 4.7. When end-system Y attaches to a new access node, Y updates its binding into the NRS with the new location. Y sends then a resolution update to the other endpoint(s) and data begin to flow over the new path(s). Finally, Y can disconnect from the previous access node.

Mobility can take advantage also from multihoming (see Figure 4.8). Multihomed objects are registered in the Dictionary with multiple bindings. Seamless connectivity can be obtained by maintaining multiple parallel connections at the same time with the same target object, or with equivalent copies of a target object. Two cases are then possible:

- A unique object (e.g., a host, a network, any unreplicable data object) is multihomed if it is connected to multiple network endpoints at the same time. In that case, parallel connections between two endpoints can be used to improve reliability and support soft handover procedures.
- A replicated object (e.g., information object with multiple equivalent copies) is multihomed if more equivalent copies of it are distributed at the same time in different locations of the network. In that case, an end system can download a multihomed data object from multiple sources in parallel, with increased reliability of the data retrieval service.

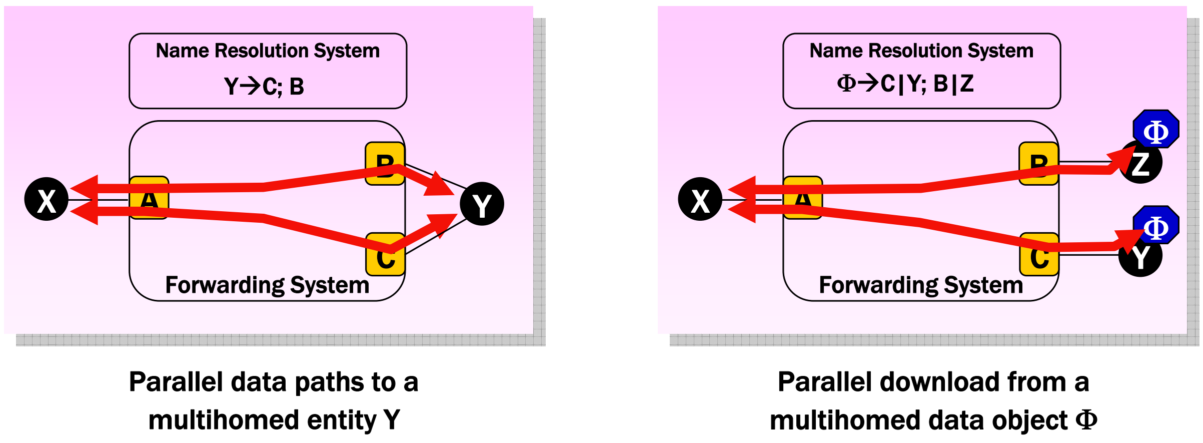

Figure 4.8: Multihoming and parallel paths

Another feature useful to support mobility in GIN is provided by the routing mechanism called *Resolution Fallback*, described in Section 4.4.1. The Resolution Fallback happens when a message cannot be forwarded by name with a simple lookup in the fast Next Hop Table (NHT) of a transit node. In that case the message is automatically sent to the node's Dictionary for resolution, that is, the message is diverted through a new resolution path. Such a feature can be useful to redirect messages that cannot reach the target destination on the current data path. That is, if the target object suddenly moves to another endpoint, or disconnects, the resolution system can provide a redirection (i.e., an alternate path) to the same or to another equivalent object. Therefore, the Resolution Fallback feature can be used to enable handover towards replicated or multihomed objects.

Another possible application of the Resolution Fallback is for Delay Tolerance Support. As an example, the NRS can activate a Custody Agent (CA) function with storage and forward operation to handle temporary disconnection of objects from the network. In that case, the Resolution Fallback can be used to redirect traffic towards a CA for destinations under intermittent connectivity.

4.5.2 Mobility based on Combined Name Resolution and Routing

As described in Section 4.4.2, support for mobility and multihoming can be based on a dynamic routing mechanism that adapts to topology changes caused by mobility and multihoming events.

Figure 4.6 illustrates how topology information in an edge domain consisting of edge networks, hosts, and IOs is registered in an LLC routing system. Based on this information, the routing system builds label stacks on demand which reflect the current topology. After a topology change which results in an invalidation of a forwarding path described by a label stack, a new label stack which describes a valid forwarding path across the modified topology is requested. Using this mechanism, a variety of mobility and multihoming scenarios can be supported. For example, the LLC routing system handles communication with mobile IOs located on mobile hosts, which in turn are attached to mobile networks, which in turn are attached to edge domains with dynamic topologies. All of the entities in such a mobility scenario may be multihomed.

The LLC support for mobility and multihoming is based on the LLC routing mechanism, which is sufficiently dynamic for such purposes. There are thus no LLC mobility mechanisms which are separate from the routing mechanism. The LLC routing Section 4.4.2 provides further details. It should be noted that the LLC approach to mobility and multihoming is based on the NetInf high-level architecture, such as the label stack, IO names, NRS, etc., see the high-level description of the NetInf architecture in Section 3.1.

4.6 Transport

NetInf is based on a general IO/chunk request and response forwarding scheme as described in Chapter 3. For transporting requests and objects in a reliable flow-controlled way, NetInf employs *NetInf Transport Services (NTS)*, which are described in this section.

The fundamental service that NTS provide is the *retrieval of chunks*, i.e., application data units for IOs. The NetInf chunking concept is described in Section 4.6.1.

The concept of a chunk-based request/delivery service enables a specific type of transport services that is typically called *Receiver-Driven Transport*, where transport transactions are rather controlled by a receiver (through explicit request) than by a sender as in most Internet transport protocols today. A conceptual description of the receiver-driven NTS operation is provided in Section 4.6.2.

In leveraging NetInf chunking and caching (also see Section 4.7), NTS provide specific support for efficient multipoint communication, which is conceptually described in Section 4.6.3.

As introduced in Section 3.2, NetInf's support for technology heterogeneity requires a transport service that can provide connectivity as well as transport services such as a reliable transmission over multiple different underlying network technologies and across multiple network domains. Two different approaches are being investigated for this:

1. NetInf can employ a **convergence layer** transport abstraction, where convergence layers provide NTS *chunk request and transport services* for a particular network link/domain. NetInf nodes implementing such convergence layers could be gateways between domains – or the equivalent of *performance enhancing proxies*, i.e., they would provide optimised hop-by-hop transport for a challenged network link.

2. Alternatively, NetInf can employ **end-to-end** transport protocols as in today's Internet which are based on the notion that a requestor requests chunks from identified nodes without intermediaries involved. Different models are possible, including the adoption of traditional transport protocols such as TCP but also the definition of new, receiver-oriented transport protocols.

The specific properties of these approaches are discussed in Section 4.6.4 and Section 4.6.5. Finally, the relation between transport and other architecture elements is studied in Section 4.6.6.

4.6.1 Chunking

In NetInf IOs often provide a structure that has to be considered for transport. Application Data Units (ADUs) such as video frames have to be transferred individually to enable streaming, selected access to certain IO parts etc. — such ADUs are called *chunks* in NetInf. A chunk is a fragment of a bigger IO and is the smallest addressable and verifiable unit on the convergence layer. Chunking is required for several reasons:

- For static objects, it provides a way to split and move an object through multiple paths, allowing to individuate and exclude faulty or malicious sources, in the assumption that each chunk contains authentication information. It also allows to cache only the most requested parts of an object, thus conserving resources on the caches.
- For dynamic objects, such as objects representing real-time video streams, it is the only way to guarantee that the object being received is indeed the one requested before receiving all of its chunks, as each chunk can be verified before the payload is delivered to the upper layer. This is true of course only if the chunks do contain authentication information.

Chunks are verifiable, which means that it is possible to verify that the data-load of a received chunk corresponds to the name of the requested chunk. Clients however do not have to verify them before passing the payload to the upper layer. The size of the chunk is not fixed and chunk size does not have to be uniform within the same object. There is no upper or lower limit to the chunk size, thus chunking can be effectively “disabled” by choosing a chunk size equal to the size of the object. Chunk size is decided basing on the intended purpose of the IO being chunked: a big file meant to be completely downloaded before use (e.g., an ISO image of a DVD) can be chunked in big chunks (even several Megabytes), in order to reduce overhead, whereas a live audio stream would need really small chunks, in order to minimise delay, and thus introducing a big overhead.

Since chunks can be verifiable, each one can be certified by the original publisher; the easiest way to do so would be either to put the signed hash in the metadata of the single chunk, or to keep a (signed) list of all the chunks in the metadata of the IO. This means that the chunking process can only be performed by the original publisher, in the case where authentication information is provided, as only the original publisher has access to his private key.

Object chunking can be either performed by a NetInf-aware application, or by the underlying NetInf layer. NetInf-aware applications can produce chunks that closely map the application-specific data that is being published. For example an application streaming live video could put each video frame in a single chunk. In any case a public key of the publisher is needed, in order to provide authentication data. Once the receiver receives the chunks that form the IO, they can be verified and/or reassembled. Progressive downloads or live streams can be passed to the application as the chunks are received, optionally after being reordered in the correct order. Since chunk size is arbitrary and determined by the publisher, chunks may need further fragmentation, which can be done by an underlying node-to-node convergence layer or receiver/sender pairs in an end-to-end approach.

Chunks have a unique identifier, similarly to the IOs. There is no restrictions on the format of the chunk ID. Ideally the chunk ID is easily calculable by clients (e.g., sequence numbers for streams, dates for newspapers, etc.). Chunks may be registered in the NRS, but doing so puts additional strain on the NRS, so only a limited number of them should be directly registered. Binding between IO and chunks can be placed in the metadata of the IO itself, for example as a list, or as a function to generate the names of the chunks (e.g., in case of a sequence). Chunks may have further metadata, like the name of the IO they belong to. The binding will provide verifiability of the chunks. If placed in the metadata of the IO itself, it can be a (signed) list with chunk ID and hash; if the chunks are generated on the fly, chunks embed all the authenticated information needed to verify the binding, and the IO metadata may have additional authentication metadata, for example the public key used to sign the rest of the object.

4.6.2 Receiver-Driven Transport

Transport of data happens between two (or more) nodes: one requesting IOs, and the other(s) providing (chunks of) such IOs. There may be an arbitrary number of nodes on the paths between the end-nodes. The nodes might communicate using a NetInf-dedicated transport protocol, which at the moment is still in the very early design stage. In fact, several different transport protocols are envisioned at long term in order to provide the different services needed; for example a reliable transport, or a best effort loss-resilient transport for wireless networks. The network architecture is pull-oriented. The receiver requests named chunks from the network and multiple NetInf nodes can attempt to fulfill such requests.

Applications on the receiver need not be aware of the chunk IDs. In such a case, NetInf-unware applications can request ranges of bytes of a named object and the underlying layers take care of translating the range requests into named chunks, leveraging the IO metadata. For more details regarding the application interface, consult the API Section 4.9.

In case of a two-step approach for name resolution, the locators returned by a query to the NRS are those of registered copies of the requested object. As the requests are forwarded towards any of the registered locators, any on-path cache that has chunks from the requested object will reply with the cached chunks. Any on-path node that did not have cached chunks of the object will cache all the passing chunks. This implies that non on-path near on-path caches will not be used. How to take advantage of such caches is still an open issue. A possible solution is that caches could register and unregister themselves in the NRS as object are cached or evicted from the cache. This solution puts more strain on the NRS; another possible solution is to do nothing, as not using non on-path caches should also yield sufficiently good performances.

IO chunks can be requested from an arbitrarily big number of different locators, and any of those requests could be fulfilled by any of the nodes on the path. How to conciliate that with traditionally end-to-end transport services such as flow control is still under research.

4.6.3 MultiPoint to MultiPoint Transport

In this section, the use of chunks and in-network caching is leveraged to provide multipoint to multipoint communication. The case of multipoint to point transport, i.e., the simultaneous download of the same object from different locations in the network, is first studied. In a second time, is the case analysed where the same content is requested at several points in the network, and provide the basis for a point to multipoint (i.e., multicast) transport.

4.6.3.1 Multipoint To Point Transport

The ability to retrieve data from multiple locations is a communication paradigm that allows enhanced features such as reliability to failures, multihoming, mobility, throughput maximisation and delay reduction. The routing protocol is responsible for the calculation of disjoint paths in order to guarantee some diversity among the different routes.

Multipoint to point communication is used in the Internet especially in P2P communication for swarm-based systems like BitTorrent for file sharing or PPLive for video delivery.

Path selection: The receiver is the data termination end-point where data requests are issued along the different available routes. The choice of the available routes to use depends on the performance of the data delivery, measured as attained throughput or response time.

The path selection mechanisms are similar to peer selection scheduling algorithms used today in swarm based communication, that is generalised here to a network of caches storing the requested data.

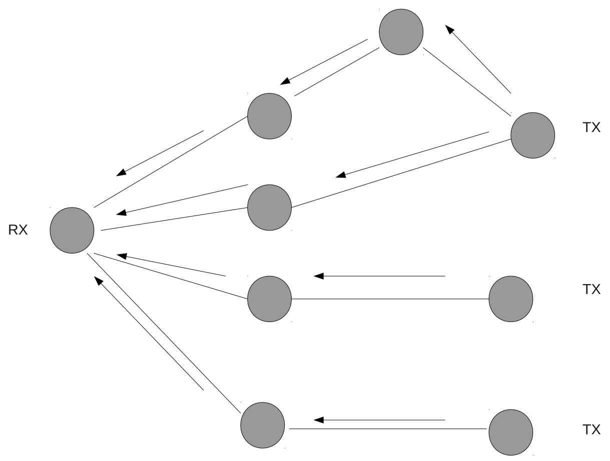

Figure 4.9: Multipoint to point communication paradigm

Usually, the routes made available to the receiver change very slowly compared to the way data is load balanced along such paths. Hence, we consider as static the set of protocols that build the transport infrastructure by calculating, maintaining and acknowledging available routes to the receiver. On the other hand, we consider as dynamic the protocols that defines the selection of the routes (and their frequency) to retrieve data, i.e., the load balancing algorithm itself.

Path calculation, maintenance and acknowledgment is the main objective of the routing protocol and are not consider here. However, the path selection process and the schedule updates are based on the information about the state of the paths that the receiver updates locally. Such path state information can be fed back to the receiver from the network or measured by the receiver itself from the performance measure.

