Objective FP7-ICT-2009-5-257448/D-3.2

Future Networks

Project 257448

"SAIL – Scalable and Adaptable Internet Solutions"

D-3.2
(D.B.2) NetInf Content Delivery and Operations

Date of preparation: **12-05-07**
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-3.2
Document Title:	**NetInf Content Delivery and Operations**
Document Responsible:	Dirk Kutscher (NEC)
Document Editor:	Gerald Kunzmann (DOCOMO), Dirk Staehle (DOCOMO)
Authors:	Bengt Ahlgren (SICS), Matteo D. Ambrosio (TI), Elwyn Davies (TCD), Anders E. Eriksson (EAB), Stephen Farrell (TCD), Björn Grönvall (SICS), Claudio Imbrenda (NEC), Bruno Kauffmann (FT), Gerald Kunzmann (DOCOMO), Dirk Kutscher (NEC), Anders Lindgren (SICS), Ian Marsh (SICS), Luca Muscariello (FT), Börje Ohlman (EAB), Karl-Ake Persson (EAB), Petteri Pöyhönen (NSN), Mohammed Shehada (DOCOMO), Dirk Staehle (DOCOMO), Ove Strandberg (NSN), Janne Tuononen (NSN), Vinicio Vercellone (TI)
Target Dissemination Level:	PU
Status of the Document:	Final Version
Version:	1.0

Production Properties:

Reviewers:	Pedro Aranda (TID), Björn Levin (SICS), Benoit C. Tremblay (EAB)

Document History:

Revision	Date	Issued by	Description
1.0	2012-05-07	Dirk Staehle	Final Version

Disclaimer:

This document has been produced in the context of the SAIL Project. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2010–2013) under grant agreement n° 257448.

Abstract:

This deliverable on "NetInf Content Delivery and Operations" reports on the results of the NetInf development work on content delivery, applications, and operations/management. It describes the NetInf Protocol and key components of the NetInf architecture such as an object model, a naming scheme, the NetInf Convergence Layer approach and several specific NetInf Convergence Layers that have been developed. This document also presents the NetInf approach to global connectivity and discusses operational considerations for the NetInf technology in specific deployments. It also provides an evaluation of several mechanisms for transport, caching and management and a conceptual description of service placement in NetInf networks, together with a service placement optimization approach. The NetInf "Event with Large Crowd" demonstration scenario integrates the different technology components and corresponding SAIL partner developments and provides a basis for evaluation.

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]. It describes content delivery and operational aspects of an information-centric network (ICN) based on the *Network of Information* (NetInf) architecture and protocol. NetInf, or in general Information-Centric Networking (ICN), is 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 NetInf approach is aiming at a highly scalable network architecture with particular support for robustness and reliability as well as at multi-technology/multi-domain interoperability. SAIL NetInf is putting particular emphasis on enabling networks that go beyond current de-facto architectures for broadband/mobile access and data centre networks.

An earlier deliverable has described the NetInf Architecture and Applications. Therefore this document has three main contributions: 1) the evolved NetInf architecture and protocol foundations; 2) the description and evaluation of specific techniques for content distribution and management of ICNs and 3) a non-trivial demonstration scenario that illustrates how the NetInf protocol components can be put together in a network.

The main achievement for the architecture and protocol work was the development of the *NetInf Protocol* – a conceptual specification of a NetInf Node-to-Node communication protocol that includes an object model for Named Data Objects (NDOs) and a detailed description of the Convergence Layer approach that we introduced earlier. Based on these specifications, SAIL partners have started different prototyping developments, from proprietary NetInf router implementations to applications and Open Source protocol implementations such as the NetInf software project that SAIL released in a multi-partner effort in March 2012.

This document reports on results of our current work on routing, transport and caching mechanisms which is using these prototyping foundations. One important aspect is how to *manage* real-world NetInf networks – and what are the specific network management requirements for NetInf (or ICN in general) compared to traditional networks. For that we have investigated management approaches comprising both traditional management elements with ICN extensions (e.g., for content placement) and new techniques for leveraging NetInf meta data for self-organised management.

Finally, SAIL has developed an *Event with Large Crowd* demonstration scenario for validating the NetInf architecture, protocol and specific content distribution and management techniques. This work is unique in two ways: 1) it involves the definition of a non-trivial real-world application scenario that requires many of the NetInf protocols and algorithms that have been presented in this document; and 2) it is actually being implemented in a significant integration effort of different partner components. The details as well the current implementation status are also described in this document.

Contents

1 Introduction

Information-Centric Networking (ICN) is a promising approach for evolving the Internet towards an infrastructure that can provide an optimal service for accessing named data objects – one of the dominant applications today. In general, ICN is providing access to named data objects as a first class networking primitive and is leveraging unique naming techniques and ubiquitous in-network caching to provide more efficient and robust networking services than current approaches allow.

The Scalable and Adaptive Internet Solutions (SAIL) project has been developing the *Network of Information* (NetInf) approach that is aiming at a highly scalable network architecture with particular support for robustness and reliability as well as at multi-technology/multi-domain interoperability. SAIL NetInf is putting particular emphasis on enabling networks that go beyond current de-facto architectures for broadband/mobile access and data center networks. While we want to support those deployment scenarios and their corresponding business requirements, we also want networks to go beyond inherited telco constraints and assumptions.

For example, ICN can be made to work with the existing network infrastructure, name resolution and security infrastructure – but that does not mean that all ICN networks should depend on such infrastructure. Instead, we want to leverage local, de-centralised communication options to arrive at a solution that is easy to deploy at small scale and is able to extend to global scale but still resilient against network partitions, intermittent connectivity and potentially longer communication delays.

Likewise, ICN is often characterised as a generalised content distribution approach, but in fact, has benefits beyond content distribution – for example, better security properties through Named Data Object (NDO) security as well as better performance and robustness through in-network caching and localised transport strategies.

We believe that NetInf's *going beyond next-generation CDN* approach will finally result in a network that better accommodates current mass-market applications (for example for content distribution) *and* future mass-market applications such as smart-object communications in constrained networks.

ICN is currently one of the fastest developing topics in networking research and many proposals and active discussions are advancing the research. For example, after the publication of our previous SAIL Deliverable D.B.1 "The Network of Information: Architecture and Applications" [2], there has been an ACM SIGCOMM workshop on Information-Centric Networking, an IEEE Infocom workshop on Emerging Design Choices in Name-Oriented Networking and related events. More such events are planned. Moreover, the Internet Research Task Force (IRTF) has chartered a research group on ICN that will start continuous activities on analysing ICN design options and on comparing different solutions.

The first project deliverable D.B.1 described NetInf in terms of an architecture framework (based on a set of invariants) and architecture elements for naming, name resolution, search, routing and forwarding, mobility, transport, caching, security, and APIs, which has been illustrated by a set of application scenarios. It made concrete decisions on key topics for interoperability such as naming and experimentation options for routing and forwarding. We have since done prototyping and experimental activities for assessing design choices and for further specifying details of the NetInf systems.

The results of this work are documented in this deliverable. In terms of architecture invariants, we have been able to formulate things more specifically, as we have been able to substantiate some proposals and to constrain the design space, for example for routing and forwarding as well as

for NetInf node requirements. One notable change, for example, is the transition from the term Information Object (IO) to the term NDO that better reflects the absence of semantics for NetInf objects. The detailed changes are documented in Section 2.1.

The main achievement to this extent was the development of the *NetInf Protocol* – a conceptual specification of a NetInf Node-to-Node communication protocol that includes an object model for NDOs and a detailed description of the Convergence Layer approach that we introduced in D.B.1 (see Section 2.1 and Appendix A for details).

The NetInf protocol work was driven by the objective to build systems that actually work in a variety of scenarios, and for that we have followed a prototyping-driven approach. This led to a set of additional specifications such as the ni: naming format [3, 4] and different Convergence Layer specifications. Based on these specifications, SAIL partners have started different prototyping developments, from proprietary NetInf router implementations to applications and Open Source protocol implementations such as the NetInf software project that SAIL released[1] in a multi-partner effort in March 2012.

Our current work on routing, transport and caching mechanisms is leveraging this work. One important aspect it how to *manage* real-world NetInf networks – and what are the specific network management requirements for NetInf (or ICN in general) compared to traditional networks. For NetInf, *network management* has two facets: 1) the more traditional management, extended to ICN-specifics such as cache and name resolution service management; and 2) more innovative approaches that leverage the NetInf model of having *accessing named data* and the corresponding object delivery as first class networking primitives. This enables the network to literally see interest in object and other helpful information so that it can react to this information in a self-organised fashion. We are describing a set of techniques for transport, (collaborative) caching and corresponding network management in this document.

Finally, it is important to validate research ideas and analytic results by experiments with real implementations, i.e., running code. We are illustrating possible ways for integrating these components into larger systems by developing a prototyping and validation application scenario that focuses on scalable content generation and distribution in large user crowds.

This deliverable is structured as follows: Chapter 2 documents the technical basis for the NetInf research work, provides the updates to the architecture framework and describes recent results such as the NetInf protocol and our approach to global connectivity. Chapter 3 is focusing on specific mechanisms that leverage the NetInf architecture and protocols and that are needed for building systems such as content transport and placement strategies. This work includes performance studies for caching that provides interesting insights for the deployment of NetInf networks. In Chapter 4 we explore ideas for leveraging NetInf for service provisioning, i.e., going beyond pure content distribution. This work includes a definition of services in ICN and a cost analysis for service placement. We illustrate the integration of our NetInf protocols and algorithms into a non-trivial application scenario in Chapter 5, where we present the design and implementation plan of the NetInf *Event With Large Crowd Scenario*. Finally, Chapter 6 concludes this deliverable.

[1] http://sourceforge.net/projects/netinf/

2 NetInf Protocol and Architecture

NetInf is aiming at a highly scalable network architecture with particular support for robustness and reliability as well as at multi-technology/multi-domain interoperability. In this section, we present an update on the NetInf Architecture Invariants in Section 2.1 that are the cornerstones for the NetInf Protocol that we describe in Section 2.2. Section 2.3 presents the NetInf approach to global, i.e., globally scalable, NetInf networks, and Section 2.4 illustrates those concepts by describing operational considerations for these concepts. Section 2.5 summarises the main insights.

2.1 Architecture Invariants

The invariants of the NetInf architecture were described in the previous SAIL deliverable D.B.1 [2]. Based on recent architecture work, we have made some modifications of the invariants for routing, and forwarding. These modifications are due to the Convergence Layer approach described in Section 2.2.4.1. The other invariants remain the same, except for a terminology change, where "IO" has been replaced with "NDO". We have moved to using the term NDO to emphasise the coupling of the actual data of the IO and the unique name used to identify and access the data using the NetInf protocol. In this section we describe the complete set of the updated NetInf invariants.

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 between NetInf nodes based on NDO names.
- NDO names are independent of the location of the object in the network topology.
- A multi-domain NetInf network may include domains with challenged networks, such as Delay-Tolerant Networking (DTN) networks.

Name Resolution & Routing: NetInf may perform routing based on the names of NDOs. However, routing based on flat names from NDOs may not scale in a global NetInf network. Therefore, the global NetInf network may use an Name Resolution System (NRS) to map these names to locators that typically will identify physical entities 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. 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 NDOs. The underlying network can be any interconnection of heterogeneous L2 or L3 subnetworks. There is (at least) one NRS for the global NetInf network (inter-domain name resolution). There may be name resolution systems which are local to a domain or a host (intra-domain name resolution).

Support for Challenged Networks: In some cases (e.g., in a challenged network domain) it is not possible to resolve NDO 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: In order both to improve scalability and to cope with situations where a NetInf layer message has to be forwarded across network domain boundaries, NetInf messages will often be forwarded 'incrementally' through several NetInf hops, where a NetInf hop might be made up of several hops in the underlying network. By using identifiers or locators that refer to network structures at various scales, aggregation can be used to control the amount of state needed for NetInf forwarding. Messages are initially directed to gateways in large scale structures which can incrementally refine the direction of the message.

Mobility and Multihoming: In a global NetInf network, mobility and multihoming is based on dynamic updates of the bindings in the NRS between the NDO names and the identifiers or locators used for forwarding. 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 NDOs.

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 Transmission Control Protocol (TCP)/IP networks, Ethernet, etc.

API: The NetInf Application Programming Interface (API) is NDO-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, NDOs are addressed directly by their names.

2.2 NetInf Protocol

The NetInf (Network of Information) Protocol developed in the SAIL project has been designed and prototyped to provide a means for applications to locate, publish and retrieve information objects in which they have an interest in an ICN context. The protocol is built around the fundamental idea of information objects that are uniquely identified by a location-independent name that is cryptographically derived from the content of the object, providing a form of "name-data integrity". The term **named data object** (NDO) is used to refer to the combination of the information object and its unique name.

The NetInf Protocol has so far been specified abstractly rather than as a concrete specification for "bits on the wire". This approach has been taken in order to allow experimentation to help determine how best to refine the protocol definition. The definition will be published as an IRTF Internet Draft before the end of the SAIL project. The current draft text can be found in Appendix A.

2.2.1 Naming and Scoping

The location-independent names for NDOs are the key to the operation of the NetInf protocol. They are used by the protocol implementations at network nodes to forward requests and may ultimately be resolved into technology-specific locators and interface identifiers used by the underlying transport technology.

There are different algorithmic approaches for ensuring uniqueness of names. Moreover, details may change over time, for example as network size grows, cryptographic algorithms evolve, etc.

The naming scheme used by the NetInf protocol is intended to be adaptable and extensible so that it can continue to be useful as the details change.

2.2.1.1 Name Format

Considering these requirements, NetInf naming has to allow for a certain degree of flexibility, similar to resource identification in the web today. We therefore claim that NetInf naming should leverage some Uniform Resource Identifier (URI) concepts and have specifically developed the Named Information (ni:) URI scheme that is specified in the Internet Drafts [3] and [4].

The basic ni: scheme provides a flat space of cryptographically generated identifiers represented in a dialect of base64 textual encoding (base64url) and is not intended to be human-readable. The scheme provides a representation that uses only printable ASCII characters in a URI-compatible format that does not require escape characters. This is a compromise between compactness and URI-compatibility.

In addition, recent work with the IETF CORE WG has also resulted in the addition of alternative formats for names, including a compact binary format suited for use in the CoAP protocol [5] and one that is intended to be readily and unambiguously communicated verbally.

This last form uses another new URI scheme (called "nih" for Named Information for Humans), and uses only one case of ASCII-HEX with a checksum to facilitate unambiguous transmission, especially by humans. Note that the nih: scheme is not intended for common uses, but rather for exceptional cases where a name based on a hash function output has to be entered manually or spoken about, e.g. on a phone call. The names themselves are still essentially not human-understandable. It is also worth noting that regardless of the format, two names refer to the same NDO when their hash values are equal.

All of this implies that ni: URIs and variants do not provide a human-friendly, mnemonic name space. Accordingly implementations and applications will need to provide ways in which to map between human-friendly strings (or images!) and ni: names.

There are many ways for this mapping from human-understandable representation to ni: names to be performed and a concrete information-centric network should therefore not force any one mapping scheme on all applications. The specification of the ni: format leaves the "query string" portion of the URI scheme available for any purposes that applications may wish and this could be used to incorporate human-readable information if desired. To potentially assist with routing and forwarding purposes ni: names support an optional authority component and the "query string" can also be used for various routing purposes (see below). Since the authority part is optional, all NetInf nodes need to be able to operate with names that are fully-flat, however, we also envisage that many useful names will have a non-empty authority, e.g., a name that is registered in Domain Name System (DNS).

The cryptographic hash functions used in naming do not, in general, rule out name collisions where two different objects may have the same hash value and, hence, potentially the same name. While this event has a negligible probability for a good hash function, there may be machine-to-machine applications that wish to use truncated hashes, for example to save network bandwidth or storage space on low-end hardware that will allow deliberate or accidental collisions.

2.2.1.2 Scoping

While ni: URIs can include an authority part, this may or may not turn out to be sufficient to support global routing or name resolution based on these names. In order to handle the possibility that additional tags of some sort are needed, we consider adding "scopes" to names and using those to assist routing and name resolution within the network. The scoping concept developed in PSIRP/PURSUIT will here be taken into account as described in [6].

For example, we expect research experiments implementing the ni: format to investigate the use of such scopes or routing hints or labels. Experimental results should provide guidance as to how to represent such additional data and how to handle it as part of implementation designs.

2.2.2 Object Model

ICN applications with broad applicability will require some common agreement on how to structure named data objects (NDOs). For example, some name-data integrity validation approaches require additional cryptographic material (signatures, keys, certificates) to be attached to the actual object that is distributed and stored in a network. Moreover, objects may have some sub-structure, including object fragments, application-specific metadata, etc. Network entities and receiving applications should understand such object structures in order to find the required pieces or for other purposes.

Similar services are required for Internet Mail and other applications today, using Multipurpose Internet Mail Extension (MIME) both as a concept and for the concrete format, enabling many services including multi-part messages, message (content) type identification and security services (CMS).