Joint path selection and rate control: The receiver manages the use of available routes, load balancing data retrieval from the set of paths according to some aggregate objective. For example, in case the receiver wants to maximise the overall throughput attained from all paths, rate control techniques can be used to estimate the available bandwidth along each route. This is the case for large sized data retrieval. Rate control and path selection must be jointly chosen when optimising the overall objective. Such objective takes into account the fact that multiple users share the same network resources, e.g., bandwidth and storage. When resources are limited, the overall allocation is the result of the implicit coordination between the multiple receivers by means of the rate controller.

4.6.3.2 Providing multicast support

In NetInf, caching of data objects can be done on path. By simply allowing the network to cache the objects in its nodes, the data distribution is enhanced, since the network behaves like a distribution tree, whose relays are the network nodes. This provides a basic multicast service, for asynchronous data distribution.

When the request are simultaneous (e.g., for live events, like royal wedding or sport game broadcasts), the previous solution is not fully effective. An ideal multicast service requires firstly to

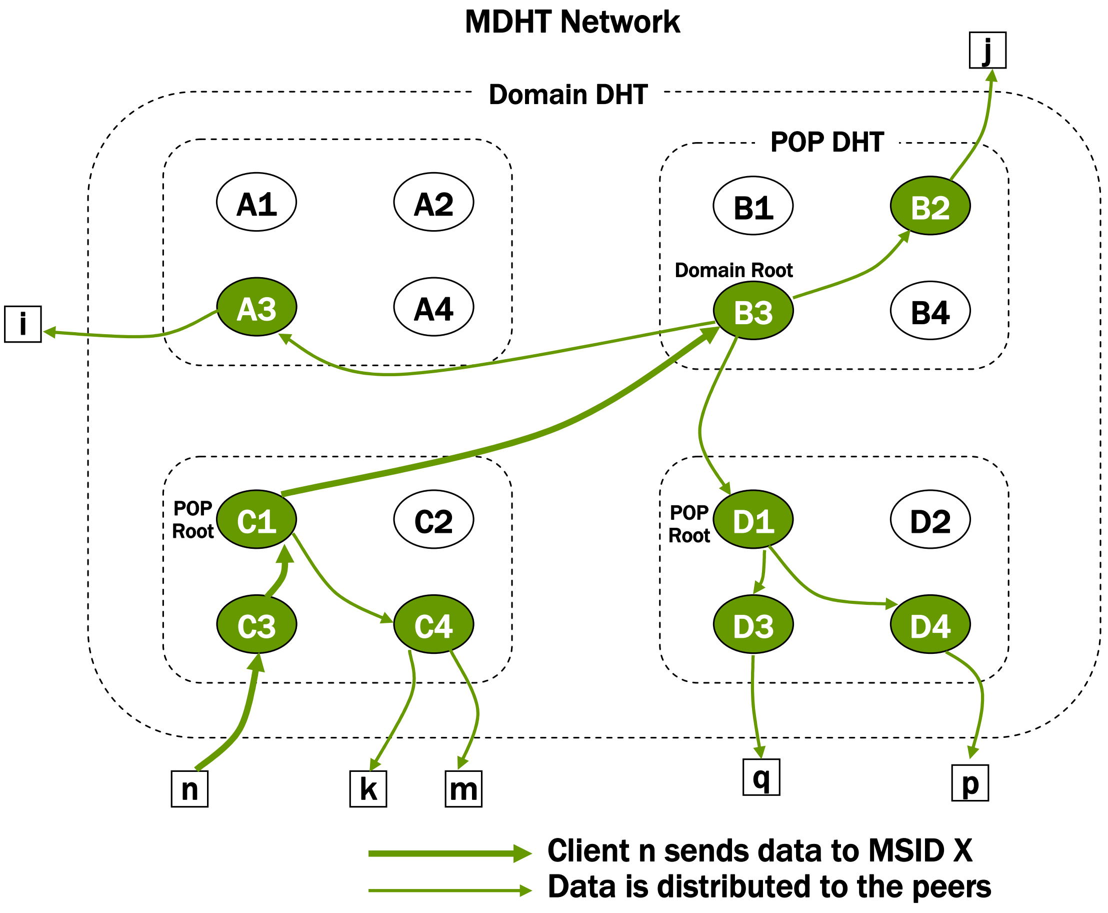

Figure 4.10: Multicast distribution of data in a MDHT Network

aggregate the identical requests, such as to avoid their duplication, and secondly to duplicate the data packets on their path to the end-users, in a way that serves all (requesting) end-users, and avoids to transport the same data packet twice on the same link.

A possible solution for this request aggregation is to keep some states in the internal nodes of the networks. More specifically, each (or at least some) server keeps a list of all the requests that it has forwarded upstream, and from which interface this request was coming. When the corresponding data comes back from the server (or possibly a cache), the data is forwarded to the corresponding interface. If identical requests arrive in the time interval, they are not forwarded, but their incoming interface is added to the interface list of the request, and the data is then duplicated upon receival at the server, and forwarded at each interface in the corresponding list.

It is also possible in NetInf to implement such a request aggregation scheme without keeping states in the internal node, by exploiting some features of its NRS. The NRS is a main component of the NetInf architecture. It provides a name resolution service with locality awareness, which enables anycast routing. An example of a NRS for NetInf is MDHT [19]. The hierarchical structure of MDHT is bound to the network topology. MDHT can be used as a multicast distribution tree, by using its network nodes as rendezvous points. A publisher can advertise a Multicast Session ID (MSID) into the MDHT. The MSID is registered at several levels of the hierarchy, and in particular at the top level (root node). Any subscriber can join/leave the multicast session by simply registering/deleting their IDs into the MDHT system under the MSID, with a *join/leave* primitive. MDHT nodes aggregate join requests on the path to the root. A published MSID then creates a shared distribution tree with root in the top DHT level. Any (allowed) source can then send data to the MSID. MDHT nodes deliver the data up to the root MDHT node and down the MSID tree. The process of sending data to a MSID and distributing to a group of peers is shown in Figure 4.10 in a typical MDHT network.

Note that the MSID tree built over MDHT is a topological tree which preserves the locality requirements. Note also that MDHT can provide a platform for access control and session control to the users of the multicast service.

4.6.4 NTS Convergence Layer Approach

The multi-domain/multi-technology approach of NetInf requires a transport architecture that is able provide IO transport between different links, networks and organisational domains. The domains serve two functions. The first is to allow different transport technologies, for example, IPv4, IPv6, DTN, Ethernet, and MPLS, in different parts of the network. The second is to provide a means to divide the network along administrative boundaries. In the following, a focus is on the first function on interconnecting technologies.

Routing over a set of interconnected domains is an important function to consider. The assumption in this section is that each domain potentially has its own addressing scheme, that is, there are no global addresses. Another assumption is that each domain can route on object IDs using either name-based routing or name resolution.

4.6.4.1 Convergence Layers

In order to support that heterogeneity NetInf employs *transport convergence layers*, i.e., network (link) specific transport functions with a service interface to higher-layer common NetInf transport. The main service of the NetInf transport is a request/response-based delivery of IO chunks, i.e., publisher-defined sub-elements of IO. This concept is described in the following. Inspired by

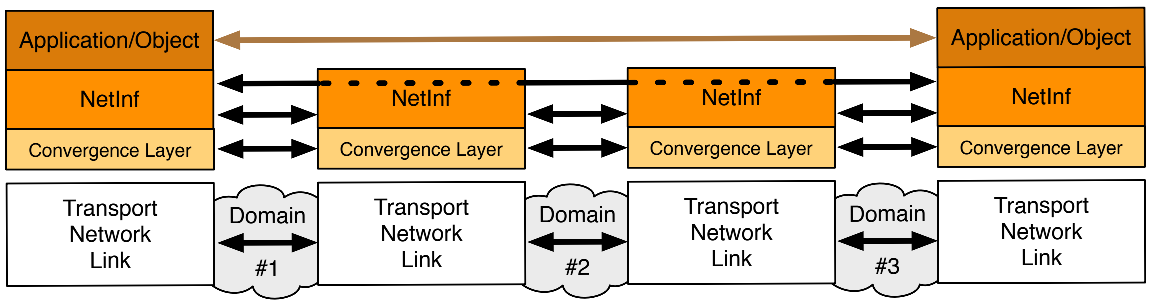

Figure 4.11: Convergence layer

the DTN Bundle Protocol Request For Comments (RFC) 5050[23], NetInf's convergence layer, illustrated in Figure 4.11, provides a cross-domain transport layer, leveraging underlying link layers (which can be transport protocols on their own, e.g., when NetInf is layered on top of TCP). Chunk (and therefore IO) transfer is performed on a hop-by-hop basis. The convergence layer allows for nodes in different domains to exchange chunks and their metadata across the NetInf network. The convergence layer provides basic node-to-node transport for requests and chunks. Each convergence layer adapter is responsible for leveraging a given legacy network or transport protocol in order to enable exchange of requests, IO chunks and associated metadata.

A convergence layer adapter provides an interface over a legacy protocol from one node towards another node or a set of nodes allowing to send or receive requests or chunks. The convergence layer adapter is provided with the object to be sent (chunk or request) and a lower layer address. Such lower layer addresses can be, for example, locators returned by the NRS. Selecting the correct protocol and address is a routing decision.

$$\begin{tikzcd} A \arrow[r, dash, "TCP"] \arrow[r, bend left=60, "interest"] & B \arrow[l, bend left=60, "chunks"] \arrow[r, dash, "Ethernet"] \arrow[r, bend left=60, "interest"] & C \arrow[l, bend left=60, "chunks"] \end{tikzcd}$$

Figure 4.12: Simple example of multi-domain chunk exchange among different convergence layers

For example: consider three nodes, A, B and C. A and B are connected with a TCP link, B and C with an Ethernet link. The node A wants an IO that is stored on C. The TCP convergence layer adapter on A sends an interest packet for the IO towards B using an active TCP connection, according to routing decisions. B will then forward that interest towards C, which will reply with the requested IO chunk. The Ethernet convergence layer adapter between C and B will take care to deliver the chunk and its metadata, fragmenting it and retransmitting fragments as needed in order to guarantee correct delivery. Finally, B will forward the chunk and its metadata on the reverse path to A using the TCP connection.

Open questions Since the convergence layer operates on a hop-by-hop basis, it is not yet clear how intrinsically end-to-end functions, like flow control, can be achieved.

One possible solution for congestion control could be to perform it on a hop-by-hop basis. This poses interesting challenges in case of DTN hops in the path.

4.6.4.2 PDUs for interdomain connection

The interdomain interface of the convergence layer in terms of abstract PDUs is described. For a minimal ICN interdomain service, two functions are needed, one for requesting information objects, and one for returning objects in response to requests. We define one PDU for each as follows.

- `REQUEST(object-ID, destination-hints, return-path)`

  This PDU requests the information object `object-ID`. `Destination-hints` help with routing the request. It is zero or more routing hints in the form of partial source routes. It can contain IP addresses, URLs, node IDs (as in Section 4.4.1), Ethernet addresses, Global Positioning System (GPS) coordinates, scope IDs, etc, that is relevant for the particular technology of the domain. `Return-path` is a (partial) source route collected en-route consisting of a list of addresses similarly to `destination-hints`. It enables the response with the object to find its way back to the requester.

- `DATA(object-ID, return-path, data)`

  This PDU is sent in response of a `REQUEST` PDU. It transfers the actual `data` for the object `object-ID` using the `return-path` from a `REQUEST` PDU.

More parameters are planned in these PDUs, for instance, to handle requesting and delivering parts of an object (chunks), and for security purposes.

Open questions The above described design with a return path collected by the request PDU has both benefits and drawbacks. A benefit is that no additional state other than the internal state of the convergence layer needs to be kept by the routers in the network; the state is instead encoded in the PDUs. The drawback is that it becomes very hard to aggregate requests for the same object as they are routed towards a source. The latter is important for supporting live broadcasts when a large number of recipients request the same object more or less simultaneously. The alternative interdomain solution with state would not have the `return-path` parameter and would have router state instead. This is a very important design trade-off that require more investigation. Our on-going prototyping work is expected to be helpful in this analysis.

4.6.5 NTS End-to-End Approach

Having a convergence layer, which handles hop-by-hop transport at the chunk level, has the obvious consequence that the resulting network node architecture is store-and-forward. As chunks have variable size, with no upper limit, they must be assembled and fragmented in the convergence layer at each node. Since underlying networks cannot guarantee an ordered transmission of chunks, and interleaving of packets of multiple chunks can always happen, a node interface has to manage the input process of several chunks at the same time. Therefore, some state is required to handle chunk transmission in a network node. Furthermore, handling chunks hop-by-hop could involve operations at the chunk level (e.g., security checks, cache lookup) to be performed on each NetInf node of the path. This can be useful in some cases, as for caching, but redundant in others, as for security checks. The convergence layer should also guarantee reliability of transmission of chunks hop-by-hop, which requires to perform integrity checks (error and loss detection) and retransmission of underlying lost PDUs, with a considerable state, chunk by chunk, to be maintained into the node.

A different approach consists in handling chunks at the endpoints of the network, in an end-to-end transport session, like in TCP/IP. In this case, a stateless forwarding protocol can be designed to route by name data packets over an interconnection of underlying heterogeneous L3 or L2 networks. In this hypothesis, chunks are data units defined and managed at the endpoints of the network and do not need to be handled in intermediate nodes. Packets of the stateless forwarding protocol should be relatively short (e.g., less than 1480 bytes), in order to be encapsulated in most underlying PDUs, in particular in the IP protocol over Ethernet. Note that fragmentation and reassembly can always happen over underlying networks with shorter PDUs, but, in this approach there is no need to manage chunks in a convergence layer below the end-to-end forwarding protocol.

Since chunks are managed in the endpoints of a communication, the caching cannot be performed hop-by-hop at the chunk level. Instead, caching can be done in the resolution nodes and/or into the edge nodes with an approach similar to that of the proxy caching.

An example of such a stateless forwarding protocol and architecture, which both integrates name resolution and routing and enables interdomain interconnection over heterogeneous L2 or L3 underlayers, is given in Section 4.2 and Section 4.4.1. In the following, a proposal for an end-to-end transport mechanism which adds congestion control to the above stateless forwarding protocol is described.

A receiver-based congestion control Congestion control can be seen for a number of reasons as part of the basic network function. Therefore this section introduces an approach for supporting explicit congestion feedback directly in the NetInf layer. The considered solution for the above goal is based on an adaptation of the Rate Control Protocol (RCP) [24] and, in principle, can be applied to any ICN routing and forwarding protocol able to deliver packets between named entities; these include the proposal discussed in Section 4.4.1. Congestion control mechanisms implemented at the NetInf layer can then be exploited by different sessions using possibly different upper transport protocols layers, depending on the applications requirements.

According to this proposal, the feedback rate is computed by the NetInf network nodes following the algorithm defined by RCP, which has been shown to achieve good performances [25]. However the behavior of the congestion control protocol is modified to become receiver-oriented. Moreover the proposed scheme does not require to keep any state in the nodes but only in the communication endpoints. The NetInf packets carry a congestion header that comprises a number of fields used by the rate control protocol; they include a Round Trip Time (RTT) of the flow, a specification of the *desired rate* preferred by the receiver for receiving data, and a *feedback rate* providing the indication from the network of the allowed transmission rate along the path. The receiver sends requests for new data packets as allowed by the current feedback rate; in case of a congestion window based implementation, its size dictates the number of outstanding request packets. The request packets specify the desired receiving rate in their congestion header. The reception of return DATA packets trigger the progression of the congestion window and provide the rate feedback, from the nodes on the backward path, to adapt the congestion window. The resulting flow throughput is therefore the ratio of the congestion window size to the flow RTT.

When such a mechanism has to be deployed in a scenario where the assumption of a truly cooperative environment can not be made, the issue of how to deal with non-complying end systems has to be addressed, as they can lead to unfair allocation. The possible solutions consist either in providing incentives for users to behave correctly or in implementing policing mechanism, typically at the edge of the network, for security purposes. In the latter case, to reduce the implementation complexity of the enforcement mechanisms, statistical sampling techniques can also be used.

4.6.6 Relation with other Architecture Elements

Naming: a fundamental aspect of NetInf transport is the use of chunks, *i.e.*, smaller, addressable and verifiable parts of a whole IO. This allows to use more efficient transport (and caching) strategies. However, as chunks need to be named and secured in a scalable way, a joint effort including naming, security and transport expertise will focus on how to define, name and secure precisely chunks;

Name Resolution and Routing: the new paradigm of ICN in general, and NetInf in particular, is to enable requests made on content, instead of enabling connection between host. This means that the requesting node will ask for IO or chunks, without specifying the location of the content. NetInf will then be responsible for locating the content and delivering it to the requester, possibly using several source locations. This function will be performed more specifically by the name resolution system and the routing protocols, and have an obvious impact on transport. In particular, the way and the format used to deliver this information locally to the network (or the node who is requesting the content) will be crucial when the content of the request can be served from several nodes and caches. A joint strategy, including routing and topological knowledge, but also the current load of different paths, need to be developed. This strategy is obviously at the crossroad between name resolution, routing and transport.

Caching: since requests on NetInf are made on content rather than on specific host, caching becomes an easy-to-deploy solution to reduce the load and the cost of a network, as well as increased the quality of service seen by user. A relevant paradigm for caching is the use of *on-path caching*, meaning that chunks are locally cached when they cross a server, and then delivered for any new request for the same chunk crosses the same server. When the cache becomes full, some chunks are removed from the cache, according to a *replacement policy*. Such caches typically do not register their content to a global NRS, due to the frequent updates they perform. They instead opportunistically serve some requests, and forward other requests. However, their presence will impact the performance of transport layer, and reciprocally, the way that chunks are routed and transported will impact the performance of these on-path caches.

4.7 Caching

Efficient information caching and distribution of content is a key element of scalable and well performing information-centric architecture. Caching is used intensively by the network in order to achieve traffic and performance optimisation. Therefore, it is important to mention that in NetInf, caching capabilities should be fully integrated in its native operation mode.

4.7.1 Background and Introduction

In the 1960's when packet switched networks were invented memory was very expensive. Consequently, considering large amounts of memory to be part of the network was not an option. Over the years things have changed dramatically. Since 1970 memory prices have dropped by a factor of 10 million, i.e, if a 1970 vintage router could afford 1000 bytes of buffer memory a router in 2010 would have 10 Gigabytes of buffer memory. Recent advances in flash memory will of course allow for much larger memories than that. Traditional web caching is widely deployed in Internet where mainly web proxies are responsible of caching user's HTTP traffic. However, this is not sufficient in future and especially when considering the ICN networks where more powerful means to further advance caching and its benefits are available.

The problem addressed by caching can be described as follows. Consider a network as shown in Figure 4.13. Essentially, the figure shows a number of end user nodes (clients) and a number of servers which are interconnected by a network of routers. Both, routers and servers, represent candidate nodes where caching can be done either by using in-network caching or information republishing. This is a simplified view of the current computer networks but will serve the purpose of describing in-network and information caching.

Assume that a number of data objects are being distributed in the network. Client nodes generate requests for those objects, these requests are routed to appropriate network nodes, where the requested objects are stored and then the objects are downloaded by the clients. A number of quality metrics can be associated to this process. From the user's point of view, it is important to get a desired object as quickly as possible. For large objects this means that there is enough bandwidth between the client and the node hosting the object. For small object this means that latency between the client and the object hosting node is as small as possible. From a network operator point of view, it might be important to reduce the traffic in the network as much as possible. Thus a network operator might desire to place objects close to the clients requesting them.

One of the problems that caching aims to solve is as follows: What is the best possible placement of objects across the nodes in the network as to optimise the metric of interest. So given the request made by the clients, the goal set by the network/application designer (minimise latency/network traffic...) as well as the constraints in the cache sizes, to which cache should each of the objects be assigned? The dynamic scenarios naturally fit into this description, the set of objects is not constant or the set of nodes/caches changes.

In practice, this problem translates to devising algorithms that monitor relevant parameters, e.g., request frequency for specific objects, and shift objects between caches as to gradually approach the best assignment of object given the observed demand for them.

Another task in caching is to create mechanisms to query the cached objects. One can for instance search objects only from the caches that are on the query path to origin server, broadcast the queries to a cluster of caches or query object locations from a centralised server.

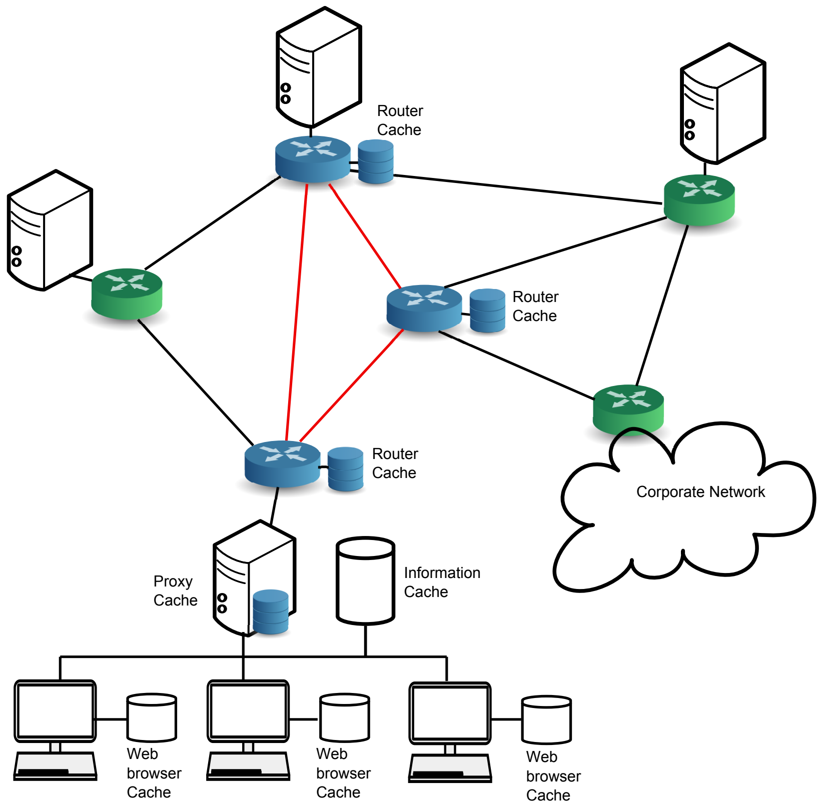

Figure 4.13: Possible locations for deploying caching

There are also a number of practical problems related to caching that will be addressed in later deliverables. Some of the most relevant ones are listed here. Imagine that an object arrives from some other cache then should the current cache assume that the object is unchanged, i.e., its content did not change while being transferred among caches or is there a need of maintaining integrity of objects? Secondly, a cache may want to evict objects for business reasons, e.g., keeping objects from partners with business relationships for longer - assuming that's allowed (net neutrality might prohibit it) then can this be integrated in caching algorithm and how will the algorithms perform then if they meet this requirement.

The following sections describe different approaches for solving the above stated caching problems in network architectures such as NetInf. Section 4.7.2 describes an approach where a NRS is used to keep track of the cache content. Section 4.7.3 describes the approach called proactive caching where the monitoring of the subscriptions of the data items is used in order to enable caching functionality for specific content through scoped content replication. Finally, the approach in Section 4.7.4 deploys the Bundle Protocol (BP) in order to achieve effective caching in DTN environments.

4.7.2 Cache Management

Management of caches involve a number of trade-offs between different performance metrics. For instance trade-off between hit rate in a cache, that is how often requests can be satisfied. This can be calculated for the requests or on byte basis, i.e., number of bytes found in cache divided by number of bytes requested. Thus, in some situations it is beneficial to keep less popular content in the cache if the information object is large in size. On the other hand expunging a large object would free large amounts of memory for storage of other information objects. Also, note that this issue is not relevant if chunking is used for information objects.

To address these issues, a model for the popularity of information objects is needed. It has often been observed that the Zipf distribution model this with high accuracy. However, a realistic setting of the involved parameters of the distribution is needed. The problem can either be approached analytically as for instance Rodriguez, Spanner and Biersack [26] did or by simulation. Since [26] addresses web caching and makes assumptions such as infinite memory space their model might not be applicable and therefore simulations to quantify the different trade-offs might be needed.

Another trade-off in cache management concerns bandwidth-lookup latency-hit rate. If caches in different locations cooperate the hit rate can be increased since all caches need not to store every information object. However, the cooperation needs some signaling between the different caches consuming bandwidth. In addition, search for objects also increase latency during lookup. Several cooperative placement algorithms have been developed for web caching. The placement strategies in these algorithms vary. For example the algorithm may place popular objects near clients to reduce network load, avoid storing duplicates of unpopular objects to increase the number of unique objects or balance cache space contention. Similar methods can be used also in NetInf environment. In the simplest form, cooperation can be implicit, since the use of of fetch cost-based algorithms (e.g., Greedy Dual-Size [27]) leads to the situation where co-located caches do not cache the same content. Typically the cooperating caches exchange information to improve placement decisions, e.g., greedy, amortising and expiration age based placement algorithms [28][29]. Below a discussion of how local NRS could be exploited in object placement is provided.

It is assumed that caches register their content in local NRS. Then NRS queries make it possible to find objects from any cache. Downside of this is that caching is limited to data that can be registered in NRS whereas in on-path caching an object can be partially received for instance due to multipath/multisource transfer or errors in streaming. Therefore this kind of caching is more suitable when objects are fetched separately to the cache.

In order to control replication of objects in the cooperating caches the placement algorithm should be aware of at least current object placement and object popularities. Instead of gathering placement information directly from the caches local NRS can provide it since it receives object registrations. Also object popularity information could be obtained by counting NRS queries. The purpose of these solutions is to reduce the amount of signaling needed for cache collaboration.

The actual placement decisions can either be centralised into an external placement algorithm or there can be centralised part that assists local algorithms in the caches. In the former case the centralised algorithm must be aware of the cache sizes and the object sizes to be able to make the placement decisions. In the latter case centralised part gathers information from local NRS and optionally from the caches, pre-processes this information and delivers it to the caches. Each cache makes the final decision on what objects they fetch and evict. The changes in cache contents are then signaled to local NRS in registration messages. The minimum information that should be delivered to the caches is object identifiers and references for fetching the objects as returned in NRS queries.

4.7.3 Proactive Caching

Proactive caching can be used as an additional functionality for the existing cache servers, meaning that all standalone caching functions can operate in parallel with the proactive caching strategy. A standalone caching function refers to opportunistic (cache all or part of content flow with round-robin style) and/or reactive (cache monitors popularity itself) caching. The proactive caching strategy can be divided into 2 functions; i) the monitoring function through which all subscriptions are passed and ii) the "proactive client" function, which is located in the caching nodes/storages and through which the caching nodes can be remotely instructed by the monitoring function to subscribe a new content. There could be multiple instances of these functions depending for instance on the used network topology, i.e., how many exit points/gateways the network has, and cache deployment/management, i.e., how many cache servers are supporting proactive caching.

In proactive caching, the monitoring function commands the cache(s) to request (subscribe) certain content (publication), for instance based on its popularity measured by its requests. The monitoring function can also support filters and policies that are used to specify what subscriptions are or are not monitored. This can be based on the information IDs such as NetInf IO IDs used in subscriptions, or publisher's authority IDs or the type of information. Publishers can also mark the published information explicitly to be included or excluded by the monitoring functions. The inclusion of data objects is challenging for global use cases due to large number of data, however the closer the caching is deployed to the subscriber, there will be less number of data in the inclusion list, e.g., inclusion list is an alternative. The exclusion list has typically better scalability potential.

Proactively cached content is under cache management since the cache space is a limited resource and therefore less popular/useful objects also need to be occasionally evicted. The monitoring function does not keep track of the content of the caches, thus the placement algorithm is executed on the cache nodes so that a cache selects the objects for eviction and it is allowed also to reject cache commands received from the monitoring functions. The communication between monitoring function and caches can then be unidirectional. If a cache decides not to download (subscribe) an object or evicts an object, a monitoring function may observe more transmissions and report the object again later.