There are good reasons to adopt this approach for NetInf:

- MIME exists, providing many registered types and a well maintained registry;
- application developers are used to and have code for dealing with MIME objects;
- MIME provides a flexible system that can incorporate multiple named and typed components in a single message without necessarily predefining the object format;
- MIME allows the flexibility to modify objects without losing information both at the application layer and, in principle, within the network and so offers a good method for, e.g., adding metadata to objects; and finally, but not unimportantly
- MIME has well-defined ways in which to robustly calculate cryptographic checksums of objects resilient to, e.g., transfer-encoding changes and also has well-defined digital signature schemes (S/MIME and OpenPGP) should those be warranted. MIME objects can also provide useful information (e.g., Content Type) that nodes in a network using NetInf may find useful, e.g., as input for caching strategies.

For metadata, we recommend and have explored a convention for the use of the `application/json` MIME type, which appears to represent a useful trade-off between flexibility and simplicity. Objects that are badly encoded or lack typing can be treated as `application/octet-stream`, resulting in additional resilience in the face of programmer error.

2.2.3 Security Services

One of the prominent features of NetInf is a different security model compared to host-based communication that mostly relies on connection security. In NetInf, objects are replicated throughout the network, which means that the authenticity of those objects cannot be asserted by any transport security services.

Also, in NetInf it is important for receivers, but also for network nodes, to be able to check that a given object actually corresponds to a given name, for instance to the name that has been used in an application's request. This property is referred to as *name-data integrity*, and in this section we present two variants of name-data integrity that can be provided based on the naming and object model concepts described above.

2.2.3.1 Basic Name-Data Integrity

Given the use of the ni: naming scheme and MIME objects, the basic name-data integrity operation (verifying that the object returned actually matches the name requested regardless of the cache from which the object was fetched) can easily be achieved, as currently offered by the ni: scheme definition. The basic scheme is to include a hash (e.g., using the SHA-256 algorithm) of the object in the name. The bytes that are input to the hash are well-defined and follow the S/MIME message specification (RFC 5751 [7], section 3.1).

Applications may choose to verify name-data integrity or may not care; NetInf routers, however, may reasonably choose to only cache objects for which name-data integrity has been verified (to avoid obvious denial-of-service attacks) subject to the availability of resources. NetInf implementations should therefore provide tools to enable applications to generate and verify name-data integrity.

2.2.3.2 Signature-based Name-Data Integrity

While the basic hash-based name-data integrity service is simple, robust and easy to implement, it does require that the bytes of the object are known before the name can be distributed. This means the basic scheme cannot easily handle dynamic objects where the value changes frequently.

An alternative way of providing this service, based on digital signature, is therefore also defined for the ni: scheme. This alternative essentially consists of including a hash of a public key in the name and then distributing a digitally signed version of the object, including the public key (or certificate) thereby allowing an application (that cares) to establish that the holder of the private key was actually involved in signing the object. This again provides a kind of name-data integrity and should be supported in NetInf implementations.

However, this scheme (and all similar schemes, e.g., as used in DONA [8], 4WARD-NetInf [9] and CCN [10]) suffer from serious problems related to access to the private key and key revocation. While technical solutions for key/certificate revocation are available (e.g., use of X.509 CRLs or OCSP), those will probably not scale sufficiently and will also impose a significant management overhead which is clearly undesirable.

In addition, these schemes create the threat that any miscreant with access to the private key can ensure that any object of their choosing will pass an application's or cache's name-data integrity checking and so also creates a requirement for on-going protection of private keys.[1] So, while we recommend that NetInf implementations do include support for digital-signature-based name-data integrity, we also consider that better approaches are really needed and should be a topic for further research.

2.2.4 Transport Services

NetInf nodes use the NetInf protocol, based on message forwarding, for which they will make use of some lower-layer networking technology. This may be IP-based, for example using a TCP connection between two Internet hosts, or may in future be based on non-IP networking technologies, e.g., being realised directly on top of point-to-point links without end-to-end networking capabilities. We refer to such communication as being provided by a "Convergence Layer" (using the almost identical term from the DTN architecture [11]). The differences between the DTN usage and the ICN/NetInf approach are explained more fully in the next section.

[1] These can no longer be considered theoretical disadvantages as was recently seen in the "diginotar" incident where a company was liquidated as a direct result of hackers gaining the ability to use the company's private key.

2.2.4.1 Convergence Layer Approach

We intend that NetInf nodes should communicate using a specific convergence layer (CL) which can be connection-oriented or not, unicast or multicast, may simply encapsulate higher-layer Protocol Data Unit (PDU)s or may reformat those entirely as long as they can be reconstructed, etc. This CL concept as depicted in Figure 2.1 provides a very useful model for abstracting the NetInf protocol from lower layers, thus ensuring implementations can be deployed in current networks but is also well-placed to leverage any future non-IP, e.g., NetInf-native, communications technologies that may emerge. CLs can also handle issues related to fragmentation and re-assembly of objects within the network.

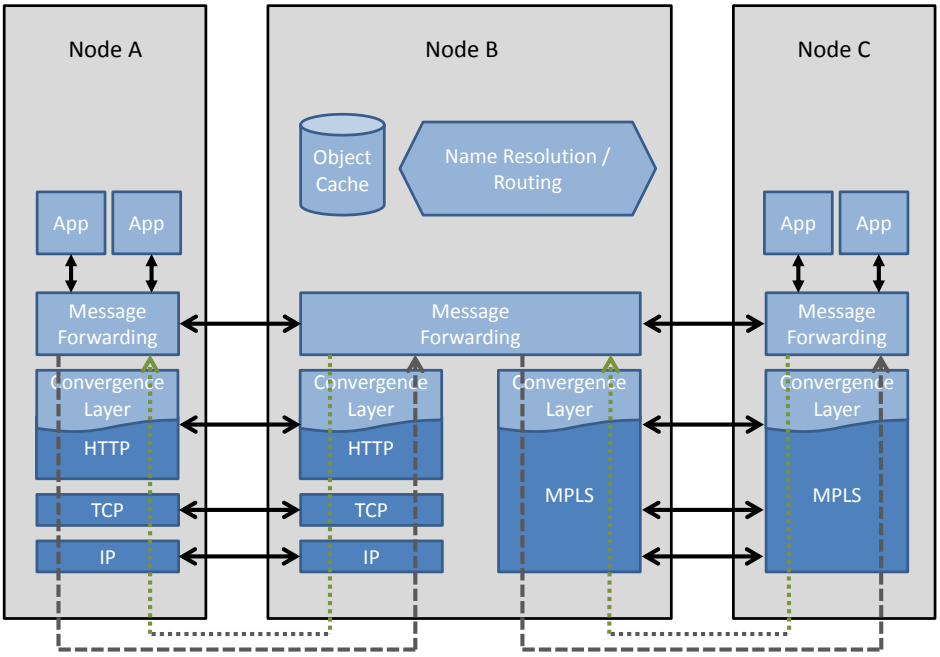


Figure 2.1: Example NetInf Convergence Layers

The communication at the convergence layer is only hop-by-hop between NetInf nodes, i.e. there is no communication over multiple NetInf node hops at the convergence layer, only direct communication between CL endpoints. The convergence layer does not make use of any NetInf layer node identifiers.

Depending on the details of a specific NetInf deployment, one or more CLs may be used in the overall network. There may be one common CL (that most nodes are expected to support) and a set of less common ones – for instance, for specific access network types. We need not specify such deployment details at this time though recognise that they will need to be specified for real NetInf deployments.

2.2.4.2 The Nature of a Convergence Layer

The idea of NetInf convergence layers is inspired by and similar to, but differs subtly from, the use of the same term in DTN. This seems to have proven to be useful to those who are already used to the DTN term, but problematic for those for whom this terminology and approach are new.

In DTN [11], [12] the terms convergence layer (CL) and Convergence Layer Adapter (CLA) are used to denote the protocol stack (CL) and accompanying code (CLA) that underlie the Bundle Protocol (BP).

At heart the CL transports the NetInf messages between two NetInf-capable nodes that are, at the time when the CL instance is used, topologically adjacent ('a hop'); mobility of nodes means that topological adjacency may change over time. This matches the way that a CL is used in DTN.

However a DTN CL can do many additional things that are not appropriate for NetInf, for example:

- radically re-formatting bundle fields and field values as is done when compressing headers in [13],
- creating new bundles as is done when reactive fragmentation is used to mitigate the effects of premature termination of a TCP connection in [14], and even
- encapsulating multiple bundle flows in one CL "session" as is done in in the Licklider Transmission Protocol CL [15].

In NetInf a message follows a path where each hop uses some CL, but each hop may use a different CL (or not): this is the same model that is used by DTN CLs, however, we consider it far less likely that a single transaction would traverse two or more different CLs, since almost all NetInf networks will be well-connected.

When using NetInf, CLs also help to isolate the higher layer protocol from lower layer details such as specific transports (TCP, User Datagram Protocol (UDP), Real Time Protocol (RTP) etc.) and how those manage congestion and fragmentation but in contrast to a DTN CL, the NetInf CL *does not* subdivide NDOs for onward transmission. The CL must deliver the NDO intact across the hop where it is used.

The other main difference between DTN and NetInf CLs is that in NetInf we are only currently dealing with an abstract higher level protocol and (to date) have not defined an analogue of the BP, with a concrete "bits-on-the-wire" format.

Partly, this is because work on NetInf is at an earlier stage and partly this is also due to lessons learned in the definition and subsequent use of the BP - while RFC5050 is a very useful basis on which to run experiments, it has also to some extent overly constrained those experiments. Experience with DTN tells us that delaying some of those higher level protocol specifics until we have done more experiments is likely to be beneficial.

In NetInf however, since we are also aiming for more short-term and larger scale deployments we would like to be able to more closely integrate the NetInf protocol with CLs that are either already in widespread use (such as unmodified Hypertext Transfer Protocol (HTTP)) or to construct CLs from existing operator networks. While DTN is not a major use-case for ICN, we can however, also consider the BP, (with Bundle Protocol Query (BPQ) [16]) as together forming a NetInf CL. We can also envisage NetInf CLs that use CoAP [5] or WebSockets [17] or SPDY [18] perhaps being eventually the main CL used on the requester side.

Once we have gained more experience with NetInf (for example when authorisation or routing have been more fully explored) we may then be in a better position to define a higher layer NetInf protocol that fully specifies the bits-on-the-wire, but so far we are not at that point.

The term *layer* as used in *convergence layer* can regrettably also be a source of confusion. In DTN, and also in NetInf, when following the *CL architecture* we are not dealing with a strict layering, as envisaged in the OSI 7-layer model. In contrast, higher *layer* PDUs may be significantly re-formatted by lower *layers* rather than simply being encapsulated (i.e. pre-pending a new lower layer header) and forwarded.

Thus a CL does not treat the PDU as opaque data but is able to understand the structure and nature of the content of the message fields so that they can be transmitted effectively but it must not alter the content.

One could think of the CL architecture as marshalling higher "layer" PDUs where the marshalling function at each CL-hop can do whatever is required to allow for exact reproduction of that higher "layer" PDU on the other side of the CL-hop and do it efficiently in the context of the transport protocol used. This is more akin to how RPC works, rather than the OSI 7-layer model. This approach allows (arguably) for more flexibility in how a CL does its job, for example network coding fits this model nicely (when multiple code words are emitted, possibly on different links) whereas network coding is hard to model based on pure encapsulation. As another example, in the NetInf HTTP CL currently being developed, we are exploring mainly encoding the NetInf protocol fields into form fields or MIME objects carried in HTTP POST requests and responses.

While this use of the term *layer* is admittedly confusing, no better term emerged despite extended debates on the topic when the DTN architecture was being developed.

2.2.4.3 Object Caching and Fragmentation

In general, NetInf nodes requests and receive NDOs, which we consider as Application Data Units (ADUs), i.e., publishing applications creating those objects decide on what constitutes a NetInf object – depending on their requirements (Application Layer Framing [19]). Additional segmentation at the NetInf layer is not envisaged. However, there may be segmentation into smaller units in a Convergence Layer, for example in order to address maximum Maximum Transmission Unit (MTU) size constraints.

Consequently, NDOs are the smallest addressable and verifiable unit in NetInf. This does not imply that NetInf can only have large objects. Instead a publishing application will typically publish a set of NDOs for "larger" objects such as videos. NDO boundaries will be decided by the publishing application, e.g., following the Application Data Unit approach mentioned above. APIs and corresponding operating system functions can perform this on behalf of applications, for example for ADU-agnostic applications and media types. NDOs can be cached independently (i.e., without considering a larger object context).

Similarly, receivers can request such NDOs independently, e.g., in case a user is only interested in a certain part of a video. In order to enable receiving applications to learn the set of NDOs that belong to a larger object context, NetInf applications can use dedicated index NDOs that provide the necessary metadata, either as a list of NDO names or as a specification of how to derive the list of NDOs names (in case of dynamic naming for example). Such information can also be leveraged to inform caches and NRS nodes in order to optimise caching and name resolution behaviour.

In order to address the scalability requirements NetInf can provide aggregation mechanisms, for example a set of NDO belonging to a common context can provide a common prefix (especially if signature-based name-data integrity is employed). Other options (leveraging NDO attributes) are being investigated.

NetInf (and the remainder of this document) is therefore using the general term *NDO* to refer to data objects that can be accessed by name, using the NetInf Protocol. The term *chunk* is not required (and hence not used) since NDOs are the smallest addressable unit in NetInf.

2.2.5 Conceptual Protocol

A specific CL implements a conceptual NetInf protocol that provides the following fundamental messages and corresponding responses:

GET/GET-RESP The GET message is used to request an NDO from the NetInf network. A node responds to the GET message if it has an instance of the requested NDO; it sends a GET-RESP that uses the GET message's msg-id as its own identifier to link those two messages with each other.

PUBLISH/PUBLISH-RESP The PUBLISH message allows a node to push the name and, optionally, a copy of the object octets and/or object meta-data. Whether and when to push the object octets vs. meta-data is a topic for future work. Ignoring extensions, only a status code is expected in return.

SEARCH/SEARCH-RESP The SEARCH message allows the requestor to send a message containing search keywords. The response is either a status code or a multipart MIME object containing a set of meta-data body parts, each of which MUST include a name for an NDO that is considered to match the query keywords.

The details of the conceptual protocol are fully described in Appendix A.

2.2.6 Convergence Layer Implementations

A number of implementations of convergence layers for the NetInf protocol have been prototyped and have been released as open source software available from the NetInf repository on Sourceforge[2]. The implementations primarily focus on the use of HTTP over TCP as a transport for the NetInf protocol, but there is also a simple UDP transport and work is in progress on a DTN Bundle Protocol based transport.

2.2.6.1 HTTP Convergence Layer

Implementations of the NetInf conceptual protocol have been made in several different computer languages including C/C++, Python, Ruby, Java and Clojure. NetInf clients are provided in each of these languages. Currently implementation has focussed on GET and PUBLISH functionality. SEARCH functionality will follow in due course.

Since HTTP operates over a TCP connection, the HTTP convergence layer is strictly unicast. A request goes to a single destination server identified in the authority field of a synthesized HTTP request. The HTTP CL uses unmodified HTTP GET (or HEAD) and POST messages - no special HTTP functionality is required.

For testing purposes, a PHP servlet has been provided that can be installed in an Apache web server and a standalone dedicated NetInf server has been written in Python. In each case the server is set up to respond to

- GET requests to the server location using paths of the form `/.well-known/ni/<digest algorithm id>/<digest base64url encoded>`
- POST requests to the server location using the paths `/.well-known/netinfproto/get` and `/.well-known/netinfproto/publish`

The ni: URIs used with GET requests are transformed into http: URIs using the following rules [4]:

- Authority copied from ni: to http: (or chosen by user).
- http: path initialized as `/.well-known/ni/.`
- Path component (there is only one - the digest algorithm identifier) in ni: appended to http: path.
- Path parameter (digest representation base64url encoded) in ni: appended to http: path.
- Query string from ni: appended to http: URI as (same) query string.

[2] http://sourceforge.net/projects/netinf/

For the GET requests, the contents of the specified NDO will be returned if available.

For the POST requests, the NetInf protocol messages are formatted as HTTP POST messages with the NetInf parameters encoded as an HTTP form (a specific type of MIME encoding). The results are returned as a MIME-encoded object. Initially these are single part MIME messages but where multiple result components are returned, the result is encapsulated in a multi-part MIME object.

Both server implementations are able to serve a simple form that can be displayed in a conventional browser in response to a GET request to `/getputform.html` to allow the user to specify the parameters to be sent in the POST requests.

In addition to the browser forms, command line clients are provided in each of the languages noted which allow NDOs to be retrieved and published. In support of this, there are also utility programs and libraries that will generate the ni: URI for a file or string using the cryptographic hash algorithms available in the Open SSL libraries (primarily the SHA-2 series) with optional truncation of the digest code.

Extensions of this work currently planned will allow multiple locators to be retrieved and searches to be performed within the cache at a server node and allow requests to be propagated to nodes over different CLs.

2.2.6.2 UDP Convergence Layer

The initial stages of a UDP-based CL have been developed and an implementation is available coded in Ruby. The UDP CL is intended to support situations where a "crowd" of nodes supporting the NetInf protocol are operating in a dense pack. In these circumstances it may be difficult to establish individual HTTP/TCP connections with each available node to determine if a particular NDO is available. To avoid this, the UDP CL operates on a multicast IP address, but instead of returning the actual object content in response to a GET or SEARCH request, nodes receiving a GET or SEARCH request over the UDP CL will reply with one or more locators for nodes where they know the NDO is available or remain silent if they have no knowledge of the NDO requested. The requesting node can then use the supplied locators to make a (unicast) request over HTTP to retrieve the object. This avoids having to deal with fragmentation, reassembly and reliability issues when dealing with replies over UDP.

2.2.6.3 DTN Bundle Protocol Convergence Layer

The DTN BP can be used as a CL for NetInf. An implementation of NetInf in the DTN2 reference implementation is being developed by SAIL partners TCD for the NetInf device prototype. The BP has been extended by the addition of the BPQ extension block which can carry ni: URIs, locators and/or search parameters. The persistent store of bundles maintained by each DTN node provides the cache of NDOs to be maintained by each node, and search mechanisms have been implemented to allow a SEARCH request to examine all the bundles that have ni: names currently stored in a node's DTN2 cache.

2.2.7 Routing and Forwarding

NetInf needs to forward and/or resolve requests for objects as well as forward the response messages. We postulate that specific implementations will differ in how they handle requests and responses. Furthermore, different parts of the network have different routing requirements and thus will need different routing protocols, just as we have multiple routing protocols for Internet Protocol (IP) today. NetInf implementations should thus provide a way to easily plug in new routing, name resolution, or forwarding schemes.

This is especially important since research on scalable and sufficiently well-performing global routing and forwarding schemes is still on-going. Credible research on this necessarily requires large-scale experimentation and serious validation based on real prototypes. To build larger experimental networks, interoperability will be a critical and primary factor – many different NetInf implementations must be able to work easily as part of the same information-centric network, at least in principle, so that large-scale experiments are possible and so that results are comparable. This means that it is important that implementations should not gratuitously add options since each "knob" that can be turned decreases the likelihood of interoperability.

2.2.7.1 Request Forwarding

For forwarding requests for objects, we envision a hybrid routing/forwarding scheme, integrating pure name-based routing aspect (in the sense of "routing on flat names") as well as name-resolution-based aspects. The name-based routing could, e.g., be based on simple wild-card name matching with default routes, so that, e.g., requests from inside a network can reach an "edge" with external access, at which point name-resolution is likely to be used (such a scheme seems suitable for a delay-tolerant network). Name resolution for ni: names is similar in concept to how the DNS works today – a name is supplied and the NRS will (usually) return locators that allow the named object to be retrieved. There are two types of NRS that we currently consider – one DNS-based and one based on Distributed Hash Tables (DHTs).

The DNS-based scheme could make use of DNS SRV records to allow a requesting node to find a service that can further resolve the name or return the object. Various DHT schemes have been suggested and even implemented for ICNs, such as MDHT [20] already implemented in the NetInf approach [2] and in the OpenNetInf implementation[3].

As argued above, there is a need for multiple routing schemes for different parts of the network. For establishing connectivity at the global level, however, we will need one routing scheme, just like we have Border Gateway Protocol (BGP) for the Internet. Such a scheme adapted to the convergence layer protocol approach is described below in Section 2.3.

2.2.7.2 Response Forwarding

Responses also need to get back to requestors. As with request routing, NetInf requires flexibility in how response routing can be done. If an implementation provides a routing/forwarding pluggability framework, then this framework should also support pluggability of response forwarding schemes; but of course should also choose and implement at least one concrete scheme as well.

For response forwarding, the main issue is how to handle the state required to make sure the response gets back to the right place when the request has spanned more than one CL "hop." The maintenance of in-network state for a single CL "hop" is something that is handled by the specific CL used, e.g., if a TCP CL is used then the response must be generally returned on the same socket from which the request was received. (Even here, reboots and other state-loss issues need to be tackled however.)

In the general case where a request has passed over many CL "hops," the issue boils down to how to associate a response coming from "downstream" (towards the requestor, from the "source") with a request previously received from the "upstream" direction (from the requestor).

The NetInf protocol allows for different approaches for that, and depending on the deployment scenario and scalability constraints, NetInf nodes can be configured to use one specific approach, including maintaining state in the routers and annotating the request with some form of token or label. Labels can be locally significant identifiers, and label stacks can be used to record a request

[3] http://www.netinf.org

route for returning responses along a path of NetInf nodes. For example, the GIN approach described in Section 3.3.6 uses this approach.

2.2.8 Platforms and APIs

The conceptual protocol specification (in conjunction with the object model, the naming approach, the security services and the convergence layer implementations) that we have described above has to be mapped to an API that NetInf applications can use. Figure 2.2 provides an example of how a C API for NetInf might look (note that other language bindings will of course be possible, too):

```
// To retrieve an \ac{NDO} and/or extra locators
int get(int handle, \ac{URI} objectName, int {*}len, unsigned char {*}{*}mimeOctets);
// To publish an object and/or locators where it may be retrieved
int publish(int handle, \ac{URI} objectName, bool fullObject, unsigned char {*}mimeOctets);
// For asking about name-data integrity etc.
int getAttr(int handle, int attrType, int {*}len, char {*}attrVal);
```

Figure 2.2: C language API

At present the convergence layer clients provide this API directly but this would be modified in situations where it was necessary to select the CL to be used for a particular request depending on the interface and locator(s) to be accessed, if they are known, with a particular request.

2.3 Global Connectivity and Inter-Domain Interfaces

The Global Information Network Architecture (GIN) [2, 21] has defined a viable and scalable routing solution for ICN global connectivity. The principal components for global routing in Global Information Network (GIN) are: (1) *The REX*, a global name resolution system that maps the authority part of NDO names into routable network domain identifiers (locators). (2) *A BGP-style routing system* for the network domain identifiers. The network identifiers can be compared with today's IP prefixes which are announced in the global BGP routing infrastructure. (3) *Label stacks* (source and destination) in the packets carrying network and node identifiers (locators). The stacks facilitate explicit aggregation of routing information and late binding of names to locators.

In this section, we describe a variant of GIN global routing adapted to the NetInf convergence layer approach described above in Section 2.2.4. The variant is also inspired from Narayanan and Oran [22].

We are here mainly concerned with the *global* aspects of routing requests for NDOs towards the corresponding publisher, i.e., routing in the NetInf default-free zone, corresponding to the current Internet's BGP-routed infrastructure. Just like in the current Internet, edge domains can utilise different routing schemes that are adapted to particular needs of the respective domain.

Conceptually, NetInf global routing is a name-based routing scheme, but to enable aggregation of routing state, routing is aided by translation to locators. These locators are called *routing hints*. For reasons of easy and incremental deployment, we furthermore assume that we use the current IP addresses as locators. The NetInf routing scheme thus operates using the ni: URI scheme in combination with IP addresses.

The NetInf global routing scheme consists of:

- *Routing hint lookup service*, a global name resolution system, that maps domain names from the ni: URI authority field into a set of routing hints. It is very similar to REX mentioned above.
- *NetInf BGP routing system* for the NetInf routers in the default-free zone.

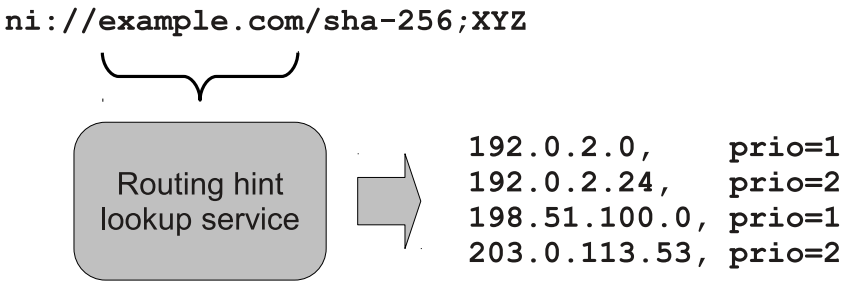


Figure 2.3: The routing hint lookup service.

- *Routing hints* that aid global routing by providing aggregation for routing information. The routing hints serve a similar purpose as the GIN label stacks.

2.3.1 Assumptions and Requirements

Before going into the detailed design we briefly review the prerequisites. An ICN needs in principle to keep track of all NDOs available in the global Internet, just like the current Internet in principle needs to keep track of all hosts. Estimates of the number of objects range from 7 billion to one trillion (10^{12}). To have margin for growth, we need to aim higher, at least to 10^{15}. The key to be able to globally route requests for this large number of NDOs is *aggregation*.

We assume that the authority part of the ni: URIs use the DNS domain name space. It has currently some 130 million second-level domain names, which we believe is a good estimate of the number of different authority names. Aggregating routing at this level is thus very attractive, since the existing DNS infrastructure can be leveraged and since the aggregation factor is very large.

2.3.2 Routing Hint Lookup Service

The NetInf routing hint lookup service provides mappings from domain names to sets of routing hints, as illustrated in Figure 2.3. The service leverages the existing DNS for scalability. Two design options are considered: define a new DNS record, or use Service (SRV) records to link to a dedicated resolver.

Publishers, who want their NDOs to be reachable from everywhere, need to define mappings for their domains used in the ni: URI authority part leading to a server where the NDOs are made available. At least one hint in each mapping has to be globally routable, i.e., announced in the NetInf BGP routing system. Mappings are shared between all NDOs with the same authority part in their names, so clients can cache the mappings similarly to DNS.

The scoping discussed in Section 2.2.1.2 is a complement to the authority part that also can be used to lookup routing hints, with the same or a dedicated lookup system. We expect that further experimentation will provide insights in the need and usability of scopes for this purpose.

ni: names with an empty authority part will need a different routing hint lookup service than the one described here. The only known technology that can provide the needed mappings is distributed hash tables (DHT). Such a system can be deployed as a complement.

2.3.3 NetInf BGP Routing System

The NetInf BGP routing system provides routing between the NetInf routers in the default-free zone. It is *not* the same as Internet's BGP system, but rather a dedicated routing system using the same routing protocol, including the IP address namespace, which provides routing for the routing hints.

Only the routing hints at the highest aggregation level for each operator or organisation need to be announced in the NetInf BGP system. More specific routing hints within an operator or organisation can be handled solely with intra-domain routing. Arbitrary levels of aggregation can be implemented, for instance, corresponding to current autonomous systems, operator, hosting site, and organisation.

Since we use the IP address namespace as routing hints, we foresee that very little modifications, if any, are needed to the BGP protocol. The use of IP anycast would be ideal for our purpose, as a publisher most likely wants to make its NDOs available from multiple sites.

There are several reasons we need the NetInf BGP routing system. The first is to enable general caching in the default-free zone. We could use the routing hints directly as destination addresses for requests, removing the need for the NetInf BGP routing system. That would mean that requests would take one hop over the default-free zone, bypassing caches on the way, and losing the possibilities BGP gives to operators to control the NetInf data flow including where to cache in their networks. That would also mean that we lose the ability to aggregate routing information. The routing hints would need to address specific nodes. Finally, it would mean that it will be hard to use more than one CL in the default-free zone.

2.3.4 Routing Hints

Routing hints aid routing of NDO requests when a router does not have better information available for the requested NDO. The hints are typically looked up once in the lookup service and put in the NDO request. A hint generally does not name the end-point (destination node) for the request. The hint is instead just used to find the next hop to forward the request to.

The routing hint lookup service typically supplies multiple hints. There are several reasons for this. For the purpose of aggregation, not all hints are present in the global routing table. It also enables retrieving an NDO from multiple sources, and/or selecting the best source.

As already mentioned, we have chosen to use the IP address name space for the routing hints. There is however one important difference in how we use the name space. We have no use for the division in network and host parts. Multiple hints serve the same purpose of aggregating routing state. This means that forwarding table lookups can be done using exact match, resulting in simpler and more efficient forwarding table and process.

Each routing hint returned by the lookup service has a priority. Hints at higher aggregation levels normally have lower priority than more specific hints. The priority enables a router to choose the most appropriate hint when there are multiple matches in the forwarding table.

The routing hints are deliberately chosen to be agnostic to the choice of convergence layer. The reason is that we cannot assume that all nodes support all convergence layers. The choice of convergence layer is only a matter between the two nodes that the convergence layer connects. We can thus not encode the choice of convergence layer in the routing hints. This selection is instead made from the forwarding table in the respective NetInf router.

2.3.5 Forwarding Tables and Process

A NetInf router has a ni: name and/or a locator (routing hint) forwarding table. The ni: name forwarding table is used by routers performing name-based routing, which primarily is intended for edge domains. In the default-free zone, the locator forwarding table is needed for looking up routing hints. Which tables are used by a particular router is a configuration issue. In addition to the forwarding tables, a router needs a table listing NDOs that the router serves directly, from its cache or from persistent storage.

Please note that we do need a NetInf-specific locator forwarding table in addition to the normal IP forwarding table, despite the fact that we use IP addresses as routing hints. If *all* IP routers also

locator	CL-specific next-hop
192.0.2.0	http://global.example.com/.well-known/netinfproto/get
192.0.2.24	http://edge.example.com/.well-known/netinfproto/get
10.1.10.1	http://local.example.com/.well-known/netinfproto/get

Table 2.1: Example locator (routing hint) forwarding table.

were NetInf routers, the IP next-hops would all be NetInf-capable, and would thus also be usable as NetInf next-hops, removing the need for separate tables. In the short term, this is certainly not the case, and it might not be in the long term either. The NetInf next-hops can therefore often not be the same as the IP next-hops for a particular router and we thus need a NetInf-specific locator forwarding table that is populated from NetInf-specific routing protocols.

Table 2.1 shows an example locator forwarding table. All routing hints in the NDO request are looked up with exact match, resulting in a set of CL-specific next-hop addresses. The lookup thus selects both which CL to use and the next-hop address for that CL. The router forwards to the next-hop matching the hint with the highest priority. In a real implementation, the next-hop field should of course be encoded in a more efficient form than is shown in the example.

The forwarding process has a number of steps:

1. Check the object table (cache and permanently served objects)
2. Check ni: name forwarding table: lookup the ni: name using exact match; if a match (incl with a default entry), forward to that next-hop
3. Perform any routing hint NRS lookups, resulting in additional hints
4. Check locator forwarding table: look up all routing hints with exact match; forward to next-hop matching the hint with highest priority

As already mentioned, these steps depend on the configuration of the particular router. The global NRS lookup step is for instance typically only done by some ingress router. A router can also be configured to use other, domain-specific, resolution services.

2.3.6 Inter-Domain Interface

A high-level NetInf inter-domain interface was described in SAIL Deliverable D.B.1 [2, Section 4.6.4.2] in the form of two abstract PDUs. That description only covers the data plane of the interface. In principle, all three pairs of messages in the conceptual NetInf protocol (Section 2.2.5) need to be handled over the inter-domain interface:

GET/GET-RESP These data plane messages for requesting and delivering NDOs are used more or less directly over the inter-domain interface.

PUBLISH/PUBLISH-RESP The PUBLISH message makes an NDO available in the network, and thus may need to interact with the particular routing protocol and/or name resolution system used in the respective domains. These messages are therefore handled in a domain-specific manner at the inter-domain interface. For instance, in the NetInf default-free zone the BGP protocol in combination with the routing hint lookup service provide this function.

SEARCH/SEARCH-RESP More experimentation is needed to understand if and how the SEARCH message is handled at the inter-domain interface.

2.4 Operational Considerations

The NetInf Protocol design, the Convergence Layer approach and the Global Connectivity and Inter-Domain Interfaces enable different instantiations of NetInf networks, each of which under their own administrative domain. The Convergence Layer approach supports technological diversity, ensuring that the same conceptual protocol can be used in different types of underlying networks. The NetInf Protocol ensures that trans-network communication is possible, and the Global Connectivity and Inter-Domain Interfaces provide the basis for connecting a large set of networks and administrative domains into a global NetInf infrastructure. This section provides a few examples of what such network constellations might look like and what operational considerations would be important in these cases.