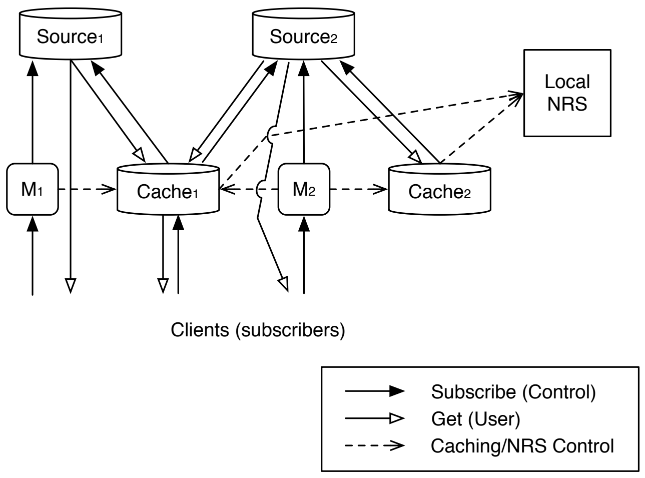

Figure 4.14: An overview of the proactive caching

Once the cache has received the whole content (all chunks) it starts to act as a new local source (publisher) for the content, which typically requires that a local NRS is updated accordingly. After this, all new subscriptions for the content in the cache's locality can be served locally by the cache and no external signalling is needed. Figure 4.14 shows an overview of the proactive caching where two monitoring and caching functions are present. The monitoring functions are located so that all subscribers content requests are visible for them and caches subscribe the content from sources when they are instructed to do. In addition, when a monitoring function does the "positive" caching decision, it may contact one or more cache servers and instruct them all to cache the content; like M_2 does when it instructs both $cache_1$ and $cache_2$ to subscribe the information from $source_2$.

The information passed in the cache commands may vary depending on the selected algorithm. The monitoring function can for instance report one object at a time or a set of popular objects periodically. Object placement decision can be based for instance on object popularity, access cost and size. Some of this information can be exploited by the monitoring function and some can be sent to the cache for the final placement decision.

4.7.4 Caching in DTNs

Delay- and Disruption-Tolerant Networks DTN [30] make use of the Bundle Protocol [23] to provide store-and-forward networking. DTN is typically used in edge networks to overcome intermittent connectivity and long delays. Making use of caches within these networks can greatly reduce the delay in retrieving information. Different routes within the DTN experience different levels of connectivity; nodes may be fully connected or it may take days for a request to be answered. While some requests will always need to traverse these high delay links, correctly positioned caches can reduce the number of these.

DTN nodes use custody requests to prevent the need for retransmission of information. Information sent within the DTN is contained within a bundle. The sender may request that nodes along the path take custody of the bundle and answer any retransmission requests. These custody requests provide similar functionality to that of a cache, using network storage to reduce the amount of network traffic. However this information can not be sent other than as a retransmission request despite it being stored in the network.

By using the Bundle Protocol Query (BPQ) Extension [31], information already in the DTN may be queried. This extension to the Bundle Protocol turns every node into a potential cache. While constrained nodes may choose not to store bundles not destined for themselves, any nodes currently accepting custody of bundles could act as an on-path cache. Unless content is known to reside within the DTN information requests are sent via a gateway node. Beyond the DTN the information may be retrieved from an external cache via an NRS. As many of the internal links are prone to disruption, NRS inside the DTN is often impractical.

A DTN node wishing to request an IO may create a typical request, wrap this request inside a bundle, append a BPQ query block describing the IO request and send it to a DTN gateway. If any node on the path to the gateway holds a cached copy of the requested IO, as described by the BPQ extension block, they may answer the request. Otherwise the DTN gateway will unpack the IO request from within the bundle and use it to query an external NRS. The gateway will then wrap the response inside a bundle, append the BPQ response block and forward it to the requesting DTN node.

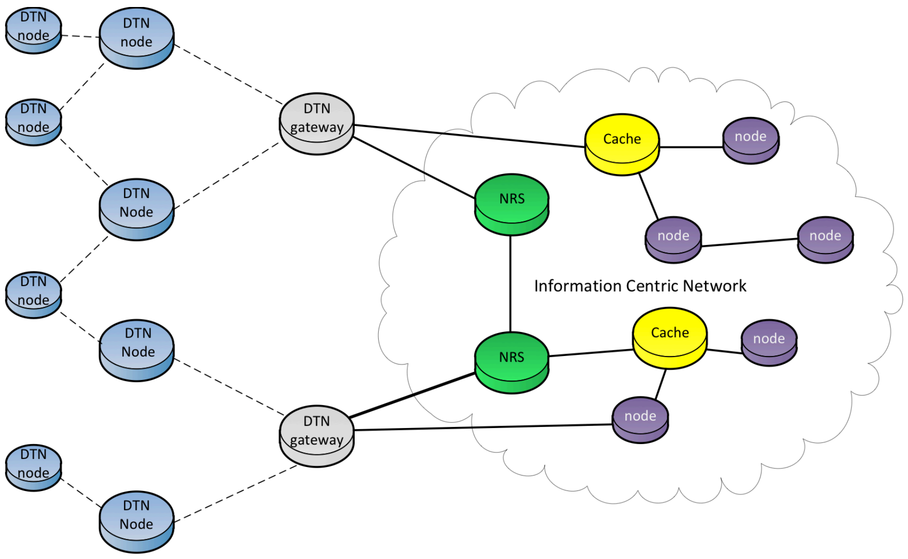

Figure 4.15: A DTN topology

Although any node in a DTN has the potential to cache information it is particularly useful for nodes adjacent to a disrupted or slow link to do so. In the case if a static topology the location of these nodes are often known in advance. Figure 4.15 describes a typical DTN topology. While requests made from within the network will result in new content being added to the caches, content may also be pro-actively added to the DTN caches. In this instance a content publisher may choose to send information via the gateway to an internal DTN node. When the gateway appends a BPQ block the content becomes query-able within the network.

4.8 Security

Security is one of the important SAIL themes – acknowledging the fact that any serious Internet-scale architecture design has to address security requirements for its core design. The general SAIL approach to security is described in [14].

Security considerations for ICNs differ somewhat from more typical host-based networking scenarios. Broadly, it can be stated that content-security is more interesting while less interest is needed in channel security, when compared with host-based networking. However, many aspects of handling security in ICNs are only at an early stage of development and the expectation on this part of the NetInf architecture will evolve significantly as a result of further analysis and experimentation.

In this section is the security considerations relevant to NetInf starting with a threat model, then considerations for relevant security services and finally some of the security mechanisms and infrastructures that may be required for a large scale ICN.

Note that more "traditional" threats (e.g., snooping) and countermeasures (e.g., TLS/Virtual Private Networks (VPNs)) are still relevant in ICNs, for the purposes of this architecture it is assumed that those are applied as usual as determined by a system-level risk analysis. So, unless there is a new aspect, further details of these threats and countermeasures are omitted.

4.8.1 Threat Model

This section lists and briefly describes relevant threats in an ICN. The focus in on threats that are either new, or that have a new aspect in an ICN. Each threat also has a two letter acronym for each of reference, e.g., *Content Mismatch* is referred to as [CM].

Content Mismatch [CM] is where there is a failure to map from some input (e.g., a search-string) to the intended objects in an ICN. The bogus content presented could make use of spam-like tricks, e.g., use of "cousin-domains" or internationalisation tricks, in an attempt to fool a human user into accepting the presented object as being correct.

Content Snooping [CS] is essentially a traditional threat, but where the probability of occurrence, and perhaps the impact, may be made worse due to the object-level (or chunk-level) in-network storage inherent in an ICN.

Privacy Invasion [PI] is perhaps made potentially worse, since each cache in an ICN can see entire objects, and is thus in a better position to track actions, when compared to a more traditional router.

False Content Injection [FC] is where a bad actor publishes an object that is intended to be returned to requestors instead of the correct object. This is analogous to actions taken by companies working on behalf of copyright owners in publishing "bad" versions of music, that may contain a warning or advertisement for a legal equivalent of the intended object.

Unauthorized Access [UA] is where a bad-actor accesses an object that was intended for a limited audience, not including the bad-actor.

Cache Pollution [CP] is where a bad-actor pollutes the content of a cache, resulting in incorrect returned objects, possibly as a denial of service.

Table 4.1: Mapping between ICN security services and ICN threats

Service	CM	CS	PI	FC	UA	CP	FA	MR
Name Security	y			y		y		
Content Integrity	y			y		y		
Content Authentication	y			y		y		
Content Confidentiality		y	y	y	y			
Authorization		y		y	y	y		
Provenance	y							

False Accusation [FA] is where a bad-actor attempts to make it appear as if a requestor has requested an object, when that is not in fact the case.

Mis-Routing [MR] is where a bad-actor modifies routing, so that ICN transactions are routed in a way beneficial to the bad-actor, but detrimental to some other good-actor, possibly the requestor or a network provider. This could encompass deliberate unfairness in caching, for example, refusing to store some objects in a cache due to the lack of a commercial agreement, or causing one's objects to be pulled into a cache in order to provide better service at someone else's expense. Some of these cache related threats may be quite similar to search engine optimisation tricks.

4.8.2 Security Services

This section considers the security services that could be integrated with an ICN, in order to counter the threats above, as well as more traditional threats. The same two-letter acronym approach are used for ease of reference.

Table 4.1 indicates how each of these services can act to counter the threats listed above. Note that a "y" in that table does not mean that applying the relevant service fully counters the threat, but only that the service is relevant to countering the threat.

Naming Security [NS] refers to services that provide cryptographic strength binding between an object name, and the form of the object returned by the ICN in response to a request. There are various mechanisms that may be used for this service, as described in Section 4.1.

Content Integrity [CI] can be related to *Naming Security* or can be independent of naming, and provides cryptographic assurance that the returned object value is unmodified since some previous event such as object creation or publication.

Content Authentication [CA] again can be related to naming security and provides assurance (possibly cryptographic) as to the identity (via some identifier(s)) of entities associated with the object - perhaps an "owner" or the entity that published the object. In principle, this service could be split into many different services, each authenticating a different involved-entity, analogous to how with email, one can separately authenticate the author, sender sending MTA and sending domain. For now, it is unclear whether or not there will typically be one or more authenticated entities associated with objects, nor is it clear in which roles these entities will be acting.

Content Confidentiality [CC] is a service that ensures that the plain text value or an object is only visible to authorised entities. There may be different forms of this service with different "end-points," corresponding to the usual end-to-end vs. hop-by-hop security services offered in host-based networking.

Authorization [AZ] refers to the general ability of an entity to control which other entities are supposed to be able to gain access to an object.

Provenance [PR] is a service that allows authorised entities to trace the overall actions of an ICN upon a set of objects.

As can be seen from Table 4.1, the threat of false accusation [FA] is not yet countered by an identified security service and the services that may be used to counter misrouting [MR] are also for future study, as NetInf routing matures. In the former case, one might expect that Provenance and Content Authentication may play a role, and in the latter case, traditional host based security mechanisms may be useful (or even perhaps sufficient) to handle NetInf routing security. However, as has been seen with the recent development of the Resource PKI (RPKI) for BGP routing in the Internet, it may also be the case that a separate routing security infrastructure of some sort may be needed. In that case, some aggregation method for routing (equivalent to an Autonomous System Number (ASN)) may be required before a sensible infrastructure could be developed.

4.8.3 Mechanisms and Infrastructure

This section is not intended to be a complete description of a full set of security mechanisms that can provide the services described above, but rather outlines the one mechanism that may currently be sufficiently mature to warrant description (basically, just name-data integrity) plus some text outlining known issues with using current (typical) protocol security mechanisms to provide the ICN security services.

Mechanisms related to Naming Security are perhaps the best developed in most ICN proposals, and the NetInf architecture is no different. The main premise is that the core security mechanism, on which many services can be built is to provide a hash-based way in which an object-name can be compared against the object value returned by the ICN. This mechanism is called name-data integrity since that is the core mechanism provided - that is, this mechanism does not provide confidentiality nor authentication in any form. A proposal for the basis of this is specified in an Internet-Draft [15] and in Section 4.1 and so is not further described here.

Name-data integrity can fairly easily be used as the basis for various basic forms of Naming security, but the same mechanism can also be extended to provide Content Integrity and Authentication services, and perhaps less obviously, Provenance, when combined with some form of semantic model for possible operations on data. However, once on goes beyond basic name-data integrity, then one faces the requirement for some form of Public Key Infrastructure (PKI), whether of the traditional RFC5820 [32] variety or some other (based perhaps on XML Key Management Specification (XKMS) or DNS-Based Authentication of Named Entities (DANE) or some ad-hoc PKI such as is implicit in CCN).

While traditional host-based security services have successfully made use of PKI (largely via Secure Socket Layer (SSL)/TLS or IPsec VPNs) none of these PKIs have approached the scale of the number of Internet users. This may mean that new security mechanisms or some form of aggregation of security (e.g., a host-based outsourced digital signature service) could be required once the number of object producers that wish to be able to provide Content Authentication and related services approaches the number of users of the Internet (which is to be expected). Such an object-signing service would be closely analogous to the certification authority role in a traditional PKI (or more correctly, an attribute authority). [33]

A harder problem however awaits once looking for scalable mechanisms to support content confidentiality. In this case, the basic issue for an ICN is how to establish a shared secret between an object recipient and some party trusted not to leak the object content, perhaps an object creator or publisher trusted for that content. Technically, a Kerberos like approach [34] could suffice to provide some level of content confidentiality (and also Content Authentication and other services), however, no cross-domain (or realm) Kerberos-like system has ever approached the scale required for a general ICN.

Privacy preserving mechanisms such as onion-routing or other obfuscatory routing schemes may or may not become an inherent part of an ICN and the fact that in-network stores return objects may also act so as to make tracking user actions more difficult. However, the fact that in-network stores cost money will likely result in their operators being willing to share information, including tracking information, with third parties.

Evidence gathering mechanisms would be required, along with cryprographic and time based mechanisms in order to provide a Provenance service. While a subset of this service (an audit mechanism) can easily be envisaged, it is not clear how an ontology, which appears to be required, could be developed so as to allow semantically meaningful provenance services to be integrated with NetInf. In other words, for an end-user to know what has happened to produce a given object, they would need to have that presented to them in terms they understand, which appears to require an ontology that represents all of the things that might happen to an object.

4.8.4 Conclusions

There is an expectation that the security aspects of the NetInf architecture will develop significantly in the course of the SAIL project. The approach planned will be to iterate on the analysis of the threats and security services along with suggested mechanisms to provide those services. However, this will be done with usability to the fore, as experience shows that hard-to-use security mechanisms do not get used, or else do not fulfill their goals.

4.9 API

As shown in the NetInf reference model, the API is between the application layer and the NetInf layer, see Figure 3.5. The NetInf API described in this section will be evaluated after a first round of prototyping work. If the evaluation results in modifications of the API, the modifications will be described in the next NetInf architecture deliverable.

4.9.1 API Overview

The NetInf API supports interaction between an application and an IO. The interaction is handled by a set of API methods which will be described below. The IO can be a static information object like a file, a dynamic information object like a live stream, or an end-point of an interactive application such as telephony.

The NetInf API has a small number of methods which are sufficiently general to support a variety of application types, such as content distribution, interactive applications (interactive web services, person-to-person communication), and machine-to-machine communication. The API supports a traditional request-response mode of operation, where a requester retrieves an IO as a direct result of a request. In addition, the NetInf API can operate in a publish-subscribe mode, where a requester requests an IO on a subscription basis, and retrieves the IO when it becomes available. Also, the NetInf API supports search of IOs based on metadata.

The API methods are PUBLISH, REQUEST, RESPONSE and INDICATION, see Figure 4.16. The exact semantics of the methods are determined by their parameters. For example, method parameters describe the service model (direct retrieval, subscription, interactive communication, DTN type of communication, or search).

The NetInf API has some similarity with a client-server API, where the requester in Figure 4.16 corresponds to a client, and the publisher corresponds to a server. However, there are also important differences. For example, unlike a traditional server API, the publisher API supports registration and and announcement of IOs.

The NetInf API is receiver-oriented in the sense that data cannot be pushed to a remote entity, but must be requested by the remote entity. [3]

[3] In DTN type of communication, there may be a need to push data directly to a remote entity in order to avoid the delay associated with alerting the remote entity, so that it can request the data. This issue is for further study

A node supporting both the requester and the publisher API can handle bidirectional communication for, e.g., peer-to-peer applications. The end-points of the bidirectional communication have IO names. Communication is initiated by a caller which alerts a callee by making a REQUEST to the callee. The callee acknowledges the alert by making a RESPONSE to the caller, which may include user data. This RESPONSE also includes a REQUEST to the caller to start sending user data back to the callee. The caller does this using the RESPONSE method. Bidirectional communication is thereby established.

Since the NetInf API is receiver-oriented, it does not have a method which supports upload of content by pushing data to a remote entity. Instead, the local entity which wants to upload content must alert the remote entity in the same fashion as in the case of bidirectional communication described above. The remote entity can then respond by requesting the content from the local entity.

A strategy function below the API decides which interface of the local node to use, and which copies of an object to retrieve. The application can communicate with the strategy function by providing policies as method parameters.

The API supports the Representational State Transfer (REST) style of interaction, which reduces the amount of per requester state that needs to be kept in the network. [4]

4.9.2 API Methods

PUBLISH A publisher uses the PUBLISH method to register an IO name and associated metadata with the NetInf name resolution and routing systems. The metadata may include keywords that can be used when searching for an IO.

REQUEST A requester uses the REQUEST method to send a request for an IO, using the IO name as an object identifier. The NetInf layer maps the request to a specific node, or set of nodes based on the data registered via the PUBLISH method. The REQUEST method can also register subscriptions for IOs based on parameters of the REQUEST method. If the IO is not available, the subscription will be stored by the NetInf layer until its TTL expires. If the IO becomes available before the subscription has expired, the IO will be returned to the requester in a RESPONSE message. Variations of the subscription parameters include:

- Notification only — When an IO becomes available the requester is only notified of its availability. A renewed REQUEST is needed to retrieve the object.
- Immediate delivery — As soon as the IO is available it is delivered to the requester.
- One-time subscription — The subscription expires when the IO has been delivered.
- Continuous subscription — Any updates of an IO will be delivered to the requester until the subscription expires.

The REQUEST method can also be used for search purposes. Keywords extracted from metadata are then included in the method parameters.

RESPONSE The RESPONSE method is used when responding to a request message. The semantics of the response depends on the REQUEST parameters. For example, a node can return a Bit-level Object (BO), or metadata associated with an IO.

[4] The per requester state needed in interactive applications is stored by the requester, and is transferred to the remote application object in a request message. The application object calculates a state transfer, and returns the new state to the requester in a response message. Since the requester stores its application state, there is no need for the remote application object to store this state.

The NetInf layer uses the RESPONSE method to deliver the requested IO to the requester as a result of a successful match between a request message and an IO instance, or a successful match between a subscription and a publication, or a cache hit (alternative *a* in Figure 4.16).

If the remote IO is interactive, the REQUEST method can express a request to the remote IO to perform a task, and the RESPONSE then includes the result of this task (alternative *b* in Figure 4.16).

The API can return the following types of responses.

- A BO that corresponds to the IO requested, or a part of the BO.
- An IO with its associated metadata.
- Status messages, e.g., information about the result of a REQUEST. The NetInf layer can for example inform the client that the request has been sent via a DTN, or that a subscription has been activated for the requested IO.
- Search results in the form of IOs and associated metadata.
- Event notification to a requester when a subscribed IO has become available.

The destination identifier of the RESPONSE method is the name of the requester that sent the request message. The NetInf layer routes the response message to the node of the requester.

INDICATION The INDICATION method informs the publisher about the result of the PUBLISH method.

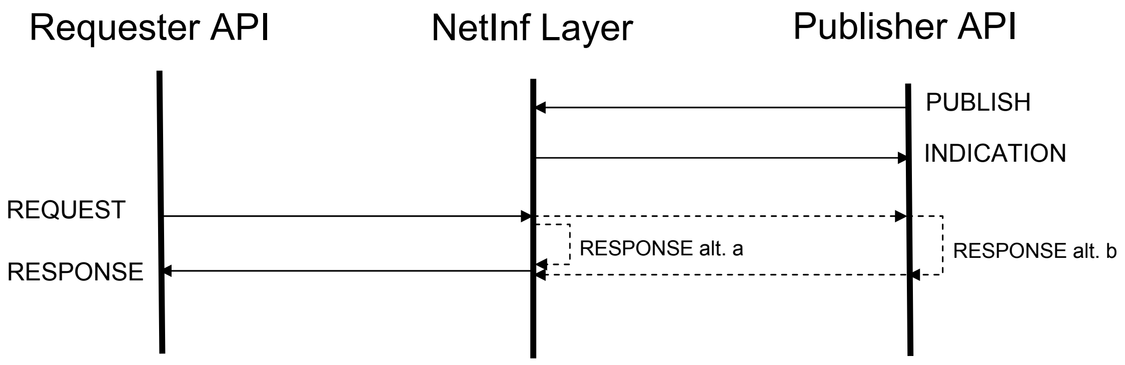

Figure 4.16: NetInf API

4.9.3 API Method Parameters

Below we describe a set of parameters for each of the API methods.

PUBLISH parameters

There is a need to differentiate between registrations for originals and registrations for cached copies, since there is a need to make sure that at least one BO is available as long as the IO is registered in the NetInf layers.

- IO ID
- Scope
- TTL for caching. Setting TTL=0 means the IO should not be cached. This way the publisher can avoid the problem with outdated copies in the caches, but will not have the benefits of NetInf object distribution.

- Metadata (size, cacheable, publisher signature, keywords, etc.).
- De-registration of an IO.

REQUEST parameters

- Name of requested IO.
- Scope — Used to restrict which domains NetInf should try to retrieve the IO from. How the scoping should be defined and which mechanisms should be used is for further study.
- Recursiveness — Full recursion means that the REQUEST method should return the complete BO that corresponds to the IO. No recursion means that the next step of indirection information is returned (i.e., a new IO with its associated metadata). It could also be possible to request the result of a specific number of indirections.
- Retrieval policies — Restrictions on which network interfaces to use, maximum cost of retrieving object, security policies applicable to the retrieval, etc.
- Subscription — Indicating that this is not a one-time retrieval of an IO, but also any updates of an IO that occur within the TTL should be delivered.
- TTL — Indication of how long the application is willing to wait for the response, so that the network knows when to drop the request if no response arrives. If the REQUEST is of the subscription type, then the TTL is the lifetime of the subscription.
- Partial retrieval of objects — Request for parts of an IO by specifying a byte range, or a sequence expressed as set of frames or a time sequence, see chunking in Section 4.6.2.
- Keywords for search.

RESPONSE parameters

- Returned IO ID.
- Returned metadata.
- Returned BO.
- TTL for caching.
- List of destination IOs for the response message.

Security parameters are needed for all methods in the NetInf API. These parameters are for further study.

5 NetInf Applications

This chapter summarises the use case and scenario work done in the SAIL NetInf and introduces the set of selected application scenarios that will be implemented as NetInf prototypes. The scenario work was also contributed to the Socio-Economics and Regulation research activity done in the SAIL project. There was a special attention on one of the NetInf scenarios in [13] with a business analysis of the NetInfTV scenario (see below). These scenarios are further divided into specific NetInf use cases, eventually leading to validation through prototyping of even more detailed application scenarios. The use case and scenario work is concluded, but they are eventually validated and demonstrated through the continuation of the prototyping work.

The use case and scenario work is divided into 4 categories;

NetInfTV: This scenario represents a heavy content distribution perspective of NetInf. The scenario focuses on video distribution scenario even though the proposed concept (and NetInf itself) is not actually content type specific. The scope of the scenario itself is wide, covering all the possible video content (live, on-demand, etc.) and both open and closed content distribution systems. The use cases, however, narrow the scope; i) a video distribution in an e2e context and ii) a video distribution in the specific context of an Internet service provider.

Next Generation Mobile Communication Networks: The starting point of this scenario is the current cellular networks that are still evolving from having been primarily used for telephony to mobile data services. Such networks also evolve from a centralised and static architecture into a more distributed and dynamic one, for example with femto cells and more dynamic routing. In this context, content and service delivery to mobile users should have a special focus to optimise the user experience and access and network resource utilisation.

Developing Regions: In the developed world, network connectivity is ubiquitous, reliable, fast, and cheap. In many developing countries, and in remote regions of developed countries, on the other hand, the situation is more problematic, since a dial-up connectivity to the Internet is norm. If broadband connectivity is available, it is very expensive. Even when connectivity is available, there is often a problem with reliability of the service. This is even more apparent in many regions where the power supply is very unreliable, with frequent power outages which mean that you cannot rely on connectivity to always be there resulting problems of reliability and capacity and coverage areas. The population density is often very high in these regions, so as popularity of the cellular networks grows (and in particular, if popularity of data traffic over them grow), there is likely to be a great capacity problem - either in the wireless spectrum, or in the access networks. NetInf could provide a better, more robust and efficient, solution to the communication needs of these communities than the existing Internet protocols. The information-centric approach and mechanisms and the inherent caching features as well as its improved resilience to disruptions make NetInf a good option here.

Social Networks: This scenario studies how Facebook[1] or any other social networking application would be implemented from scratch in a NetInf environment. What would be necessary and what would be the advantages if Facebook was to be implemented from scratch in a NetInf environment? The scenario work also includes the study of well-known problems and issues

[1] `http://www.facebook.com`

in the current Facebook implementation; issues that can be at least partially solved by using NetInf approach and mechanisms.

In the following sections, the selected application scenarios that are used in the prototyping work are described. More detailed prototype description for each application scenario can be found in Section 6.2.2.

5.1 Localised CDN

A localised CDN application scenario is based on the use of the proactive caching strategy described in Section 4.7.3 together with the Rendezvous system and the NetInf NRSs. This application scenario was developed based on the *Next Generation Mobile Communication Networks* scenario and its *user generated content* use case. There is both a local and global NRSs, which is needed to support the Rendezvous's scoped publishings. The proactive caching is divided into 2 functions; the monitoring and caching functions. Both of these functions are in the operator network where the main benefits of the proactive caching can be easily concluded: By making the cache to retrieve the popular (or preferred based on policies for instance) content, the operator can save in transit costs, because subscriptions can be served at its own network. In other words, a content source can be brought close to the point in the network where the demand is high for the content. This provides savings in cost and network resources for the operator as well as a lower latency for the local content subscribers. In addition, there is a side effect; when the local cache serves all new local subscriptions, then this hides the information popularity from the outside, e.g., from the original source/publisher. This might introduce new provisioning challenges, i.e., how to ensure that the locally republished content is provisioned properly and that the original publisher is part of the value chain. Figure 5.1 shows an example of localised CDN application where caching decisions have done by the monitoring function for the contents from three different CDNs; A, B and C. As the result of these decisions, the contents are subscribed from their origin CDNs and re-published again with a new scope in the operator network. It may not be in the interest of the operator to let subscriptions from other networks to enter this rendezvous point, because it is designed to optimise resource usage in the local domain(s) defined by the operator. Therefore the rendezvous point can be hidden from the global rendezvous system.

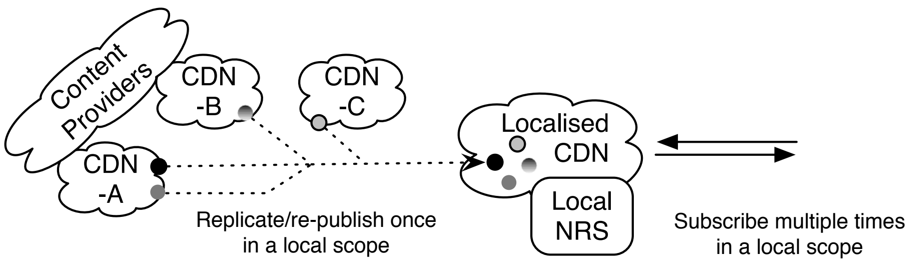
Content Providers
CDN -B
CDN -C
CDN -A
Localised CDN
Local NRS
Replicate/re-publish once in a local scope
Subscribe multiple times in a local scope

Figure 5.1: A localised CDN application

The monitoring function is supported by all potential gateways/edge routers though which all users' (control) traffic is going to ensure that all subscriptions can be monitored and related caching decisions can be done. In addition, the proactive caching complements the existing caching functionalities, but it does not require their existence, thus it could be considered to be a standalone caching functionality. An example of information-centric architecture that could easily be used to implement the proposed localised CDN application model is the rendezvous architecture, where rendezvous nodes provide a natural location for subscription monitoring Subscriptions are routed within the rendezvous network and typically each domain would have a rendezvous node that acts as a gateway to the global rendezvous system/external networks. Because the rendezvous architecture can also be used as an overlay on top of various routing and forwarding architectures, it is a good example of how a localised CDN application can be deployed in the current networking environment and how it could look like in practice.