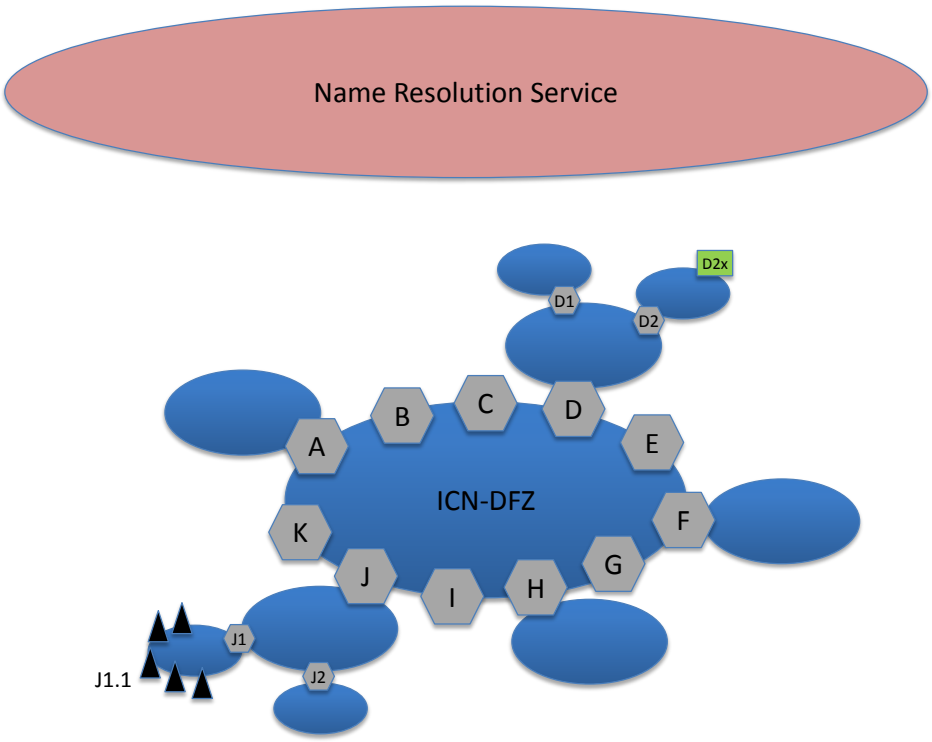


Figure 2.4: Example NetInf network

Figure 2.4 depicts a schematic global NetInf network with different access networks, aggregation networks and a core network that connects the other networks (and that provides a routing infrastructure for that). In addition there is a NRS, for example a multi-layer DHT-based NRS as described in [2] or, either in addition or alternatively, a DNS-based NRS that can be employed by NetInf nodes to look up NetInf names and obtain next-hop locators.

In general, all nodes in such a network are NetInf nodes, i.e., they implement the NetInf protocol and at least one Convergence Layer. Depending on the specific function of a node and its location in the network, it may implement specific additional functions – such as caching or name resolution – and may exhibit specific behaviour, for example participating in a name-based routing protocol or engaging in a global Name Resolution Service.

For building real-world networks such nodes will be arranged in a certain topology and additional mechanisms will be applied, for example individual nodes execute certain transport and/or caching strategies, and there may be cache management (and other node management) interfaces. The general NetInf node features have been described in previous sections of this chapter. Chapter 3 describes a set of additional mechanisms (transport strategies, caching). In the following, we

describe how different nodes in our example network can make use of the NetInf node features and the additional mechanisms.

NetInf User Device Here, we assume the lower left hand side network connected to node `J1` to be a cell in an access network and node `J1.1` to be a NetInf node, for example a user device, within that network. A NetInf user device provides one or more physical network interfaces. It supports the NetInf protocol and provides at least one Convergence Layer implementation (here: one implementation that allows it to connect to the operator network.)[4] We assume that the user device is configured with a designated next-hop NetInf node that performs access router (and possibly caching) functions. It should be noted that a user device is not limited to content (NDO) consumption. Naturally, it can also create NDOs and make them available to other users. In this case, the user device would `PUBLISH` locally generated NDOs to its designated access router.

Access Network The access network is assumed to be operated by a network operator, i.e., there is infrastructure in terms of base stations, caches, NetInf routers and NRS systems. A NetInf access router acts as a default next hop for user devices in a cell. It may provide a particular Convergence Layer implementation for the wireless network interfaces (i.e., one that is suitable for the respective link layer properties) and another Convergence Layer implementation for communicating with uplink nodes.

In access networks, caching will be of particular importance and so we expect the access router to either perform caching itself or to be able to utilize a cache in the operator's access network. In addition to caching such a node should also perform request aggregation to minimize requests and responses for the same (popular) NDOs passing the network.

This can be achieved by preferentially performing name-based routing and maintaining data structures for outstanding requests. The access router can maintain tables for locally published NDOs and can participate in a simple name-based routing between a set of access routers in the access gateway. Keeping track of outstanding requests is quite feasible in access networks, since the number of downlink nodes is typically limited. Chapter 3 presents some results of our work on corresponding transport and caching strategies.

Aggregation and Operator's Core Network We assume some form of aggregation at a higher level in the network, for example in an operator's backhaul or core network (depicted as the network between `J` and `J1` in figure 2.4. How to do request forwarding decisions and what particular Convergence Layers to use is in principle an operator decision. Also, operators are of course free to configure caching strategies for different nodes at different tree levels and to decide on other operational details (e.g., cache pre-population, other resource management). Here, we assume that the particular operator employs name-based routing at `J1`, with a default route to `J`. Node `J` is connected to a Name Resolution Service that is able to map NetInf names (respectively their authority component as described in Section 2.3) to routing hints.

Request Forwarding Example Figure 2.5 depicts an example message forwarding sequence based on the operational considerations discussed so far. The different messages (numbered 1 to 6) in this example exhibit the NetInf protocol message type and additional message parameters such as the requested NDO, routing hints and a label stack for recording a return route.

The setting is as follows: as described above, there is a requesting user device in `J1's` access network cell.

[4] Chapter 5 describes an extended version of this scenario where nodes have additional interfaces for node-to-node communication.

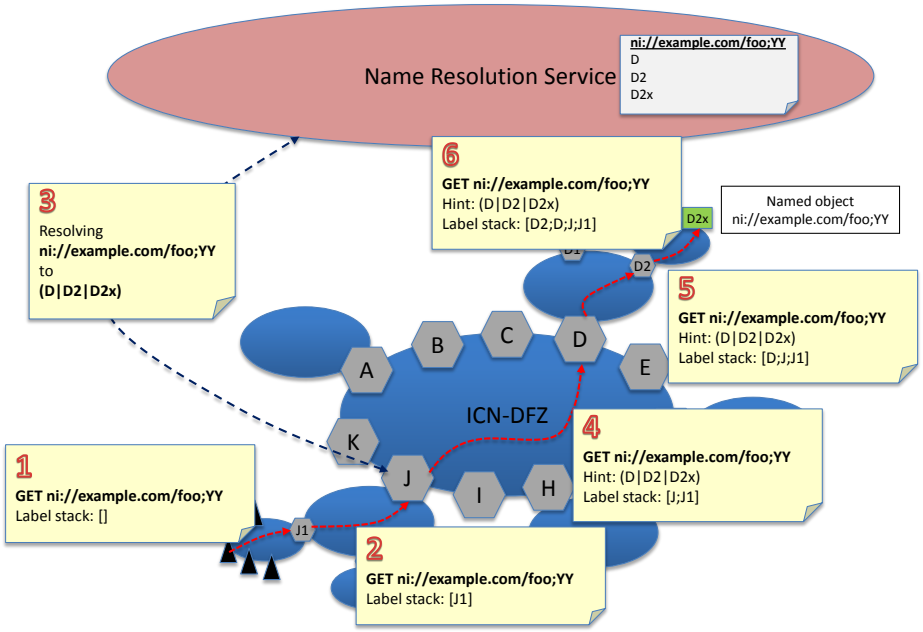


Figure 2.5: Example NetInf forwarding sequence

1. The user device is requesting the NDO `ni://example.com/foo;YY`.
2. The request is forwarded (per default forwarding rules) to node `J`.
3. Node `J` does not have any routing information for deciding about a potential next-hop for the request, so it consults the NRS and retrieves a list of routing hints (`D|D2|D2x`).
4. Node `J` is assumed to participate in an inter-domain routing scheme as described in Section 2.3 and is thus able to use the routing hints to find a next hop. Here, we assume that node `D` (we did not specify the lower-layer network address here) would be that next-hop.
5. Node `D` belongs to a different operator's domain and does its own forwarding decision for the request. This could be based on the routing hints that had been passed to it and that are mapped to a next-hop leveraging routing information that is available within that operator's network.
6. Node `D2` happens to be the next hop and `D2` knows how to reach node `D2x` in the leaf network so that the request can finally be delivered.

The response would be delivered in backwards direction of the recorded route in the label stack that was received in the request packet (not shown here). In this example, there was obviously no cached copy of the requested NDO that was accessible to the node along the path to `D2x`. If there had been one, of course the requests could have been satisfied from such a cache.

Moreover, it should be noted that each of the hops in figure 2.5 corresponds to a NetInf Convergence Layer hop. In principle each hop could employ an individual CL; however we assume that in real-world networks a few standard ones would emerge. For example, standardization for mobile communication systems would likely yield an optimized wireless CL and there would be probably be one standard CL for inter-domain communication, too.

Transport Services Obviously such a network has to deal with resource management, specifically congestion and flow control, but also cache management, as mentioned above. The following Chapter 3 describes mechanisms that we expect to be used for such functions.

2.5 Summary

This chapter has described the NetInf foundations that are based on a selected set of invariants, outlining important guidelines for how the NetInf ICN should work in general. With these invariants as corner stones, we have defined the NetInf Protocol as the general *conceptual* protocol that is supported by *NetInf nodes*, i.e., the entities in NetInf networks. The NetInf Protocol provides the *Convergence Layer* concept as a key element, which is one enabling element for the multi-technology and multi-domain support in NetInf. Another enabling element is the *Global Connectivity and Inter-Domain Interfaces* approach that has been described in this section.

The concepts, protocols and conventions described have been deliberately defined in a way that they can be applied to different specific scenarios, also taking requirements for future evolution into account. Section 2.4 has put the different components together for illustrating a specific example (here: an Internet with operated edge/access networks). It should be noted that this is *just one* example. The NetInf Protocol can obviously be used in quite different ways, for example in networks with lesser infrastructure support/dependencies. Chapter 5 provides a more detailed description of an application scenario that extends the one in this chapter by local (direct) communication capabilities and improved resilience against infrastructure overload and other failures.

3 Transport and Content Placement

3.1 Introduction

The NetInf architecture gives details on naming NDO's, NDO lookup and delivery. They have been explained in Chapter 2 and the first project deliverable D.B.1 [2], specifically Chapter 4. Within this chapter we focus on solutions for the transfer and storage of NDOs in a NetInf network. Section 3.2 explains the transfer of NDOs in point-to-point and multipoint transfer mode. Section 3.3 deals with NDO storage mechanisms realized through on-path caching.

3.2 Content Transport

3.2.1 Reliable and Unreliable Transport Across Multiple NetInf Hops

As described in Section 2.2.4.1, "The communication at the convergence layer is only hop-by-hop between NetInf nodes, i.e. there is no communication over multiple NetInf node hops at the convergence layer, only direct communication between CL endpoints." As a consequence, NetInf does not need to use a traditional TCP- or UDP-like transport protocol end-to-end between a Requester and a Publisher when they communicate across multiple NetInf hops. Instead, NetInf may use a transport protocol that operates on a per-hop basis. For example, if an HTTP Convergence Layer is used over a hop between two NetInf nodes, HTTP may run over TCP. In this case, each end-point of the hop terminates both HTTP and TCP. Similarly, other Convergence Layers with other transport protocols may be used over other hops between NetInf nodes. Alternatively (or in addition) there can be Convergence Layers that provide a message transfer service only (without providing flow control).

Therefore, a path between a Requester and a Publisher may traverse multiple Convergence Layers, and multiple transport protocols as shown in Figure 2.1. The impact of this approach on transport reliability as well as flow and congestion control will be described below. However, to prepare for this discussion, we first need to describe in more detail the NDO transfer and multi-point aspects of NetInf communication.

3.2.1.1 Multi-Hop and Multi-Point NDO Transfer Scenarios

To facilitate transport and allow for load balancing, a large data object such as a movie, can be divided into smaller NDOs. Consider the multi-hop and multi-point transport scenarios shown in Figure 3.1. Two cases, a single Requester (case a), and a multiple Requester (multicast, case b), are shown. In case a, a single Requester in NetInf Node J requests 16 NDOs of a data object, numbered 1 through 16. Based on available routing information, Node J decides to try to reduce the response time by balancing the load on different links and next-hop nodes. Therefore, it splits the request into two parallel requests by sending one request for NDOs 1 through 8 to Node G, and another request for NDOs 9 through 16 to Node H. Nodes G and H make similar decisions, and try to reduce the response time by splitting their respective requests. As shown in the figure, an original request can be split recursively along a multi-hop path, so that NetInf Nodes A, B, C, F, and I finally serve the original request from Node J.

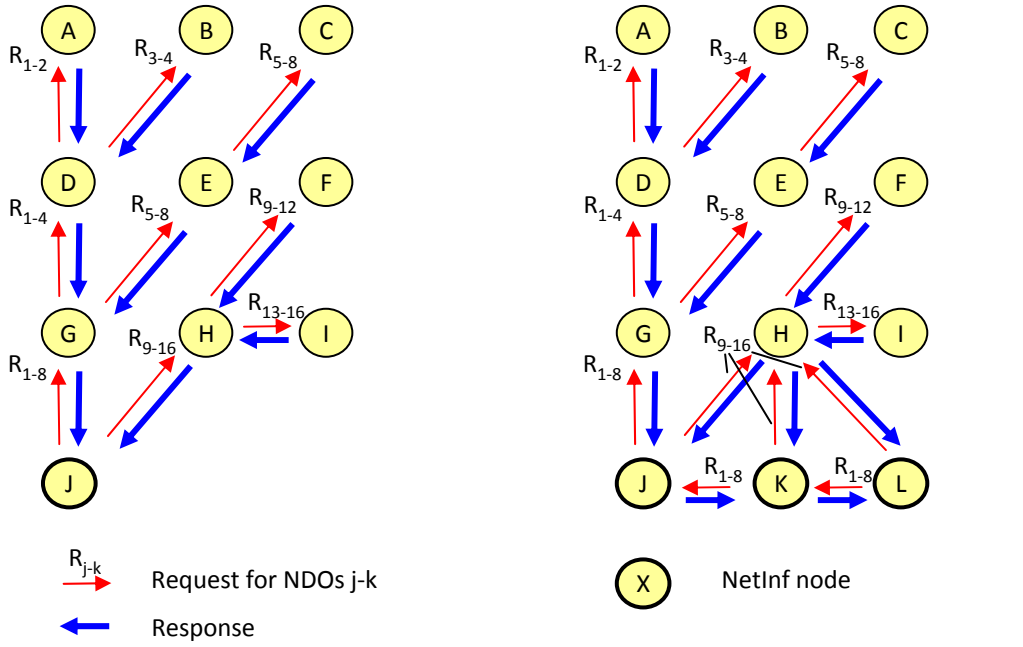


Figure 3.1: Multi-hop transport

Note that based on the Convergence Layer approach, the request message originating at Node J is processed by every NetInf Node along the path to the nodes that finally serve the request. Each node can make an independent decision on how to load balance a request by splitting it into parallel requests as shown in the figure. This is quite different from traditional end-to-end communication based on e.g. HTTP.

Case b is a multicast variant of case a. Requesters in NetInf Nodes J, K, and L each make a request for the same 16 NDOs as in case a. The request from Node J is split into two parallel requests to Nodes G and H in the same fashion as in case a. Node J sends a request for NDOs 1 through 8 to Node G, which load balances this request on downstream nodes in the same fashion as in case a. Originating Nodes K and L also load balance their requests, and split them into two requests in the same fashion as Node J, i.e. one request for NDOs 1 through 8, and one request for NDOs 9 through 16.

As shown in case b of the figure, Node J sends a request for NDOs 9 through 16 to Node H. Based on available routing information, Nodes K and L make the same decision and send their requests for the same NDOs to Node H, which thereby receives three different requests for NDOs 9 through 16. Node H aggregates these three requests into one request for NDOs 9 through 16. Node H then load balances this request on Nodes F and I in the same fashion as in case a.

The example shown in case b of the figure deserves some attention. When Node K sends a request for NDOs 1 through 8, Node J has already received and cached these NDOs. Node J has also published these cached NDOs. Node K is aware of this publication and therefore decides to send its request to neighbour Node J. Likewise, Node K has already received NDOs 1 through 8 and publishes that it has cached these NDOs when Node L sends a request for the same NDOs. Node L therefore decides to send its request to Node K.

This example shows that the timing of requests is important. The topology of the request aggregation tree may depend on the exact timing of publication of cached content. If Nodes K and L had sent their requests for NDOs 1 through 8 before these NDOs were published by Node J, a different request aggregation tree would have emerged. In this case, available routing information

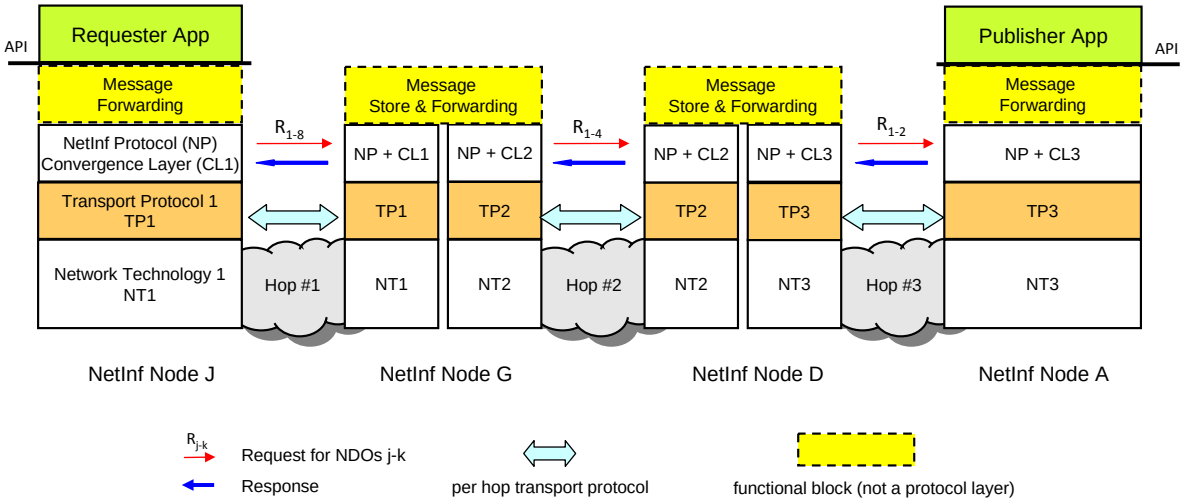


Figure 3.2: Per hop transport protocols between Requester and Publisher

would have indicated that Nodes K and L should have sent their requests to Node G, which would then have received three different requests for the same set of NDOs, and aggregated these requests.

3.2.1.2 Reliability Aspects of NetInf Transport over Multiple Hops

Based on the discussion above, we are now ready to describe the reliability aspects of NetInf transport over multiple hops. As previously noted, in the multi-hop case there is no end-to-end transport protocol under the Convergence Layer, but transport protocols operating hop-by-hop as shown in Figure 3.2. The figure shows a protocol stack view of the communication path between Node J and Node A previously outlined in Figure 3.1. These transport protocols may be reliable, such as TCP, or unreliable, such as UDP. We discuss these two cases separately below.

As a consequence of the Convergence Layer approach, there are no end-to-end semantics in the NetInf protocol. This protocol only communicates with neighbour NetInf nodes. When a NetInf node sends a request message to a neighbour NetInf node, this node returns a response message as shown in Figure 3.2. For example, Node J sends a request message to Node G in the figure. If Node G is successful in retrieving the set of NDOs from Node A via Node D as shown in the figure, it forwards the set of NDOs to Node J in a response message. Node J is thus completely unaware of the requests and response messages beyond node G, and only sees the NetInf Protocol messages exchanged with the neighbour NetInf nodes.

In case of a reliable transport protocol per hop, a node sending a NetInf protocol message to a neighbour node can trust that the message has been received correctly if an acknowledgment message is received from the neighbour node. However, even in case of reliable transport protocols at each hop on a path between a Requester and a Publisher, a message may still be lost due to e.g. an error in a forwarding table, or an error in the NetInf Protocol and Convergence Layers. Since these layers are above the transport layer, a transport protocol will not be able to detect errors in them. Such errors may be temporary, and a retransmission of a NetInf Protocol message may help. However, the remaining issue is to quickly detect that a retransmission is needed, and to decide which node should make the retransmission. This issue is for further study.

Consider for example the forwarding path from Node J across Nodes G and D to Node A in Figure 3.2. Assume that the request message originating from Node J is lost in Node D. Since there is no end-to-end transport protocol, Node J cannot assume that it should receive a transport layer acknowledgment in case of a successful end-to-end transmission of the message. Therefore, traditional end-to-end retransmission mechanisms based on timeouts cannot be used at the trans-

port layer in NetInf. Instead, Node J would have to regard the failure to return the requested set of NDOs within a timeout period as a trigger to retransmit the request.

Alternatively, Node G can also retransmit the request, since it has processed and forwarded it to Node D, and is expecting a response from this node. However, it would be a waste of resources if both Node J and Node G retransmitted the request. Therefore there is a need for a mechanism to decide which node should make the retransmission.

Now, consider the case with an unreliable per hop transport protocol. From a Requester point of view, this case is indistinguishable from the case with a reliable transport protocol per hop. A message may be lost along the path in both cases, and from a Requester's point of view it does not matter whether the message was lost due to an error in a forwarding table, an error at the convergence layer, or a transmission error not handled by an unreliable per hop transport protocol. Therefore, the same type of retransmission mechanism above the transport layer is needed both for a reliable and an unreliable per hop transport protocol.

At which layer should the retransmission mechanism be located? Assume that NetInf offers a reliable transfer service at the API. Then the retransmission should be performed under the API, either by the Message Forwarding function, the NetInf Protocol, or the Convergence Layer. However, note that HTTP does not have a timeout-based retransmission mechanism, so maybe it is better that the Message Forwarding function or the NetInf Protocol shown in Figure 3.2 handles the retransmission. Alternatively, the NetInf API does not support a reliable transfer service. Then the application would have to support a timeout based retransmission of messages.

3.2.1.3 Flow and Congestion Control

Finally we briefly discuss the flow and congestion control implications of not having an end-to-end transport protocol. Consider for example a case where Hop #1 in Figure 3.2 only offers narrowband links, while Hops #2 and #3 offer broadband links. When Node A returns a large NDO to Node J via Nodes D and G, due to the lack of end-to-end flow and congestion control, the transfer of this NDO will utilize all available resources at each hop. Assuming that sufficient resources are available at broadband Hops #2 and #3, the NDO will quickly be transferred over these hops, and will be buffered in Node G before it can be transferred over narrowband Hop #1. Therefore, Node G will need much larger buffers than a traditional IP router. For example, if Node J requests a sequence of NDOs with a total size of 10 GB, these NDOs need to be stored in Node G before they can be transferred over the narrowband Hop #1 to Node J. In general, due to the lack of an end-to-end back-pressure mechanism, each NetInf nodes will need much larger buffers than a traditional IP router.

Basically, NetInf messages will be stored and forwarded in a fashion similar to forwarding of e-mail messages between e-mail servers, where each forwarding hop is handled by a separate transport protocol instance, and the messages are stored on the servers until transfer resources of links and servers are available. The storage capacity needed in NetInf nodes to decouple the per hop transport protocol control loops for flow and congestion control is for further study.

The increased storage capacity in NetInf Nodes compared to IP routers can be used not only for buffering, but also for caching. We note some similarity between the transport approach described in this section and the Content Centric Networking approach described in [10].

Another implication of not having an end-to-end transport protocol is that the start-up time for content transfer will increase when transport protocol connections need to be established sequentially across each hop along a path. If TCP is used over all hops, the slow-start mechanism will operate separately over each hop along the path. The evaluation of TCP performance for this type of per-hop use case is for further study. TCP performance will probably improve if semi-permanent TCP connections can be set up between NetInf Node neighbours, and request-response exchanges can be aggregated on these connections.

3.2.1.4 Summary

In summary, this section has described the implications of the Convergence Layer approach without an end-to-end transport protocol for NetInf transport. Both unicast and multicast cases with NDO transfer have been described. It was noted that a retransmission mechanism above the convergence layer is needed if reliable end-to-end transport is desired. The need for increased per hop storage due to the lack of end-to-end flow and congestion control was discussed. Finally, an increased transport start-up time due to sequential cold starts of per hop transport protocols was discussed.

3.2.2 A Request Flow Control Protocol Design and Evaluation

In the following we will discuss transport protocol mechanisms for convergence layers where the NDO size can exceed the maximum packet size. A robust receiver-driven flow control protocol is responsible for realizing efficient data retrieval by controlling the receiver request rate and adapting it according to available network resources. A transport session is realized by sending queries progressively sent out by the receiver, routed towards a series of caches and then up to the permanent storage. Our protocol objectives are:

Reliability: the first goal of the flow controller is to guarantee reliable data transfers by re-expressing requests in case of data packet losses. Requests are rescheduled at the expiration of a timer τ, maintained at the receiver. In this way, the flow control protocol does not rely on losses as congestion notifications, but on delay measurements and timer expirations.

Efficiency: the second objective is to minimize the completion time of data transfers. To this aim, we employ an Additive Increase Multiplicative Decrease (AIMD) mechanism for adapting the request window size as a means to attain the maximum available rate allowed on the network path.

Fairness: the third goal is to achieve fair bandwidth allocation among flows (namely, content retrievals), sharing the same route and, thus, the same limited down-link bandwidth.

Protocol Description Data packets are identified by a unique name (the object name plus a segment identifier) and are requested via requests (GET) in the order decided by the application. A *routing protocol* guarantees that queries are properly routed towards a data repository, following one or multiple paths. Every intermediate node keeps track of outstanding queries, for a limited amount of time, to deliver the requested data back to the receiver on the reverse path. In addition, nodes temporarily cache data packets on the response path. Data may come from the repository, or from any intermediate cache along the path with a temporary copy of the requested data. Packets of the same content can therefore be retrieved in a *multi-path fashion*, i.e. from multiple locations with different round trip times, affecting the overall delivery performance. The receiver window, W, defines the maximum number of outstanding requests a receiver is allowed to make.

Request Window Increase Additive increase is performed by adding η/W to W on data packet reception. This operation is implemented using integer arithmetic so that W is incremented by η (set to 1 as default value) when a complete window of requests is acknowledged by data reception.

Request Window Decrease Multiplicative decrease is initiated by an expiring timer (*congestion signal*). The protocol reacts by multiplying W by a decrease factor $\beta < 1$, but no more than once for a duration equal to the timer as in Figure 3.3(b). Note that re-expression of a request, after a timeout, is only necessary to recover from lost data and not when data has only been delayed at the bottleneck.

Timers Properly setting the timer value τ and the node timers (which allows the removal of pending requests at each node) is crucial. In general, τ must be trivially set larger than the

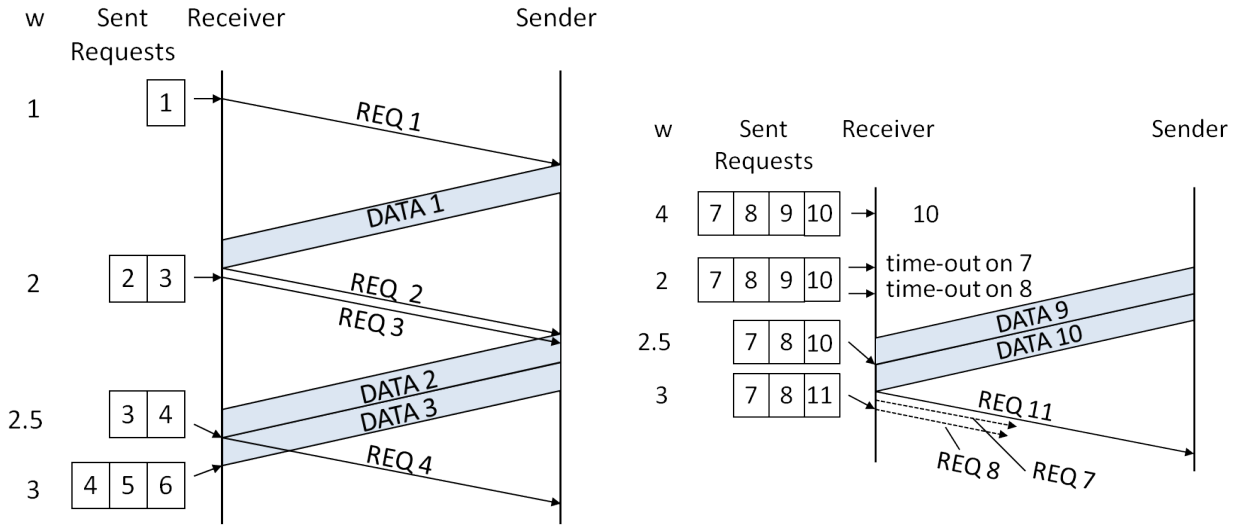


Figure 3.3: Trellis diagram for window evolution: increase (left), decrease (right).

minimum network delay, otherwise every request would trigger a timer expiration. In addition, a minimum τ value is necessary to guarantee high utilization of the available bandwidth. A trade-off between a large τ value for efficiency reasons and a small τ value for faster window adaptation, is given by an adaptive timer based on the estimation of the minimum round trip delay, as is commonly described in TCP literature. Therefore, in the case of variable τ, the protocol maintains round trip delay estimates at every data packet reception, by updating $\mathrm{RTT}_{\min}$ and $\mathrm{RTT}_{\max}$ averaged over a history of samples (20 data packets in our implementation), excluding retransmitted packets. Each flow adapts its timer τ according to the rule:

$$\tau = \mathrm{RTT}_{min} + (\mathrm{RTT}_{max} - \mathrm{RTT}_{min})\delta \tag{3.1}$$

with $\delta = 0.5$. Also, in our setting, we use an initial value of $\tau = 10$ ms and $\mathrm{RTT}_{max} = 2\mathrm{RTT}_{min}$ initially after the first measured round-trip delay.

The node timers are fundamental to limit the number of pending queries to be managed in every intermediate node. The larger this value the higher the number of filtered requests, not forwarded upstream towards the content repository. On the other hand, a small node timer implies a large number of unnecessary retransmissions, since delayed packets arriving after node timeouts are discarded.

Design Guidelines and Protocol Assessment The choice of the parameters can be critical and can potentially lead the system to an inefficient regime. The present section provides guidelines for tuning protocol parameters, i.e. η, β and τ.

Parameters Setting The *additive increase parameter* η, is set by default to 1, corresponding to an request window increase of one packet per round-trip time. By increasing the window size more aggressively ($\eta > 1$), a flow can achieve higher throughput, if and only if bandwidth resources are not fully utilized. However when bandwidth is efficiently utilized η affects only the limit cycle duration and not the average rate. Similar considerations hold for the *multiplicative decrease parameter* β, set by default to 0.5, such that the request window is halved upon the expiration of the timer. As for η, a smoother window decrease, setting $\beta \geq 0.5$, results in higher flow throughput as long as bandwidth is not fully utilized. In case of efficient bandwidth sharing, β affects the limit cycle duration and not the time average.

The most critical parameter is the *Request retransmission timer* τ, that must be chosen not smaller than the minimum network delay to have non zero throughput, and large enough to queue at least the bandwidth delay product along the path to have 100% utilization. Moreover, an arbitrarily large τ would let the protocol converge very slowly, which motivates the choice to adaptively set τ according to Equation (3.1).

Simulations In this section we assess protocol performance in some simple scenarios, comparing simulations to the optimal working point calculated in [23]. Such working point guarantees that bandwidth is fairly shared among parallel flows, in the max-min sense. We write with $\gamma(i)$ the fair rate for route i.

Example: A Three Links Network Path Consider the case of a single path composed by three hops over which content is relayed to the user from the repository. We distinguish the following cases, given that the number of ongoing flows on route i is N_i, $i = 1, 2, 3$, where $\gamma(i)$ vary according to C_i/N_i values.

- Case I: $C_1/N_1 < C_2/N_2, C_3/N_3$; $\gamma(1) = \gamma(2) = \gamma(3) = C_1/(N_1 + N_2 + N_3)$.
- Case II: $C_2/N_2 < C_1/N_1, C_3/N_3$; $\gamma(1) = (C_1 - C_2)/N_1$, $\gamma(2) = \gamma(3) = C_2/(N_2 + N_3)$.
- Case III: $C_3/N_3 < C_2/N_2 < C_1/N_1$; $\gamma(i) = (C_i - C_{i+1})/N_i$, $i = 1, 2$, $\gamma(3) = C_3/N_3$.
- Case IV: $C_3/N_3 < C_1/N_1 < C_2/N_2$; $\gamma(1) = \gamma(2) = (C_1 - C_3)/(N_1 + N_2)$, $\gamma(3) = C_3/N_3$.