Next a more pragmatic and small scale example where content providers and CDNs are replaced by end users and their (user generated) content is represented. In this scenario, a user publishes a new content via a rendezvous network and after this, the content is requested by other users in another network. The first content request (subscribe) triggers the proactive caching in the subscribers' network and results in the content's re-publishing in the local scope. The local NRS is also updated with the publishing information from the rendezvous network. This information contains RID and SID that are unique for the re-published information and by which the re-published content is addressable in the rendezvous network as explained in Section 4.4.3. Next subscription to the same content is then resolved to the local rendezvous point, which forwards the subscription to the local cache, which then serves the local user's request. This is presented in Figure 5.2, where the monitoring function when it sees the content request A from 1st user does a cache decision for the content A and instructs the cache to fetch the content. Once this is done, the caching function does not proceed with the registration, including both the rendezvous publishing and the local NRS update,until the whole content is retrieved. However, it should be noted that if the cache has standalone caching functions active, it may already have cached some chunks of the content A. Therefore, when the cache is requesting the content A from the publisher, it will request only the missing chunks.

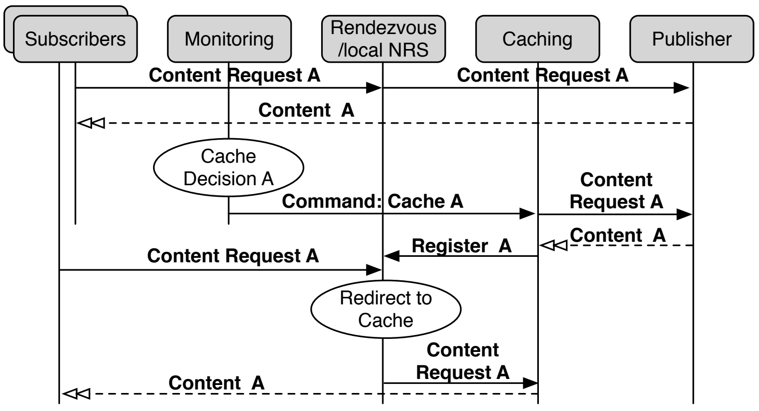

Figure 5.2: A localised CDN signaling example

5.2 Event with Large Crowds

The "Event with large crowds" application scenario is based on the use case developed in the *Next Generation Mobile Communication Networks* scenario work. The application scenario utilises NetInfs ability to perform local caching and local object location resolve (broadcast) to solve content download problems in Large crowds with limited uplink bandwidth.

Dynamic local networks are formed in order to optimise the usage of a limited shared uplink connection. Figure 5.3 represents an example scenario where it is assumed that every node has

3G/Long Term Evolution (LTE) capabilities. All the nodes connected by an edge are close enough to communicate using low range technologies (Bluetooth , Wireless Fidelity (WiFi)). Some nodes have an additional fast uplink connection (WiFi in this example).

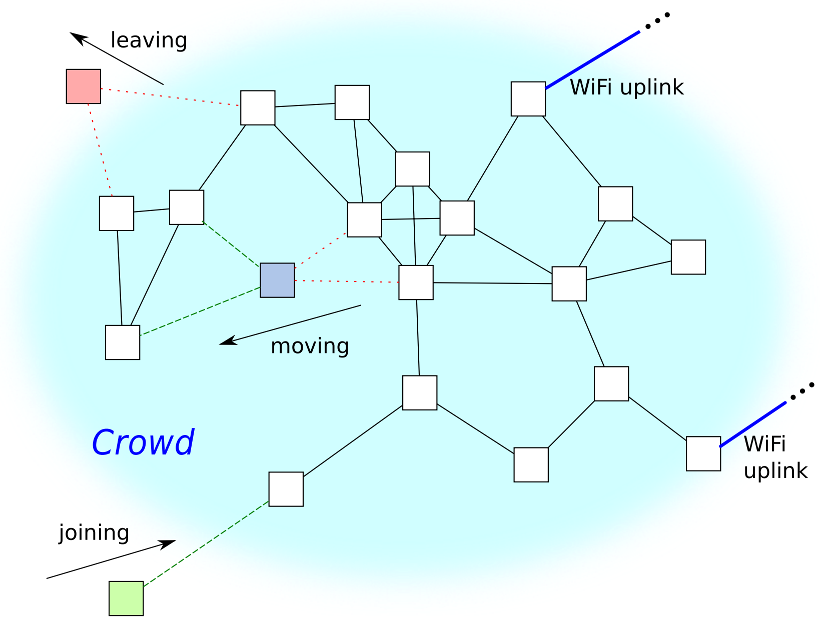

Figure 5.3: Large crowd scenario

An example of such scenario can be an event, planned or unplanned, with thousands to millions of participants, such as football matches, demonstrations, festivals, accidents, etc., where all the participants have a slow shared uplink connection, like 3G/LTE or even WiFi. In such a scenario, nodes can seamlessly join, leave or move into the crowd without affecting too much the performance of the neighbouring nodes.

Nodes create an ad-hoc mesh network between themselves using Bluetooth, WiFi, and/or any other suitable technology. A suitable routing protocol is implemented in order to be able to direct the requests towards the uplink. Nodes try to cache all the content they can, in order to optimise bandwidth usage.

5.2.1 Assumptions and Desired Features

It is assumed that all client nodes and all base stations are fully NetInf-enabled. It is possible to leverage IP, in order to ease migration. By tailoring object search, name resolution and source selection mechanisms for the event crowd case, the local peer to peer like content distribution can be optimised.

5.2.2 Operation

One solution is that nodes with an uplink connection will advertise their capabilities. When a given content is required by some client, it is cached along the path. Since most of the clients will mostly request the same event-related content, requests of popular IOs will likely hit the caches. IOs are first searched on the local net segment, with broadcast requests to neighbours.

Mobility support allows a node to roam around the crowd without being negatively affected and without disrupting the functionality of the other nodes. In order to provide the best user experience, the content should be chunked in small enough chunks, so that moving nodes will not cause big disruptions in case of live streams.

From the point of view of the application, there are no differences between this scenario and normal operation. The application requests some IOs and probably receives them. The standard APIs for IO retrieval are used. All the crowd-aware logic is in the NetInf (underlying) layer.

By adding a domain to the NetInf name (Domain Name/NetInf Name) an RendezVous Server (RVS) location can be resolved trough existing DNS. The BO content location is then resolved in the RVS. This resolution schema can be implemented on top of existing IP infrastructure. The obvious advantages with this solution is that it enables migration from existing Internet, and also that it is possible to build prototypes using off the shelf TCP/IP technology.

5.3 Delay Tolerant Video Distribution

The Delay Tolerant Video Distribution (DTVideo) application scenario is designed to provide video content to a disconnected community, such as the ones described in the *Developing Regions* scenario work. In particular, during the design of this application, intended end-users from the Saami indigenous people has been consulted regarding what they want from an application like this, and it will be field-tested in the mountain region of Padjelanta, Sweden in July 2011. This will be made possible through a network integrating NetInf with the Bundle Protocol[23] for Delay Tolerant Networking through the BPQ extension block[31] as described in Section 4.7.4.

An example of a network where this application would be used is shown in Figure 5.4. A remote community is connected to the global Internet through a DTN using a *village router* and some end user hosts. These nodes run the bundle protocol with the BPQ extension. Somewhere in the connected network there is a BPQ server that is able to globally resolve video names and retrieve the content from its source (in the figure, there is a single video source, but there can be multiple video sources as well; similarly, the BPQ server is colocated with the DTN Internet gateway in the figure, but it could be located elsewhere as long as there is a route for bundles to be sent to the BPQ server).

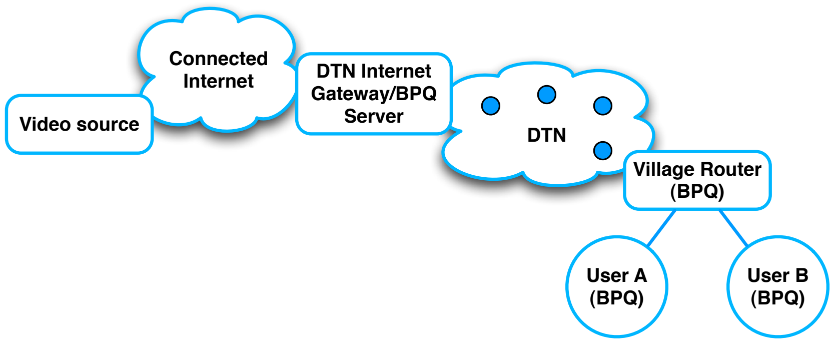

Figure 5.4: DTVideo overview

When new video content is requested by a user, an attempt to resolve the object name will be made locally to see if the content is already available in the local cache. If that fails, a request for the content will be sent over the DTN in a bundle using the BPQ extension block, addressed to the BPQ server. Through the use of BPQ, DTN nodes on the path to the Internet gateway will also query their local caches to see if the content is available there. If the content is found, the request will not be forwarded further and the content will be sent back to the requester. If the content is not found on the path, the request will eventually reach the Internet gateway where the name can be globally resolved and the content retrieved. The benefit of using this system can be easily seen by looking at an example. Assume that *User A* in Figure 5.4 requests some video content X. This has not been previously requested so it is not locally available and a request has to be sent over the DTN to the BPQ server that resolves the name of the video object and returns it to *User A*. Since BPQ is used and as the village router is BPQ enabled, on its way back to *User A*, the bundle containing the video object will be cached in the village router. If *User B* now request the same video object, the request will only be sent to the village router, where the object will be found in the cache and returned to *User B* at once, saving resources and reducing the delay for the user.

A local web portal can be run on the village router in order to serve the content that is available in local caches to the end users, allowing them to connect with any device capable of TCP/IP communication, making the system accessible to as many users as possible. End users that run NetInf/DTN software will also be able to aid the system in local caching and distribution of content, allowing for an organic growth of the system, as well as being able to use their devices for viewing the video content out of range of a village router.

The use of the BPQ extension can be incrementally deployed as it is not mandated that all nodes in the network or even on the path used by a bundle supports BPQ. Nodes that do not support the BPQ extension will simply forward the request or response on towards its destination without considering caching actions. Similarly, the use of BPQ does not impose the use of a certain routing protocol in the network, but can be used in conjunction with any of the existing routing protocols. There might be a use for routing protocols that are optimised for BPQ networks such that they for example would send a request on what would seem to be a detour in order for it to pass through a well known cache of popular content. Work on such protocols is still future work.

For content that is expected to be popular (such as daily news programmes), it is advisable to also do some proactive caching in a subscription type of manner so that such content is proactively pushed to the caches in order to reduce delay and improve user experience. Such subscriptions can be set up by automatically generating BPQ response bundles with the correct video content name in the BPQ extension block and sending this to a community gateway, or some other device where it should be cached.

In addition to this delivery of video content to the remote users, the application will also operate the other way around, giving the local community a way of making video content available to a larger audience on the global Internet.

5.4 Active Information Objects

To optimise the user experience content and services should be provided from servers placed on geographical locations nearby the users. In networks where users are mobile the content and service placement should be more dynamic than in the fixed Internet and adapt to the variations of the geographical user population and service popularity. This allows more efficient usage of the network resources and higher Quality of Experience (QoE) for the users due to lower latency. In particular this is important for more complex services than basic content delivery, where a service consists of programs running within the network and the interactions between terminal and servers may add multiple RTTs of latency. The mobile operators may use their knowledge of the network state and user locations to dynamically move services and content in order to optimise the user experience and network resource usage. The concept of mobility can be extended to cover mobile objects which are able to move while maintaining their reachability and in some cases also continuity of ongoing communication sessions.

Active information objects is a concept that is used to develop an application scenario that connects cloud networking with information-centric networking. This application scenario is based on on the *Next Generation Mobile Communication Networks* scenario work. While information-centric networking has so far focused on retrieving information from the network, this application scenario considers objects that can also be applications or processes, analogous to object oriented programming. There are several different aspects of this where information-centric approaches can help to realise the vision of supporting services in a distributed computing infrastructure.

One use case where active objects may help deployment is in deployment of flexible video delivery networks, addressed by the cloud networking work package in one scenario. The active objects may be software that can be installed on nodes in the network to support the distribution infrastructure. This would allow the network to scale with the demand from users.

Name-based resolution and routing

When software processes are treated as objects the same routing and resolution methods can be used to find them as for other information objects. Hence, it will be possible to route a service request to any available instance of a service processes in the network, analogous to how information can be collected from any location where a copy is stored. The processing objects may also need to be interconnected with data objects. This can also be handled by the resolution system by letting the active objects request other objects. For example, if two persons should be extracted from two different video streams and merge them onto a background from a third stream, this might be written as: *Merge(Backgroundstream,ExtractPersons(Inputstream1), ExtractPersons(Inputstream2))*. There may be several ways that the objects, the requesting client and the resolution system could interact to compose the service. The most straightforward seem to be that the methods of an object accept other objects as arguments and look up these objects using the resolution system. By letting an object that will connect to another object doing the lookup itself it will also find the instance with the best location.

Active objects

A generalised model of information objects would be one way to support more general service delivery. An active object may have executable methods which implements different functions. Some methods could implement well known standard methods for maintaining the object in the network, such as registration in a resolution system or copying itself to a remote system. An object may also have non-standard methods, which would implement the main functionality of the object. The available methods can be indicated in the metadata, in particular the non-standard methods could be found that way.

Processing nodes

Execution of the methods should in general be done on the server where the object is located. This implies that there is a need for processing capabilities on the nodes, while storage is sufficient for passive information objects. The requirements on the runtime environment could differ between different objects. Hence, there is a need to decide how to address this, one approach is to provide the requirements in the metadata, another would be to specify one standardised environment, but the latter would have drawbacks in terms of limiting the possibilities for services to evolve.

Placement of service components

Efficient operation of the network will to a large extent be based on good placement of processing nodes and different service components. This may be compared to the placement of caches and data objects in the caches in the case of CDNs, but the problem will be more complex. For example, it is necessary to consider that some components will need to be interconnected between each other. Furthermore, the nodes may differ in their processing capabilities, which adds another dimension to the storage and bandwidth constraints of the caching problem. The placement should also be based on the different characteristics of the objects, for example the number of users, the requirements on communication, storage and processing and the delay sensitivity of the communication. Moreover, mobility of both users and objects should be supported. The goal is also that connections should be maintained during mobility. A general mobility support has to take into account that some instances of objects are unique while in other cases there are multiple equivalent copies. In particular processes that keep user specific state have to be treated as unique objects. Hence, this case is more common for active objects than for passive.

Security

Security is an important topic for a network with active objects. A first issue that is similar for passive objects is the protection of data and software from unsolicited usage and copying. A second problem that occurs specifically for active objects is protection from malicious code.