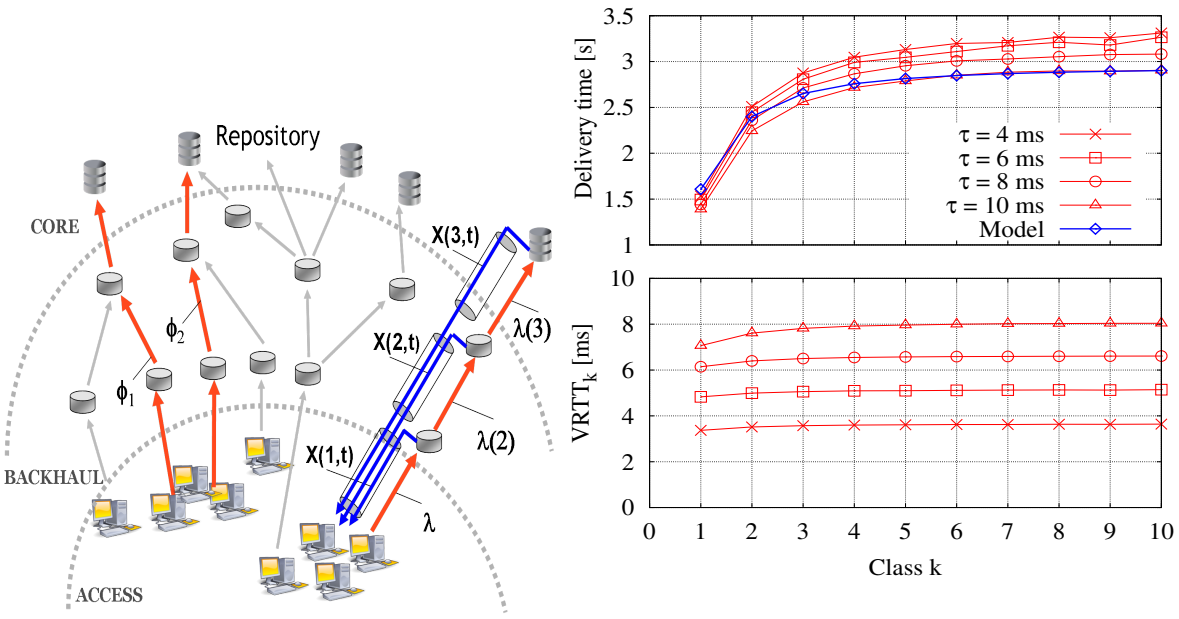


Figure 3.4: (a) Topology, (b) Delivery time and RTT_k in case II with capacity (C) C1=C3=100Mbps, C2=40Mbps with fixed τ

Constant τ: We consider a linear network with tree links in a multi-bottleneck configuration, assuming negligible propagation delays; Content items of class k are requested according to a Poisson process of intensity $\lambda_k = 1 \times q_k$ content/s, with $q_k = \frac{1}{m}\frac{c}{k^\alpha}$, where $m = M/K = 5$ is the number of content items in the same class, and $\alpha = 1.7$ the Zipf parameter. The Zipf popularity distribution is given by $q_r = r^{-\alpha}$ for large r and exponent $\alpha > 1$. A content item has fixed size $\sigma = 5,000$

packets (packet = 1kB) and cache sizes of 50,000 packets. Figure 3.4 reports the average delivery time (top) and virtual round trip time, VRTT_k (bottom), measured for content retrievals in the ten most popular classes, $k = 1, ..., 10$, for different values of τ. VRTT_k is shown to be proportional to τ in Figure 3.4 (bottom). The measured delivery time is compared to the optimal reference curve, indicating full utilization and fair allocation of resources; moreover it is shown to be negatively affected by a too small τ value, especially for classes $k > 1$, since longer routes r_2, r_3 (of two/three hops) does not fully utilize the bottleneck capacity C_2. Conversely, the most popular data transfers ($k = 1$) get slightly better performance than what predicted, as the τ value is well set for the shorter route r_1 they mainly utilize. We argue that each data transfer would benefit from adapting τ to the measured delay from used routes.

Variable τ: In Figure 3.5 we present a set of simulations, where each receiver estimates the value of τ, by implementing the algorithm described in Eq.(3.1). Globally, the queuing delay converges to a much smaller value than that obtained using a constant τ, as a consequence of the fact that τ_k, converges to value that is slightly larger than RTT_k, function of the popularity class and, hence, of the data distribution along the path.

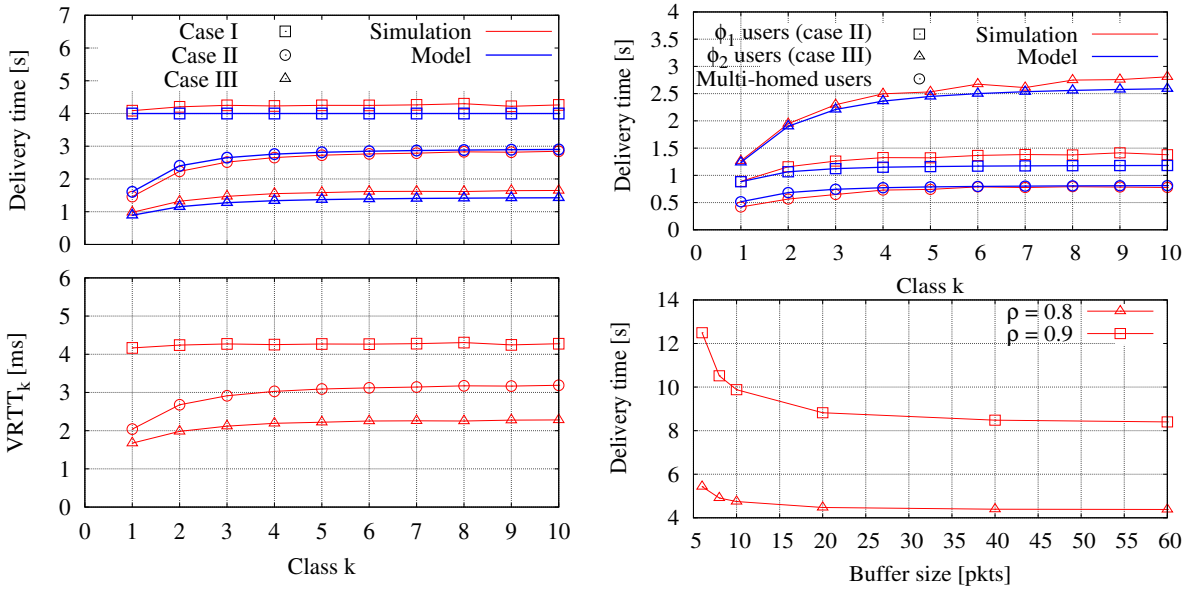


Figure 3.5: (a) case I,II,III with adaptive τ (b) Multi-Homed network

Multi-homed scenario: Routes are built following the reverse path of the requests, routed towards the closest hitting cache. Every receiver can easily exploit multiple forwarding interfaces by load balancing requests on the different interfaces. Let us consider a network topology with multiple repositories, where one group of users is multi-homed, namely it uses two disjoint paths, ϕ_1 and ϕ_2, with $C_1^{\phi_1} = C_2^{\phi_1} = 100Mbps$, $C_3^{\phi_1} = 60Mbps$ and $C_1^{\phi_2} = C_3^{\phi_2} = 100Mbps$, $C_2^{\phi_2} = 60Mbps$. *Multi-homed* users share bandwidth resources with other *single path* users exploiting ϕ_1 or ϕ_2 only. Results are reported in Figure 3.5 (b, top). A good agreement can be observed between simulation results and analytical predictions both in the single and multi-path case. Moreover, as expected, RTT_k and delivery time experienced by *multi-homed* users, are significantly reduced w.r.t. those perceived by ϕ_1/ϕ_2 *single path* users as a consequence of the higher throughput achieved by using both path in parallel.

Buffer Sizing: We have considered so far, networks with unlimited buffer capacity. System dynamics in a small buffer regime have not been analysed due to lack of space, for completeness,

we report in Figure 3.5 (b, bottom) the average delivery time measured in a single bottleneck scenario at two loads, 0.8, 0.9, under different buffer sizes. The loss of performance due to little buffering can be significant. However, we believe that there is no clear advantage in saving output buffer memory in networks of caches where a loss virtually never happens thanks to the caching functionality.

3.3 Content Placement

In-path network caching of data has been researched from the early 90's starting with web server caching to more recent work in reducing traffic in 3G networks and data amounts between AS's. Companies such as Amazon, Limelight and Akamai have deployed intelligent caching schemes within the Internet. As these companies' services show, caching can be successfully deployed in today's Internet. In this upcoming section we extend the ideas of caching to NDOs. We believe the NetInf philosophy can leverage the usefulness of intelligent caching much more successfully than the end-to-end transport that predominates today, however we need to investigate what performance gains and mechanisms are needed. These are explained next.

3.3.1 Introduction to Caching Concepts

Some of the required control and policy interfaces are specified to make deployment models of the NetInf functions possible including the caching policies. From a topology perspective we focus on a hierarchy of caches, which aggregates links up a tree to a small number of servers at the root. This type of structure is commonplace in Video on Demand (VoD) systems. Trailers or complete programmes stored within the caches of the distribution tree can potentially reduce the load on servers, reduce access times for clients and importantly increase the Quality of Experience (QoE) for the end user. The cost is extra infrastructure in the network, where elements not only forward data, but also perform store and forward data operations. Splitting the client-server retrieval model also adds extra complexity to the operational side of running a content distribution network using an ICN approach. The control and policy interfaces of NetInf key functions are introduced. The philosophy is to assess the placement of the cache at the most effective level. Real world values are chosen for fan-outs, price per storage and sizes. The second work looks at the replacement algorithms for caches also in a hierarchical structure. A key result is that the simplest eviction policy can be used when very fast speeds are needed at cache interfaces. The third work looks at how to move NDOs closer to the point of consumption, given the requests are stored within the metadata of each object. The final work in this section looks at collaborative caching with the GIN ICN framework.

3.3.2 Management and Control of Content

Figure 3.6 represents the different entities that are relevant from a control perspective. For NetInf, there are two roles, one for the client side and another for the infrastructure side. Both roles naturally require that the NetInf protocol is implemented. Two different kinds of caches are considered; A NetInf cache and a standalone cache such as is defined in the Decoupled Application Data Enroute (DECADE) draft. For the client side, only a partial control interface is supported. The control interface can be divided into two parts a i) Primitive interface and a ii) Policy interface. The former is used between caches and NetInf (Infra) nodes and it is used to carry out content specific actions and the latter is for adjusting policies and reporting resources. It should be noted that when a NetInf (Infra) node implements its own caching functionality, then this control interface between NetInf and cache is more like a logical interface inside the NetInf (Infra) node.

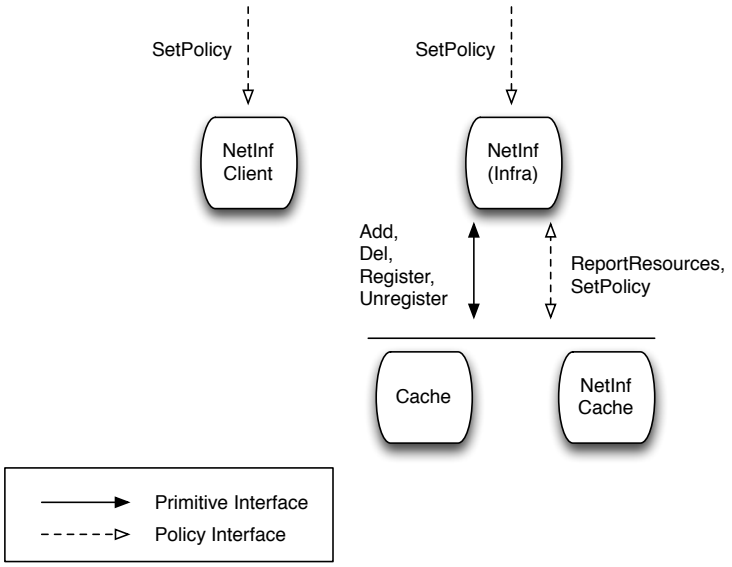


Figure 3.6: Cache control interface

3.3.2.1 Cache Control

Algorithm control: A NetInf or other control entity should be able to specify what type of algorithm (e.g., Forward Meta Data (FMD) (see Section 3.3.5), Least Frequently Used (LFU), Least Recently Used (LRU) or Random (RND) policy) and with what parameters a cache should implement and use. Typically this is a selection of available already installed cache algorithms and parameter sets. For NetInf nodes to know the alternatives a cache report is needed/checked, see shown next.

Content specific primitives: These are primitives are based on more sophisticated procedures that can be implemented using NetInf mechanisms and the cache control. *Add* and *Del* are primitives by which NetInf or instructs the cache(s) to either fetch and cache an NDO (*Add*) or evict the existing NDO (*Del*). *Register* and *Unregister* are primitives used by cache(s) to register new cached NDOs to NetInf and unregister the content copies which are to be evicted. The *Add* primitive has two different uses; i) fetching a NDO and serving it to a user and ii) fetching a NDO and caching it without sending it any further, i.e., replication (see Section 3.3.5) of an NDO.

Cache reporting: caches report their generic resources such as free storage space, CPU load as well as content specific statistics such as measurements related to content popularity.

3.3.2.2 Scoping

The ability for the NetInf node to use cache control methods requires that the cache registers with a NetInf node. This can be done with the cache ReportResources method. The available cache capabilities can be assigned specific scope, this further makes specific data possible on particular caches. Different kinds of cache behaviour can be deployed for each piece of scoped data by controlling the policy of cache operation (e.g., algorithm and threshold parameters, see the section on cache control options above).

3.3.3 On the Effectiveness of In-Network Caching

We have developed an analytic model of the *average* performance of the system as function of a few key parameters. Given a probability distribution for the content request rates, an eviction policy for the local caches and a particular network topology, we would like to determine the cache

hit rates and content availability throughout the whole network. In order to do this we study the system (in the limit) where the number of requests is so large that we can approximate the arrival process of discrete requests with a continuous *flow* which is specified as a rate of requests per unit time. This approach is similar to fluid analysis of queuing systems [24]. Our analysis then operates on distributions of these flows rates rather than on discrete request events.

Rather than considering arbitrary network topologies our model is restricted to trees of routers of constant degree, i.e. the fan-out below a router is constant throughout the tree except for the leaf routers that have no children. Only the leaf routers have hosts connected and as their children. We model the system as follows: Starting at the leaf routers we have flows of requests coming in. The requests are distributed over the flows according to the given probability distribution. Based on the eviction policy[1] we calculate the probabilities for items to be cached. Then we remove the cache hits from the incoming flows and aggregate what remains from the sibling flows (i.e. the residual requests). The resulting flows correspond to the cache miss traffic and this is in turn passed up to the next level router. After this the process repeats until we reach the root. A more detailed and mathematical description of the model can be found in [25].

3.3.3.1 A Realistic Example

Next we will apply the previously described model on a fictitious access aggregation network. The network will consist of three levels of routers each with a fan-out (degree) of 10. The topmost (root) router is directly connected to the operators metropolitan or core network and the remaining routers are connected below the root. A possible interpretation of this sample network is as 100 pools of Digital Subscriber Line Access Multiplexer (DSLAM) equipment that in turn is connected by 10 Broadband Remote Access Servers (BRAS) that connects to the root router. Each router is equipped with 360 Gbytes of flash memory (motivation later in this section) to be used for in-network caching. For simplicity all routers is equipped with the same amount of cache memory.

Orange Labs has written a report about video media accesses [26] that we will take parameter values from. One of the traces in the report describes accesses to a mixture of movies and trailers during an 8 day period. The average content size in this trace is 703 Mbytes which with a 360 Gbyte cache corresponds to a cache capacity of 512 items. The values are summarized in the table below.

Duration	8 days		
Requests	29,707	5,199 clients	5.7 reqs/client
Catalogue	3,518 items	2,473 GB	703 MB/item

With these parameters as input to the analytic model we calculate a hit rate of 45% at the routers closest to the terminals. At the next level hit rate will be 26% and at the top 22%. From the graphs of the Orange report a simulated LRU cache of the same size at the first level shows a better than 60% hit rate on real data. These percentages are conservative as our model assumes accesses to be independent. In practice, accesses are not independent and real caches are designed to make benefit from accesses' correlations. Our cache model does not take correlations into account and real hit rates are thus expected to be better.

The results (hit rates) of the calculations are summarized for five different sizes of caches below.

Level	128GB	256GB	360GB	512GB	768GB
root	0.130	0.184	0.218	0.260	0.323
mid	0.157	0.223	0.264	0.315	0.386
leaf	0.243	0.376	0.448	0.523	0.606
aggr.	0.444	0.604	0.683	0.758	0.836

[1] Currently we have derived models for LRU and LFU.

An important question to consider is if flash memory will provide enough capacity to satisfy all hits. For reference we have picked a standard PCI-Express card with inexpensive flash memory that costs 700 EUR. It provides for 540 Mbytes/s of sustained read capacity. From the trace we can deduce an average VoD rate of 30 Mbyte/s (240 Mbit/s) of which 13.5 Mbyte/s should be read from our cache. In our experience, peak VoD load can be as much as 5 times the average load. In this case there is sufficient capacity to satisfy the demand. At the next level, 44 Mbyte/s will have to be read from the cache (remember to multiply by 10 because we are aggregating 10 subtrees). There is still plenty of capacity. Similarly, at the root level we must read data from the cache at an average rate of 267 Mbyte/s. This is possible, but if we expect peak load to be as much as 5 times the average load we need to use flash memory with higher read performance. Such memory exists and considering that this must be a rather high end router anyways this additional cost can likely be motivated.

In June 2011 Cisco reported [27] *Internet video is now 40 percent of consumer Internet traffic, and will reach 62 percent by the end of 2015, not including the amount of video exchanged through Peer-to-Peer (P2P) file sharing. The sum of all forms of video (TV, VoD, Internet, and P2P) will continue to be approximately 90 percent of global consumer traffic by 2015.*

Consequently, introducing the comparatively small caches mentioned above should already today save some 27% of all consumer Internet traffic and extend to 42% savings by the end of 2015.

3.3.3.2 Other Observations from the Analytic Model

Next we solve the equations for a tree of degree $d = 10$. Requests will be Zipf distributed using $\alpha = 0.7$. Figure 3.7 shows the hit rate of individual levels for a very deep tree network of 10 levels and cache sizes corresponding to 0.1% of the catalogue[2]. We note that the first level (leaf) cache is of highest importance. Caches at higher levels can still contribute to the aggregate cache effect but only to a much lesser extent. Thus, when designing future in-network caching architectures one should consider using either larger caches at higher levels or to make groups of caches collaborate to create a larger virtual cache (paying the internal communication costs but increasing content availability). This holds for the whole parameter range but the relative performance of level 1 decreases with catalogue size. This is intuitive since for very large catalogue sizes each level in the cache will store very popular items and will have relevance on performance.

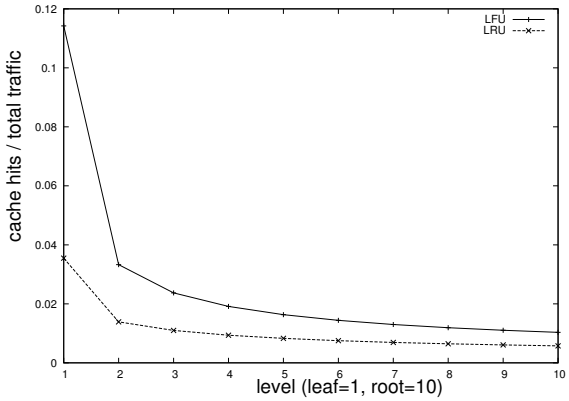


Figure 3.7: Hit rate for various levels in the cache hierarchy.

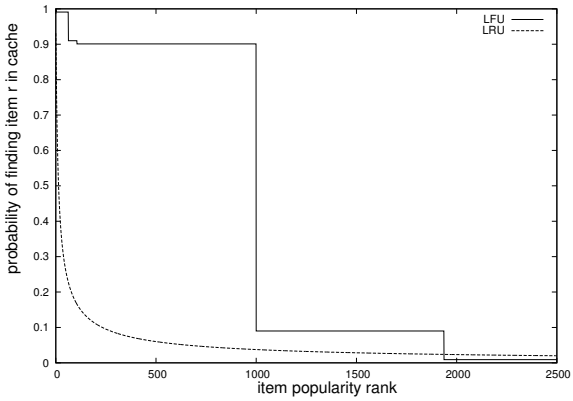


Figure 3.8: Comparison of which items are cached by LRU and LFU by rank.

Next we examine what items will likely be cached by the two eviction policies. A rank plot of popular items and the probability of finding them cached is given in Figure 3.8. Here we see the

[2]Catalogue is the total set of data offered and 0.1% can be considered conservative (see Section 3.3.3.1).

difference between LRU and LFU clearly. LFU caches *only* the most popular items (head of the distribution) whereas LRU will cache all items with a non-zero probability. This can have important implications on collaborative caching schemes. If we make caches collaborate, LFU would still have to request many items from outside of the tree whereas LRU could often find the item locally. Thus, if one considers to implement collaborative caching LRU might be a better option. The random (RND) replacement policy (see Section 3.3.4) may also be a good candidate if one intends to improve availability by introducing collaborative caching.

3.3.4 Analytical Analysis of the Random Policy

In this section, we analyse the miss (or hit) ratio of a cache using the Random replacement policy. Detailed results and proofs derivations can be found in the full paper [28]. The RND policy is known to perform worse than LRU, but we show here that the theoretical difference is not large. The main advantage of RND is that it is the simplest possible policy, requires significantly less operation for cache index management than any other policy: this advantage can be crucial on the feasibility of high-speed caching. A second possible advantage of RND is that it avoids the need of explicit coordination about which NDOs should be expunged, which can be prohibitive for large collaborative caches.

Single Cache Consider a cache memory of finite size which is offered requests for objects. The object replacement policy is assumed to follow the RND discipline, *i.e.*, whenever a miss occurs, the object to be replaced is chosen at random, uniformly among the objects present in the cache. Given the total number N of objects, the probability that object r, $1 \leq r \leq N$, is requested is defined by q_r, $1 \leq r \leq N$. Let $C \leq N$ be the cache capacity and denote by $M(C)$ the stationary miss probability; finally, define $M_r(C)$ as the per-object miss probability, given that the requested object is precisely $r \in \{1, ..., N\}$. A general combinatorial expression of $M(C)$ has been given in a paper from Gelenbe ([29], Theorem 4) for any popularity distribution.

We here provide asymptotes for probabilities $M(C)$ and $M_r(C)$ for large cache size C. We can show that the miss probability $M(C)$ is asymptotic to

$$M(C) \sim \rho_\alpha C q_C \tag{3.2}$$

for large C, with prefactor $\rho_\alpha = \left(\frac{\pi/\alpha}{\sin(\pi/\alpha)}\right)^\alpha$. Furthermore, given the object rank r, the per-object miss probability $M_r(C)$ is estimated by

$$M_r(C) \sim \frac{\rho_\alpha r^\alpha}{C^\alpha + \rho_\alpha r^\alpha}. \tag{3.3}$$

Figure 3.9a depicts $M_r(C)$ as a function of the object rank r for both RND and LRU policies with fixed $C = 25$ and $\alpha = 1.7$. Numerical results confirm the asymptotic accuracy of estimate Equation (3.3) for RND and the corresponding one for LRU policy from the paper from Jelenkovic [30] when compared to simulation. Besides, RND and LRU performance are very close for large enough r.

In-network Cache Model We generalize the single-cache model and consider a *homogeneous tree* network, where all leaves are located at a common depth from the root, and the cache size of any node at a given level ℓ is equal to C_ℓ. Requests for content are routed towards the root of the network, until they experience a hit at some cache. Any request experiencing a miss at some cache of level k, $1 \leq k \leq \ell$, and an object hit at level $\ell + 1$ is copied backwards to all downstream caches on the request path. A request miss corresponds to a miss event at all levels. We assume that **(H)**

any cache considered in isolation behaves as a single cache with IRM input produced by consecutive missed requests originated by its children nodes.

For any level $\ell \in \{1, \ldots, K\}$ and cache sizes $C_1, \ldots, C_\ell$, let $M_r(C_1, \ldots, C_\ell)$ denote the "global" miss probability for request r over all caches of a route through levels $1, \ldots, \ell$. Then for large C_1, ..., C_ℓ,

$$M_r(C_1, \ldots, C_\ell) \sim \frac{\rho_\alpha r^\alpha}{\rho_\alpha r^\alpha + \sum_{1 \le j \le \ell} C_j^\alpha}. \tag{3.4}$$

Figure 3.9b reports miss probability $M_r(C_1, C_2)$ at the second level of a two-level binary tree, *i.e.*, the probability to query an object of rank r at the repository server. We note that objects with small rank are slightly more frequently requested at the server when using RND rather than LRU, but RND is more favourable than LRU for objects with higher rank. We also be observe that the miss probability at the second level is very similar either using RND or LRU, with a slight advantage to LRU. We notice that the approximations calculated in this section for RND and in a paper from Muscariello *et. al.* [31] for LRU are very accurate.

Mixture of RND and LRU This section addresses the case of a two-level tree network, where the RND replacement algorithm is used at one level while the LRU algorithm is used at the other. As in the previous paragraph, these results also hold in the case of a homogeneous tree.

I) When level 1(resp. level 2) uses the RND (resp. LRU) replacement policy, the global miss probability at level 2 is given by

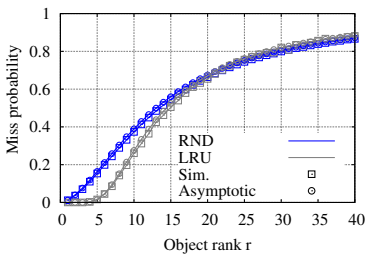
$$M_r(C_1, C_2) \sim \frac{\rho_\alpha r^\alpha}{\rho_\alpha r^\alpha + C_1^\alpha} \exp\left(-\frac{\rho_\alpha C_2^\alpha}{\alpha \lambda_\alpha \left(\rho_\alpha r^\alpha + C_1^\alpha\right)}\right) \tag{3.5}$$

for large cache sizes C_1, C_2 and constants ρ_α, λ_α.

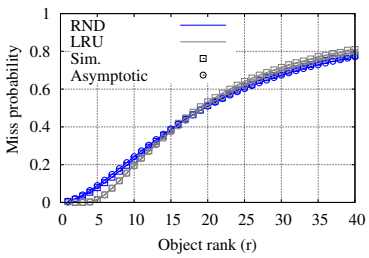
II) When level 1 (resp. level 2) uses the LRU (resp. RND) replacement policy, the global miss probability on level 2 is given by

$$M_r(C_1, C_2) \sim \frac{\rho_\alpha r^\alpha}{\rho_\alpha r^\alpha \exp\left(\frac{C_1^\alpha}{\alpha \lambda_\alpha r^\alpha}\right) + C_2^\alpha}. \tag{3.6}$$

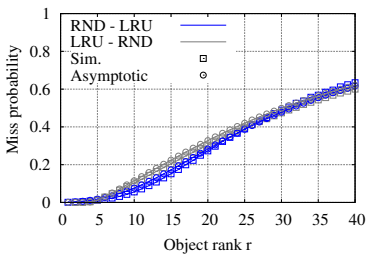
In numerical experiments, we have simulated tree networks with 2 leaves with cache size C_1 and one root with cache size C_2. In Figure 3.9c, we observe that the total miss probability $M_r(C_1, C_2)$ in the two mixed tandem caches is similar, while the distribution of the objects across the two



(a) $M_r(C)$ for a single cache with $C = 25$, $\alpha = 1.7$ for RND and LRU policies.



(b) $M_r(C_1, C_2)$ for RND and LRU policies compared to simulation for a two-level tree network with $C_1 = C_2 = 25$, $\alpha = 1.7$.



(c) $M_r(C_1, C_2)$ for a mixed tree cache networks with $C_1 = 25, C_2 = 50$, $\alpha = 1.7$.

Figure 3.9: Asymptotes and numerical simulation results of the random policy

nodes varies considerably. Further numerical experiments suggest to prefer LRU at the first cache because it performs better in terms of miss probability, and to use RND at the second one in order to save significant processing time.

Summary The performance of RND being reasonably close to that of LRU, RND is a good candidate for high-speed caching when the complexity of the replacement policy becomes critical. In the presence of a hierarchy of caches, caches at deep levels (i.e. access networks) typically serve a small number of requests per second which can be sustained by a cache running LRU, thus providing the best performance at the bottom level. Higher-level caches that serve many aggregated requests should use the RND policy, which yields similar performance while being less computationally expensive.

3.3.5 Replication and Migration of NDOs

As well as the commonly used performance metrics such as object hit rate, bit hit rate for a single cache, in a hierarchical structure, the number of hops or levels to locate an item is important. In a NetInf environment where caching will be exploited more than in today's Internet, we take into account this factor.

A problem with the LFU and LRU (as described previously) is that these policies tend to evict potentially popular content before they have an opportunity to grow popular when storage space is small compared to the amount of NDOs visible to the cache. To mitigate this effect caches higher up in the hierarchy are provided with larger storage space. This, however, works to a limited extent since the cache size grows and the latency for searching the cache becomes prohibitive. Ideally, the most popular content should be stored in the cache closest to the client while the next popular content is stored at the next level in the cache hierarchy and so on. Furthermore, one could argue that once popular content has been distributed out to the caches close to the edge of the network this content could be expunged from caches higher up the hierarchy to provide storage space for new content.

Thus, rather than seeing the cache management policy solely as an eviction policy, one should consider the performance of a *migration* policy. That is to hold only the NDOs which have a number of accesses greater than those expunged from the cache. By holding only the NDOs in caches within the hierarchical tree structure the NDOs with the appropriate hit rates are held at each level as mentioned. This should lead to reductions in the access times, path lengths and traffic in a distribution tree. Given the complexity of the problem, and the lack of analytical models (however see Section 3.3.3) we built a custom C++ simulator to explore the properties of the hierarchical caching issue in an ICN context. The simulator supports LFU, LRU and RND eviction policies as well the proposed cache management policy called Forward Meta Data (FMD). The process has two phases, first clients shown located at the leaves of the tree in left plot of Figure 3.10 request random NDOs with a probability given by the Zipf law described in Section 3.3.4.

Once found, the second phase is on the return path to the client, the metadata according to the NDO is compared with the metadata of the NDOs that are cached. We assume that the (reverse) response path is the same as the (forward) request path issued by the client. If the popularity of the newly requested NDO is higher than the least popular content in the cache the NDO is replaced with the newly arrived NDO. The popularity of an NDO i is calculated as:

$$r_i = \frac{n_i}{N}$$

Therefore the process compares r_i for each NDO with those in the intermediary caches. Just to be clear, the metadata field containing the hit rates for the NDOs are compared. n_i denotes the number of requests for NDO i and N the total number of requests during the measurement period

Table 3.1: Average rank of document in memory at each level in cache hierarchy, 1000 000 documents and cache memory size 100 documents (α =1)

	LFU	LRU	RND	FMD
Level 1	43428	98729	75171	183
Level 2	62765	98357	95019	164
Level 3	75756	84124	83547	145
Level 4	84159	80345	80459	80

for each cache. Note care must be taken that NDOs are not of the same size and a number of NDOs must be replaced in order to provide storage space for the arriving NDO.

Figure 3.11 shows the mean number of hops before a cache hit for the NDO in a five level cache hierarchy as a function of relative cache memory size in each node (i.e. cache memory space divided with the sum of all information objects). The plot shows a clear advantage of using FMD compared to LFU and LRU. Naturally RND performs the worst, but maybe the fastest to implement as was shown. To study how FMD perform for different values of the parameter α the number of hops before cache his is plotted as a function of α is shown in Figure 3.12. As can be seen in the plot FMD outperforms both LRU and LFU for a wide variety of α. As mentioned a desirable property of any hierarchical caching system is to keep the most popular items closest to the requester and less popular content further away from the requesting clients. Table 3.1 list the average rank of NDOs at each level for each cache replacement policy. From the table we can see that FMD is better at storing more popular NDOs close to the clients than the other policies.

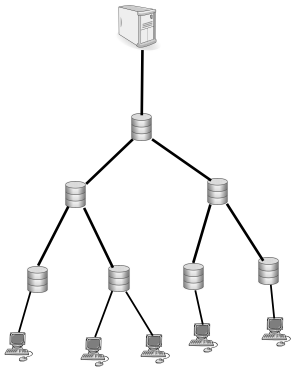


Figure 3.10: Cache tree structure

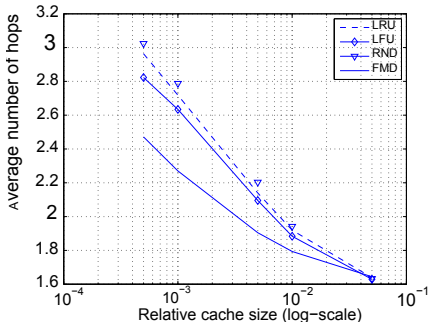


Figure 3.11: Hop count

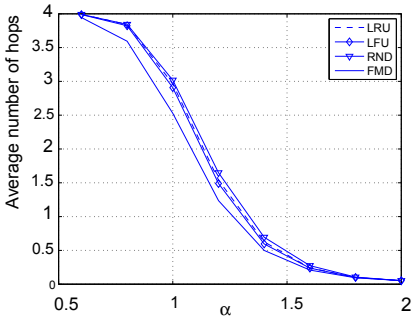


Figure 3.12: Popularity factor

Summary Our initial experiments indicate that FMD outperform both LFU and LRU. However, with an analytical model of that presented FMD shows a reduction in the number of hops or links that need to be traversed. That is, even in a small hierarchical caching scheme, Additional experimentation is underway to look at fan-outs which are greater than 2 (the current results are based on a binary tree only).

3.3.6 A Study of the Performance of Collaborative Caching in an ICN Environment

GIN (Global Information Network) is an ICN proposal for the Network of Information, described in detail in [21]. It is a hybrid architecture that is able to support both dissemination and conversational communication models. GIN aims to interconnect NDOs over heterogeneous L3/L2

underlayers, in the global network, by means of an integrated name-based resolution and routing mechanism. Data are routed by names into the GIN network on a double path: the resolution path is used to route NDO requests to destination(s) through a chain of Dictionary nodes arranged according to a Multi-level hierarchy of DHTs (Multilevel DHT (MDHT) [20]). Each object request initiates a communication session between the requesting entity and the object source(s). Data in the communication session are routed on the shortest path with fast lookups in forwarding tables. Caching of data is performed in the GIN network in the MDHT infrastructure. That is, caching is performed in the resolution path and not in the fast data path. In this section we review and discuss results on caching provided in detail in [21].

Network nodes may cache NDOs autonomously, in a non-collaborative way. Non-collaborative caching, however, implies an inefficient usage of network storage resources. In fact, the same object can be cached in closer caches and the overall storage of a cluster of cache nodes contains a smaller number of objects. This is also what happens in delivery systems which perform trivial on-path data caching: an object is cached all the way through the delivery data path. Such usage of network storage resources can be non-optimal, according to our analysis, because it reduces the cache hit ratio in a network and increases its average latency and off-net traffic load!