5.5 Local Collaboration

The purpose of the local collaboration application scenario is to investigate the benefits of ICN for applications that are not of the mass distribution type. The scenario is frequently occurring in our own project work and is inline with the *Social Networks* scenario work. A group of people at a project meeting is sharing documents through one or more server-based systems, for example a Subversion version control repository. There are also other tools, for instance a wiki system, that are of similar nature. The Subversion repository stores a shared file tree and additionally keeps track of all modifications, or versions, facilitating joint document writing and code development.

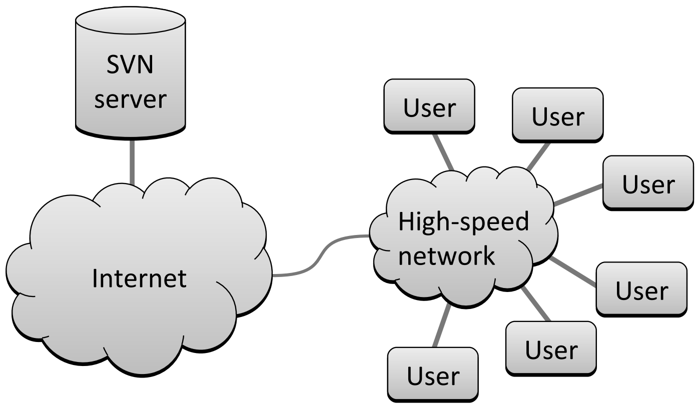

Figure 5.5: Local collaboration using Subversion

Figure 5.5 illustrates the scenario in more detail. The people in the meeting room, the users, have good local connectivity between each other. The Subversion server is typically located in a different country, with low and varying network capacity and higher delay. Without ICN, each user uploads a document, presentation, or similar, to the server, and all other users then download it from the server, resulting in many identical transfers over the network. By using the ICN communication services, Subversion clients automatically fetch updated files in the repository from other nearby clients when possible, and thus the redundant identical transfers over the wide area path to the server are eliminated.

An initial experiment [35] is done by adapting a Subversion client and server to use the network service of the ICN prototypes OpenNetInf [36], from the 4WARD EU project, and CCNx [37], from the CCN project at PARC. These experiments show a clear benefit already for two clients when the connection to the server has some limitations.

As one example result, Figure 5.6 shows the measured performance of a checkout operation using Subversion over OpenNetInf relative to the performance of legacy Subversion over HTTP.

Values below one (1) on the y-axis mean lower performance than legacy Subversion, and above mean higher. The three curves show the performance for different connection bit-rate and delay to the server, simulating different Wide Area Network (WAN) characteristics for the path to the

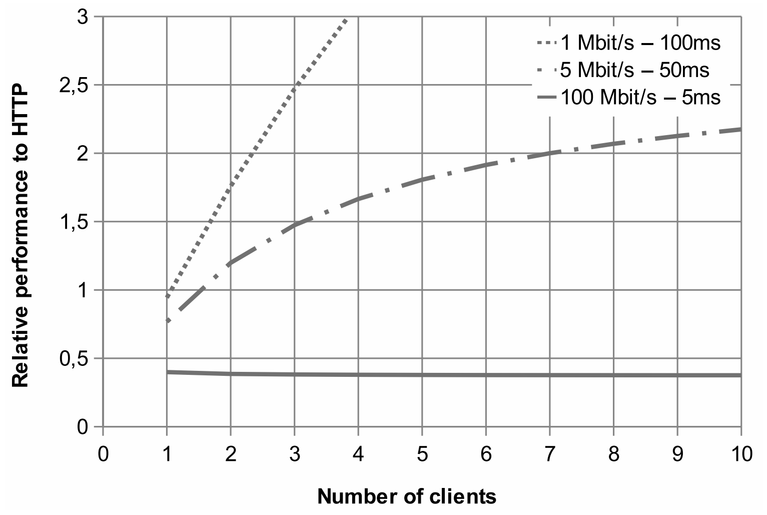

Figure 5.6: Subversion/OpenNetInf performance relative Subversion/HTTP

server. There is a clear performance gain for the 1 and 5 Mbit/s cases already when there are two local clients.

6 Conclusions and Future Work

Information-Centric Network (ICN) as a general concept is one of the promising directions for evolving the Internet to better support today's dominant applications and the associated increase in bandwidth demand. This general concept can be implemented in different ways. The SAIL NetInf approach is centered around a well-defined set of architecture invariants (such as unique naming, location-independence and a strict information-centric service model) and puts particular emphasis on supporting multi-technology/multi-domain interoperability.

Section 6.1 summarises the main results of SAIL NetInf so far. The adopted research approach favors early concept validation and specification adoption based on running code. Prototyping the different NetInf instantiations provides valuable feedback for the NetInf architecture work and allows to experiment with different research ideas at an early phase. The current state of these activities are described in Section 6.2.

In order to enable NetInf to have an impact on the future development of the Internet, it is important to consider standardisation options and to develop migration paths that provide possibilities for introducing ICN in a non-disruptive way. Standardisation and migration activities are described in Section 6.3.

6.1 First Year Achievements

SAIL NetInf is an ICN approach that explicitly caters for supporting diverse application scenarios, networks and corresponding requirements for specific ICN mechanisms. For instance, the developed scenarios range from NetInf as a solution for CDN-style IPTV content distribution to content caching in mobile communication networks and to content dissemination in DTN-type networks.

This heterogeneity needs to be considered for building a viable ICN infrastructure that is evolvable into different directions so that it has realistic chances for adoption as an Internet-scale technology. Moreover, NetInf should not require a flag day transition from the existing Internet, which calls for migration options, i.e., the ability to be gradually introduced to the current Internet infrastructure.

Catering for this heterogeneity is an ambitious approach – it requires NetInf to accept any underlying network technology, such as existing HTTP/TCP/IP, while still being able deployable as native inter-networking protocol, i.e., as an IP replacement.

NetInf is therefore based on a set of architecture invariants that define fundamental concepts and mechanisms that are needed to ensure interoperability between different technological and administrative domains. These invariants (that are described in Section 3.1) helped to shape different instantiations of NetInf and have reached a relatively stable state now.

By these invariants fundamental models of NetInf networks with elements such as unique object naming, concepts of convergence layers and name resolution are developed – an overview is provided in Section 3.2. The current research work has then focused on these different elements to achieve good results on different topics.

For instance, the naming work (Section 4.1) – one of the most fundamental NetInf elements – has led to the specification of an URI-based format for named information that can provide optional features such as secure name-to-content binding, supporting different mechanisms in order to address the heterogeneity requirement. This specification [15] is submitted to the IETF.

NetInf names can be used in different ways in a NetInf network. Fundamentally, they are object identifiers and are used in requests for objects. In some instantiations, they can be resolved to other names, such as locators on different network layers. A Name Resolution Service based on MDHT than can be applied to global name resolution is developed. A detailed description of this system has been published as [19].

Another way of using NetInf names is to use them as keys for name-based routing. Foundations for such approaches in challenged networks, i.e., networks without stable end-to-end paths is worked on, and the BPQ extension for the DTN bundle protocol that allows DTN nodes to carry NetInf requests and responses that may be distributed using arbitrary DTN routing protocol is developed. The corresponding specification has been submitted to the Internet Research Task Force (IRTF) as [31].

A developed transport abstraction provides a common, chunk-based transport across different domains and is able to leverage different underlying network and transport services through a convergence layer concept. The work on receiver-based transport protocols and resource management across domain boundaries is currently ongoing.

This architecture and protocol specification work has been accompanied by ambitious, yet promising prototyping activities (see Section 6.2 for details) that indicates the perceived importance of and commitment to NetInf as a technology – from both the industry and academic partners in the work package.

Moreover, ICN is introduced as a general concept and NetInf technologies in particular to partners and bodies outside of SAIL, for instance in co-organising events such as the Dagstuhl seminar on ICN and the ACM SIGCOMM 2011 workshop on ICN. In addition, various standardisation activities which are summarised in Section 6.3 are established.

6.2 Prototyping

Prototyping is in the centre of SAIL's NetInf work providing support for architecture, use case and evaluation work. Therefore considerable amount of effort is and will be put in to it. While it may not be possible to have working prototype and/or demonstrator for each NetInf application and use case, the selected areas in the NetInf prototyping activities are covered. The NetInf prototyping approach follows the guidelines and experimentation framework defined in [14]. The prototypes can be roughly divided into two categories; i) proof-of-concept prototypes and ii) application prototypes. The former category focuses on validating NetInf architecture elements whereas the latter shows what kind of benefits the use of NetInf architecture invariants and techniques can provide in the selected use cases described in Chapter 5. The following sections present the selected prototyping categories and describe the prototyping plans and future steps and show how the overall contribution maps into the other work items in the SAIL NetInf. More detailed information will be provided in the forthcoming deliverables.

6.2.1 Proof-of-Concept Prototypes

In this section the proof-of-concept prototyping activities validating NetInf architecture elements and techniques are described.

Name-based resolution and routing in ICN environment

This proof-of-concept prototype is mainly aimed to assess the feasibility of an information-centric network based on the GIN approach [38]. The prototype will adopt an integrated approach for the name-based resolution and routing. In particular, the name resolution system will be based on the MDHT [19] technical approach. It will work in conjunction with a stateless packet delivery protocol, namely the GIN Protocol (GINP), that will manage the name-based forwarding of data units within the network. Some of the main functionalities that the prototype will hopefully allow to demonstrate are the registration of object/host IDs into the network resolution system and their retrieval by IDs, the name-based forwarding of data over the shortest path and the locality features (i.e., content locality, resolution locality and routing locality) supported by the MDHT architecture. The prototype will be implemented on virtual network environment.

OpenNetInf

The open source OpenNetInf prototype [36] is building on and extending earlier work from the 4WARD project[9]. The OpenNetInf implementation is a proof-of-concept implementation of the major NetInf elements, including the NetInf API, inter-NetInf-node interface, information model, naming concepts, security concepts, name resolution, caching, and data transfer. The goal is to evaluate the major NetInf design decisions and the overall NetInf architecture in practice. Currently, a focus is on developing and implementing a hierarchical name resolution system (MDHT-based) and on integrated caching functionality. Another focus is on the NetInf API and on the inter-NetInf-node interface. The prototype will be tested using a video distribution use case.

6.2.2 Application Prototypes

This section describes the prototyping activities implementing the selected application scenarios (see Section 5.1-Section 5.5) to show NetInf benefits.

Active information objects

To optimise the user experience, content and services should be provided from servers close to the users. In networks where users are mobile the content and service placement should be more dynamic and adapt to the variations of the geographical location of users and service popularity. Therefore, the objective of this prototype is to demonstrate the benefits of active (information) object/service migration. The scenario covered in this prototype will support video distribution from locations close to mobile users. More specifically, when the user asks for the service, a point in the network close to the user has to be selected and the service delivered to the user from that point. The service in this prototype is merging two input video data streams such as a movie and local weather information into one resulting stream that is delivered to the user. The basic functionalities to be demonstrated are how to effectively migrate service modules among network nodes, i.e., what technologies are needed for that, and how to detect the need for that in the network. Needless to say, the prototype will follow the NetInf centric communication paradigm. In addition, this prototype also connects cloud networking with information-centric networking and covers the Elastic Video Distribution use case that is addressed by the cloud networking work package (CloNe). This prototype will serve as a proof of the service migration concept.

Flash/Large crowd

In order to enhance user experience in situations with an unexpectedly large amount of users in a relatively small area (flash crowd), the clients in the crowd should cooperate in order not to saturate the available uplink and still provide access to the most popular objects. In order to demonstrate the benefits of NetInf in such a scenario, two prototypes are planned for the Section 5.2 scenario. The first prototype is heavily based on the event large crowd application described in Section 5.2. The plan is to use HTTP RESTful as the preferred interface for communication between modules. Json is used for data interchange. The plan is to have well defined and documented interfaces to enable others to interact remotely with the prototype. A basic rendezvous server and some basic client functionality such as local caching, rendezvous server update and local Name resolution server has been implemented so far. The second prototype under development is based on the global architecture of SAIL NetInf, with the large crowd scenario (see Section 5.2) as main target. The structure of the router is modular, easily allowing its functionalities to be extended. Modules can be implemented as shared libraries or as external processes communicating with the core using a non-HTTP RESTful protocol. Once the protocol is stable enough it will be available for others to implement their own modules. Headers will be provided for the shared library approach. A basic node with local caching, simple static name resolution, and HTTP access has been implemented so far. The testbed for the target scenario will consist of Access Points and clients (e.g., laptops and smartphones) running the NetInf prototype.

Localised CDN

This prototype is based on the rendezvous network implementation from the PSIRP project to demonstrate proactive caching and further on localised CDN NetInf application at least for video distribution use case. The application scenario is described in Section 5.1. During the first year existing prototype has gone through small re-factoring and as an additional feature initial NRS function has been implemented. Moreover Apache based open source Traffic Server (TS) software has been integrated to adopt the role of localised CDN cache. Next steps in the prototype work include further fine tuning of the existing features, replacement of the initial NRS API with the NetInf API based on the OpenNetInf sources and implementation of the demonstration network consisting around 10 network nodes running in virtual network environment.

6.3 Standardisation and Migration

ICN in general and NetInf in particular are approaches that differ significantly from current network designs. With the current Internet as a crucial infrastructure for today's societies and businesses, it is unfeasible to proclaim a flag day when the world will transition to NetInf. Instead it can be expected that NetInf has to co-exist with the existing Internet, the web, and P2P networks for a while in order to (eventually) gradually evolve into a dominant communication service.

This calls for migration possibilities: for example content that is available in the web today must still be accessible from NetInf. Ideally, NetInf names would also be usable as URIs by existing applications to not require users to upgrade all applications immediately. Similar requirements can be identified for other NetInf elements such as NetInf Transport Services, security mechanisms etc, where the key to successful adoption is not only a sound technical design but also the consideration of **migration and interworking requirements**.

Moreover, upgrading the global inter-networking infrastructure will affect many stakeholders' interests, include those of governments, regulators, vendors, network operators, service providers and, most importantly, the actual users. The development of the Internet has shown that this requires a balance of interests and agreements for achieving interoperability – which is typically done by **standardising key components**. The identification of these components and a strategy towards agreeing on particular approach in a standardisation process is thus another important element that needs to be considered to ensure successful evolution and deployment of NetInf's technology.

In this section, there is a discussion about approaches for migrating to NetInf and for arriving at a globally acceptable and deployable solution, on an Internet level. Migration and standardisation are considered together, because they are both considered to be activities that are related to Internet evolution. Also, the expectation is that the same key components that are eligible for standardisation are also important as migration enablers.

One such component is certainly **naming of information objects** – a key NetInf architecture element that is described in Section 4.1. From a NetInf design perspective, it is desirable to have one uniform concept and format to identify and access all possible named objects – independent of specific access technologies and policies. From a standardisation perspective, it is beneficial to de-couple the naming system from technology- and application domain-specific constraints to ensure a broad applicability. Also, assuming there would be different specific ICN approaches, a named object should have a name that, in principle, makes that object accessible via any ICN protocol, not just (variants of) NetInf to avoid fragmentation. From a migration perspective, NetInf names should be usable to identify existing resources in the web, P2P networks etc. Moreover, NetInf names should be compatible with names commonly used by applications today, i.e., URIs.

With the NetInf naming scheme and the URI format for Named Information (NI) [15], an approach that fulfils these requirements is proposed. Further, this format specification have been submitted as an Internet Draft, providing a uniform way to identify resources in different information/content-centric scenarios, with different requirements for name verification, object transport etc.

In addition to genuine ICN approaches, the NI format is intended to be useful for unique naming needs in existing and emerging applications such as P2P and network storage applications, which are current IETF standardisation topics. Applying the NI format to those applications could thus contribute to enabling interworking with and migrating to NetInf (in addition to being useful by itself).

Furthermore, there was a more specific contribution made on security properties of object names (”Secure naming structure and P2P application interaction” [39]), describing security objectives, corresponding concepts and possible applications to P2P streaming, in-network storage systems (Decoupled Application Data Enroute (DECADE)), and CDNs. The considerations presented in this Internet Draft exhibit common possible requirements on security properties of object names and thus provide additional thrust for the planned migration efforts.

When implementing NetInf and other ICN-based systems, one important aspect is the design of caches (which are described conceptually in Section 4.7). Since many applications can benefit from in-network caching, including CDNs, P2P-based distribution and streaming, and ICNs, it is beneficial to enable them to use common storage platforms – for both interworking and efficiency reasons. The IETF DECADE working group [40] is standardising access to in-network storage, and there was a contribution to that effort with a requirements document [41] and an architecture specification [42] that addresses the above-mentioned goals.

Another component that is appropriate for migration and standardisation considerations would be **name resolution** (leveraging the existing DNS infrastructure and its on-going evolution). NetInf application design and API considerations are introduced in Chapter 5 and Section 4.9, and one migration challenge is enabling unchanged legacy applications to use of NetInf networks. Legacy applications would typically utilise a DNS lookup stage to obtain a locator for accessing data. This DNS lookup can be directed into a NetInf resolution step. The result is that legacy applications would typically require a two-step approach: first a lookup stage to locate the location of the data, second to retrieve the data via a transport protocol. Thus migration is best supported if legacy application can utilise the two-step approach such that the NetInf network can hide the change of functionality (the DNS lookup and data retrieval) into NetInf native activities. However the goal of the NetInf network is to offer native NetInf functionality through NetInf APIs.

Finally **NetInf transport** is another addressed area. For communication in challenged networks, the Delay-Tolerant Networking Research Group (DTNRG) has developed the DTN architecture, the Bundle protocol and several convergence layer specifications that can be employed for NetInf transport in challenged networking scenarios. A specification of a the BPQ extension block [31] have been developed that describes how requests for NetInf objects (and corresponding replies) can be transmitted in DTNs. This approach is a particularly interesting way of migrating from a NetInf-unware DTN to an information-centric one, since the mechanism does not require a conversion of all DTN node, but rather allows for a gradual introduction of NetInf nodes, while still being able to leverage genuine DTN routing and bundle transport mechanisms.

Besides technical considerations, there are also socio-economic factors that need to be considered to ensure a successful development and deployment of NetInf technology. Specifically, the introduction of new information-centric paradigms can imply changes in business relationships and lead to new regulatory requirements. The incentive and business aspects of NetInf has been studied in the SAIL Impact & Collaboration Enabling (ICE) starting with use case scenarios ([13]) and a closer investigation of roles and business models with the new NetInf functions. [43] has content distribution cases (several CDN business cases) looked at in more detail.

Complementary to these different efforts on the system component level, there is potential in addressing ICN standardisation in a more comprehensive fashion, i.e., at system level, specifying ICN-specific functions and overall system architectures. This is an on-going effort in collaboration with other activities in the area. One potential cooperation instance is under development within the IRTF where the formation of an ICN research group is under discussion

List of Abbreviations, Acronyms, and Definitions

ABNF	Augmented Backus-Naur Form
ADU	Application Data Unit
API	Application Programming Interface
ASN	Autonomous System Number
AS	Autonomous System
BGP	Border Gateway Protocol
BO	Bit-level Object
BPQ	Bundle Protocol Query
BP	Bundle Protocol
CANON	Canonical Chord
CA	Custody Agent
CCN	Content Centric Networking
CDNI	Content Delivery Network Interconnection
CDN	Content Delivery Network
CID	Compact Identifier
CL	Convergence Layer
CMS	Cryptographic Message Syntax
DANE	DNS-Based Authentication of Named Entities
DECADE	Decoupled Application Data Enroute
DHT	Distributed Hash Table
DIS	Destination ID Stack
DNS	Domain Name System
DONA	Data-Oriented Network Architecture
DTNRG	Delay-Tolerant Networking Research Group
DTN	Delay-Tolerant Networking
DoS	Denial of Service

EID	Endpoint Identifier
FIND	Future Internet Design
FQDN	Fully Qualified Domain Name
GIN	Global Information Network
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICE	Impact & Collaboration Enabling
ICN	Information-Centric Networking
ID	IDentifier
IETF	Internet Engineering Task Force
IO	Information Object
IRTF	Internet Research Task Force
ISP	Internet Service Provider
LLC	Late Locator Construction
LNA	Layered Naming Architecture
LNB	Late Name Binding
LTE	Long Term Evolution
MDHT	Multilevel DHT
MIME	Multipurpose Internet Mail Extension
MSID	Multicast Session ID
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NHT	Next Hop Table
NI	Named Information
NRS	Name Resolution System
NTS	NetInf Transport Services
NbR	Name-based Routing
NetInf	Network of Information
P2P	Peer-to-Peer

PDU	Protocol Data Unit
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PPSP	Peer to Peer Streaming Protocol
PSIRP	Publish-Subscribe Internetworking Routing Paradigm
QoE	Quality of Experience
RCP	Rate Control Protocol
REST	Representational State Transfer
REX	Resolution EXchange
RFC	Request For Comments
RFID	Radio Frequency ID
RID	Rendezvous ID
RPKI	Resource PKI
RTT	Round Trip Time
RVS	RendezVous Server
SAIL	Scalable and Adaptive Internet Solutions
SBB	Setup Before Break
SID	Scope ID
SIP	Session Initiation Protocol
SIS	Source ID Stack
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TS	Traffic Server
TTL	Time To Live
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VPN	Virtual Private Network
VoIP	Voice over IP
WAN	Wide Area Network

WG	Working Group
WiFi	Wireless Fidelity
XKMS	XML Key Management Specification

List of Figures

List of Tables

Bibliography

[1] The SAIL project web site. `http://www.sail-project.eu/`.

[2] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Börje Ohlman. A Survey of Information-Centric Networking (Draft). In Bengt Ahlgren, Holger Karl, Dirk Kutscher, Börje Ohlman, Sara Oueslati, and Ignacio Solis, editors, *Information-Centric Networking*, number 10492 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2011. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[3] Christian Dannewitz, Jovan Golic, Börje Ohlman, and Bengt Ahlgren. Secure naming for a network of information. In *Proc. 13th IEEE Global Internet Symposium 2010*, San Diego, USA, March 2010.

[4] The CCN project web site. `http://www.parc.com/publication/2318/networking-named-content.html`.

[5] Koponen T., Chawla M., Chun B.-G., Ermolinskiy A., Kim K. H., Shenker S., and Stoica I. A data-oriented (and beyond) network architecture. In *SIGCOMM*, 2007.

[6] Paul S. Postcards from the edge: A cache-and-forward architecture for the future internet. In *NeXtworking07*, 2007.

[7] The FIND initiative web site. `http://www.nets-find.net/`.

[8] Balakrishnan H., Lakshminarayanan K., Ratnasamy S., Shenker S., Stoica I., and Walfish M. A layered naming architecture for the internet. In *SIGCOMM*, 2004.

[9] The 4WARD project web site. `http://www.4ward-project.eu/`.

[10] Bengt Ahlgren, Matteo D'Ambrosio, Christian Dannewitz, Anders Eriksson, Jovan Golić, Björn Grönvall, Daniel Horne, Anders Lindgren, Olli Mämmelä, Marco Marchisio, Jukka Mäkelä, Septimiu Nechifor, Börje Ohlman, Sabine Randriamasy, Teemu Rautio, Eric Renault, Pasi Seittenranta, Ove Strandberg, Bogdan Tarnauca, Vinicio Vercellone, and Djamal Zeghlache. Netinf evaluation. Deliverable D-6.3, 4WARD EU FP7 Project, June 2010. FP7-ICT-2007-1-216041-4WARD / D-6.3, http://www.4ward-project.eu/.

[11] M. Pathan and R. Buyya. A taxonomy and survey of content delivery networks. In *Technical Report, GRIDS-TR-2007-4, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia*, February 2007.

[12] The Content Delivery Networks Interconnection (CDNI) WG Draft Charter. `http://trac.tools.ietf.org/bof/trac/attachment/wiki/WikiStart/CDNI%20Draft%20Charter%20.txt`.

[13] Thomas Edwall et al. Description of project wide scenarios and use cases. Deliverable D-A.1, SAIL project, February 2011. available online from http://www.sail-project.eu.

[14] Benoit Tremblay et al. Architecture Guidelines and Principles. Deliverable D-A.2, SAIL project, July 2011. available online from http://www.sail-project.eu.

[15] Stephen Farrell, Christian Dannewitz, Börje Ohlman, and Dirk Kutscher. Uris for named information. IETF Internet-Draft draft-farrell-ni, March 2011. version 00.

[16] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005.

[17] S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648 (Proposed Standard), October 2006.

[18] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045 (Draft Standard), November 1996. Updated by RFCs 2184, 2231, 5335.

[19] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone. MDHT: A Hierarchical Name Resolution Service for Information-centric Networks. In *ACM SIGCOMM Workshop on Information-Centric Networking (ICN 2011)*, Ottawa, Canada, August 2011.

[20] Matteo D'Ambrosio, Paolo Fasano, Marco Marchisio, Vinicio Vercellone, and Mario Ullio. Providing data dissemination services in the future Internet. In *Proc. World Telecommunications Congress (WTC'08)*, New Orleans, LA, USA, December 1-2, 2008. At IEEE Globecom 2008.

[21] Michael K. Bergman. The deep web: Surfacing hidden value. *Journal of Electronic Publishing In TAKING LICENSE: Recognizing a Need to Change*, 7(1), August 2001.

[22] J. Rajahalme, M. Srel, K. Visala, and J. Riihijarvi. In *On name-based inter-domain routing*, volume 55, pages 975–986, 2011.

[23] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050 (Experimental), November 2007.

[24] N. Dukkipati, M. Kobayashi, R. Zhang-shen, and N. Mckeown. Processor sharing flows in the internet. In *13th International Workshop on Quality of Service (IWQoS)*, 2005.

[25] M. Proebster, M. Scharf, and S Hauger. Performance comparison of router assisted congestion control protocols: Xcp vs. rcp. In *Proceedings of the 2nd International Workshop on the Evaluation of Quality of Service through Simulation in the Future Internet - SIMUTools 2009*, March 2009.

[26] P. Rodriguez, C. Spanner, and E. Biersack. Analysis of web caching architectures: Hierarchal and distributed caching. In *IEEE/ACM Transactions on Networking*, 2007.

[27] P. Cao and S. Irani, 1997.

[28] M. R. Korupolu and M. Dahlin. In *Coordinated placement and replacement for large-scale distributed caches*, volume 14, pages 1317–1329, 2002.

[29] L. Ramaswamy and L. Liu. In *A New Document Placement Scheme for Cooperative Caching on the Internet*, 2002.

[30] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838 (Informational), April 2007.

[31] S. Farrell et al. In *Bundle Protocol Query Extension Block*, November 2010.

[32] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[33] S. Farrell, R. Housley, and S. Turner. An Internet Attribute Certificate Profile for Authorization. RFC 5755 (Proposed Standard), January 2010.

[34] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021.

[35] Bengt Ahlgren, Börje Ohlman, Erik Axelsson, and Lars Brown. Experiments with subversion over opennetinf and ccnx. In *7th Swedish National Computer Networking Workshop (SNCNW)*, Linköping, Sweden, June 13-14, 2011.

[36] The OpenNetInf web site. `http://www.netinf.org/`.

[37] The CCNx web site. `http://www.ccnx.org/`.

[38] Matteo D Ambrosio, Paolo Fasano, Marco Marchisio, Vinicio Vercellone, and Mario Ullio. Providing data dissemination services in the future internet. In *Proceedings of the Global Communications Conference, 2008. GLOBECOM 2008, New Orleans, LA, USA, 30 November - 4 December 2008*, pages 5606–5611. IEEE, 2008.

[39] Christian Dannewitz, Teemu Rautio, Ove Strandberg, and Börje Ohlman. Secure naming structure and p2p application interaction. IETF Internet-Draft draft-dannewitz-ppsp-secure-naming, March 2011. version 02.

[40] Decoupled Application Data Enroute (decade) WG Charter. `https://datatracker.ietf.org/wg/decade/charter/`.

[41] Börje Ohlman, Ove Strandberg, Christian Dannewitz, Anders Lindgren, Roberta Maglione, Bengt Ahlgren, and Dirk Kutscher. Requirements for accessing data in network storage. IETF Internet-Draft draft-ohlman-decade-add-use-cases-reqs, October 2010. version 02.

[42] Richard Alimi, Y. Richard Yang, Akbar Rahman, Dirk Kutscher, and Hongqiang Liu. Decade architecture. IETF Internet-Draft draft-ietf-decade-arch, May 2011. version 01.

[43] Jukka Salo et al. New Business Models and business dynamics of the future networks. Deliverable D-A.7, SAIL project, July 2011. available online from http://www.sail-project.eu.