One way to do collaborative caching is to register cached objects into the Name Resolution System (i.e. Dictionary nodes). However, such a choice implies an increased burden of registration traffic. Another more efficient way to implement collaborative caching is to exploit the MDHT infrastructure of Dictionary nodes. By leveraging the MDHT hierarchy, we can define a certain level of the hierarchy as the caching level. For example, in a MDHT network configured with three DHT levels (access node level, POP DHT level, Autonomous System (AS) Domain DHT level), it can be useful to cache objects at the Point of Presence (POP) level, that is, to distribute objects in the caches of nodes belonging to the same POP, as in Figure 3.13. This is a reasonable assumption, because, most of the time, the nodes of a same POP are collocated in the same building or local area network. Hence, a network POP can be seen as a cluster of cache nodes. If, in our example, the POP is the level of caching, then decisions for object caching are taken at that level and not at the first access node level. In fact, in MDHT, if an access node cannot resolve a request for object X in its local cache (or in the attached hosts), it forwards the request to the next Dictionary node in the MDHT hierarchy responsible for key X at the POP DHT level. Since the POP level is also the caching level, either a copy of X already exists in the local cache, or a caching decision for X can be taken. In this collaborative caching example, all nodes still take decisions autonomously, but the caching decision is deferred to the caching level of the MDHT hierarchy. The result is that data objects are cached in the distributed cache of the POP with a great saving of POP storage resources and/or a higher hit ratio, and without the need of an explicit registration of cached objects in the Dictionary.

Another possibility given by the MDHT architecture is to take different caching decisions at different levels. In *multilevel caching*, it is possible to divide the node cache space in two or more partitions, with each cache portion allocated to a different MDHT level. In such a scenario, we expect the most frequently requested objects to be available in the access node or POP caches, while other less frequently requested objects can be found in the AS cache.

The obvious question is where to place the caching level in a MDHT network, to get the greatest benefits. The MDHT cache system can be adapted to the size and topology of the network. The operation of the cache system may be tuned in order to get the best tradeoff between the usage of the network storage resources, the latency of the service and the throughput on external transit links.

Throughput and latency analysis have been performed for the GIN/MDHT caching system in [21]. The MDHT-based cache system has been modelled in an idealistic scenario, as shown in [32], where the object accesses are independent and the objects reference probability follows a Zipf-like

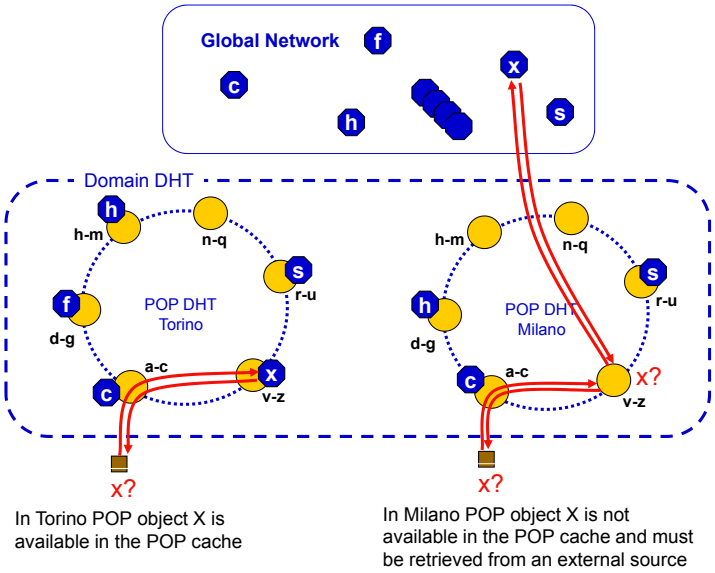


Figure 3.13: Collaborative caching with MDHT at the POP level

distribution. Paper [32] suggests values for the Zipf parameter a to be in the range between 0.6 and 0.85. In particular, for traces analysed from homogeneous environments the Zipf parameter a seems to be centred around 0.8, while systems with more heterogeneous clients show a smaller Zipf parameter a, around 0.7. The popularity ordering is determined by the request frequencies for each object. Given that the access frequencies to data objects seem to be weakly or not correlated with the size of the objects and with their modification rate [32], it is possible to use the formulas suggested in that paper to model the ideal case in which a cache system holds the C most popular objects, with the ideal assumption that the page removal policy is a perfect-LFU (Least Frequently Used). Alternative approaches to caching have been discussed in Section 3.3.1.

In order to perform an analysis of a MDHT caching system, we also need a characterization of the ICN traffic. Unfortunately, the unique characterization available is for the Internet traffic. So, in our analysis, we arbitrarily assume GIN traffic to be analogous to IP traffic. We have extracted main traffic statistics from IP traffic collected by CAIDA at the Equinix datacenter in Chicago from a backbone 10 Gigabit Ethernet link of a Tier 1 Internet Service Provider (ISP) between Chicago, IL and Seattle, WA in the USA. The statistics were collected in the week July 10th-17th, 2011. The throughput per client in the peak hour is about 100 Kbit/sec today in main ISP networks. In one Mbit of traffic, there are about 14 data flows, each composed of 18 packets on average. The mean size of an IP packet is 507 bytes. For the GIN traffic, we just increase the bytes/packet from 507 to 600 bytes, in order to consider the larger overhead introduced by the GINP header with respect to the IP header. Each data request is composed on average of 18 packets/flow, where the first request packet is a GET message sent through the resolution path and the following data packets are sent over a fast data path (with only fast lookup forwarding). We assume the GIN throughput per client to be about 1.2 Mbit/sec in the peak hour, i.e. about 12 times the peak load of an IP client in 2011. For the average object size, we make a rough hypothesis that each flow corresponds to a NDO. Therefore, with an average payload of about 470 bytes, the average object size is 8460 bytes (i.e. 470x18).

With such a characterization of the GIN traffic and the above assumptions on the object reference distribution, in [21] we have analysed the performance of a GIN MDHT-based infrastructure used as a collaborative caching system. In a three level network (Access Node, POP, AS), caching can be performed at the access node level, or may be deferred to a higher level. If caching is performed

at the access node level, however, we have a non-collaborative system, because any access node performs caching autonomously, and it is not possible to exploit other close caches which may hold a copy of a desired object.

Figure 3.14 shows the hit ratio of a three level network, where the number of POPs is approximately the square root of nodes (e.g. 10 POPs for a network with 100 nodes). Each node is equipped with 5 TB of cache space. The network hit ratio is plotted for different caching strategies: "Node Cache" means that caching is performed at the access node level, "POP Cache" at the POP DHT level and "AS cache" at the AS DHT level. The strategy "POP/AS(x/y)" means that x% of the node cache space is used at the POP DHT level, while the remaining y% (x+y=100) is used at the AS level. As one can see, the worst option is to perform caching at the access node level, that is, in a non-collaborative way. With 5TB of cache space, only a 10% hit ratio is achieved in a network, independently from the network size. If a collaborative approach is adopted, the hit ratio grows significantly with the network size, up to 100% in the largest networks with the AS level caching (that is, the entire cacheable digital universe can be stored in the network storage system). The multilevel POP/AS(x/y) strategy is an option which is, of course, halfway between POP and AS caching. The advantage of the multilevel approach, if any, can be only in terms of packet latency, and reduced internal backbone load. The network hit ratio, and therefore, the ratio of on-net traffic, is maximized with the "AS Cache" strategy.

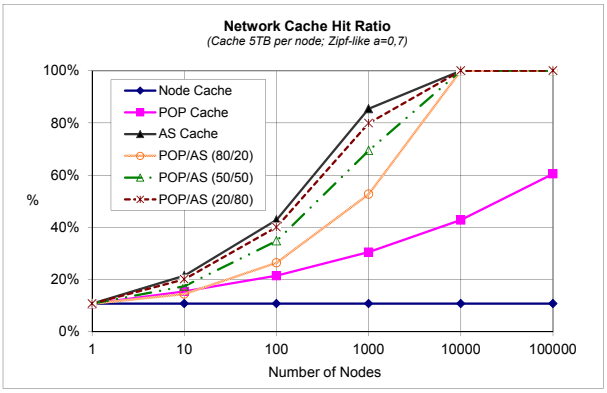


Figure 3.14: Network cache hit ratio with different collaborative caching strategies.

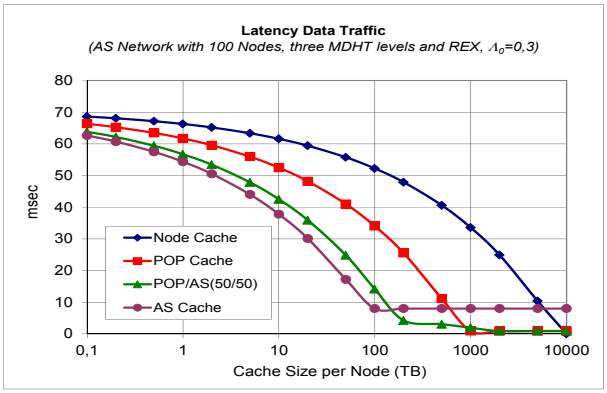


Figure 3.15: Latency of data traffic as a function of the node cache size.

Looking at the latency in Figure 3.15, we have considered a three level AS with 100 access nodes arranged in 10 POPs, an average AS width of 1000 km and an *intrinsic locality factor* $\Lambda = 0.3$. That is, 30% of communication requests originated from network clients are satisfied internally, before adding any cache to the network nodes. Figure 3.15 shows how the latency of data packets changes as we add cache space to the network nodes. The latency decreases tremendously, because the network locality (i.e. ratio of on-net traffic) is increased by the objects stored in the cache system. As an example, let us consider a cache size of 5TB per node, in a network with 100 nodes. If the available storage is used at the AS level, Figure 3.14 gives a hit ratio around 40%. If the network has a 30% of initial intrinsic locality ($\Lambda = 0.3$), the total amount of on-net traffic increases to 60% and the average packet latency is reduced of about 40%.

Summary Placing the cache system at the AS level seems to be the most effective strategy in terms of hit ratio, latency and increased ratio of on-net traffic. For large cache sizes, the latency of packets can be further reduced using multilevel caching at both the POP and AS levels. In that case, no advantage in terms of on-net traffic ratio can be obtained from the multilevel caching.

Instead, there can be an advantage in terms of decreased internal AS backbone traffic, since a higher hit ratio is achieved at the POP level.

3.4 Conclusions

In this section we have addressed the transfer and storage of NDOs. The first section addresses transfer of NDOs, in point to point operation mode and multipoint. The important difference between the approaches is the first describes a transport protocol with a control loop per NetInf hop, whereas the second describes a control loop that can operate over multiple NetInf hops. This chapter has described the implications of the Convergence Layer approach without an end-to-end transport protocol for NetInf transport. Both unicast and multicast NDO transport cases have been described. It was noted that a retransmission mechanism above the convergence layer is needed if reliable end-to-end transport is desired.

The second issue addressed in this chapter was storage and caching. Broadly classified they are performance indicators for policies of cache management. Traditional replacement policies of evicting the least recently or least frequently used items have been studied as well as mixing least recently used with a random replacement policy at different levels in the cache hierarchy. Extensions of cache management policies include NDO migration and replication. In this work we have considered LRU, LFU, RND as well as the new FMD policy for cache management. Another extension is caching in a collaborative environment within the GIN system. An implementation of GIN is in progress with some running functionality.

4 In-Network Services

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 a more efficient usage of the network resources and higher 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 Round Trip Times (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.

In this chapter we introduce our concept of *in-network services*, that can be placed dynamically in the network, and the related system architecture (Section 4.3). One central aspect of service placement is the placement algorithm described in Section 4.4.

4.1 Concept

Active information objects or *in-network services* is our concept of connecting cloud networking with information-centric networking. While information-centric networking has so far focused on retrieving information from the network, active information objects can also be applications or processes, analogous to object oriented programming. Information-centric approaches help to realise this vision by providing functionalities like registration, discovery, and name-based resolution and routing.

In an information-centric network, content must be controlled and managed (see Section 3.3.2). A CDN offers high-performance delivery of content via dedicated servers, thus reducing the origin content server load, reducing latency for end users, and increasing availability of the content. Nowadays, many CDN networks provide a simple content placement strategy, where the content from an origin server is simply replicated amongst replica servers. The regions where content is replicated are specified by the content provider paying for the CDN usage. In addition, caching concepts (see Section 3.3) help to reduce load on servers, reduce access times of clients and potentially increase the QoE for the end user. Whereas traditional caches act independent of each other, collaborative caching strategies improve the usage of storage resources and thus result in a higher hit ratio (Section 3.3.6). Collaborative caching is one possible realization of content placement. In general, content placement tries to "optimize" the replication of content (i.e., the content distribution) by deciding which content will be replicated at which replica servers according to a cost model that includes parameters like storage costs, transport costs, the content popularity, and provider/operator policies.

Service placement is extending the concept of content placement towards services, i.e., the services itself are moved and replicated in the network (that is why we refer to them as *in-network services*). Dynamic content, e.g., personalized HTML and XML pages built on the fly are a first step towards this direction. Service placement is also an important concept in ad hoc networks, where the number and location of service instances in the network should be adapted over time (and a changing network topology) such that the service demand of the network is met at a minimal cost.

The concept of moving services in the network is also called Service Program Mobility (SPM) [33]. It is especially appealing as an extension to information-centric network with in network caching

as one of the key features, as registration, discovery, and name-based resolution and routing of content and service is already provided by the network.

As an example, the future will more and more bring dynamic and highly personalized multimedia content. Caching of this kind of content in the "final version" as delivered to the customer will be of no use since the content will typically be requested only once in the personalized form. However, the piece of software used for personalizing the general multimedia content is typically much smaller in volume than the content. Consequently, the general, not yet personalized, content should be cached close to the user and, in order to effectively use this local content, the service component for making the personalization should also be moved and run on a node close to the user. Thus, the objective is to cache the content before personalization and move the services towards the user such that the personalization takes place on the path between the cached content and the user.

4.2 Definition of Terms

In this section, we refer to the term *service* as a computer-based IT service, which could, e.g., be an end-user service, a web-service, an application-like service for customers, or a value-added service. Our definition of *service* is in line with the context of *System architecture* or *Service Oriented Architecture* (SOA) [34]. In general, a service is thus a software component hosted and executed on one or several *application servers*. It consists of a service-specific logic and may contain state. Stateless services can be services like data aggregation services, whereas stateful services can be used for executing business logics. A service can be accessed by local or remote clients by means of issuing service requests that, in case of remote clients, are transmitted over the network. Usually, several different services are active in the network simultaneously, and each application server may host multiple services.

According to [35] several attributes of services are relevant in the context of *service placement*. First, a service must be node-independent (and not tied to a specific network node) in order to think about service placement. Second, services may by centralized or distributed, i.e., either the service requires for it to run on one single node, or multiple service components may spread out over several nodes. Third, for monolithic services, identical replicas (service instances) of the service may freely be created; whereas composite services require consideration of service-specific interdependencies between the subservices (also referred to as service enablers or components) they are composed of.

Clients or *client nodes* are requesting services. The number of service requests over a period of time is referred to as *service demand*. One operator policy could be to satisfy the service demand of all client nodes. Another operator might apply *admission control* to limit the number of client nodes in order to better serve the admitted clients.

Servers or *application servers* are nodes which are able to run service instances. In order to ease service placement, application servers considered for a certain service should providing the same (or at least a similar) framework for instantiating and executing the service instances, i.e., providing common interfaces and the same operating system. Similarly, *content servers* are nodes where content can be stored and retrieved from.

We refer to the term *service configuration* as the set of nodes hosting instances or subservices of this service. *Service placement* or *service migration* is the (continuous) adaptation of the service configuration to varying service demand (i.e., a changing frequency of service requests in on or more regions of the network) or changing network topology. Similar to that, *content placement* is the adaption of content locations based on the varying content demand. Thereby, content and service instances may be moved, replicated or deleted from/to content and application servers, respectively. A good summary of related work on service placement can also be found in [35].

4.3 Architecture

In this section the basic architecture and functionality of SPM is described. This description is the basis for a prototype that is targeted for by the end of the SAIL project. The SPM prototype can be seen as another application scenario for NetInf since SPM heavily relies on the caching and name resolution functionality of NetInf. Thereby, the SPM controller runs on top of the NetInf architecture and relies on NetInf to obtain locators and content; otherwise, it is independent.

The key components of SPM are an SPM controller and an SPM enabled server. The SPM controller receives and interprets service scripts. A service script corresponds to an active information object. It describes the interaction of multiple basic services that process content and forward it to another service or the requester. A simple example is a service script that consists only of a single watermarking service. The original content, a video, is passed to the watermarking service that processes the video by adding the watermark and forwards it to the requester. The service script specifies the service flow and contains the names of the content and of the services involved. Here we distinguish a service program and a service instance. A service program is a file that can be started in a suitable service environment. This could be, e.g., a .jar file that is started in an OSGi environment. A service instance is then a specific instance of the service running and addressable on a distinct server.

The basic architecture and functionality of SPM is illustrated in Figure 4.1. (1) The requester sends the service script that contains content and service names according to the NetInf naming scheme to the SPM controller. The SPM controller interprets the service script and (2) relies on NetInf for resolving content names to content locators but does not yet fetch the content. This means that an option in the NetInf GET is required for specifying that, in this step, the requester is interested in locators only (and not the content). The SPM controller is aware of both running service instances as well as servers in which service instances can be started. Consequently, the SPM controller contains the intelligence of the SPM concept and determines the optimal location for the service instances, i.e., it runs the service placement algorithm. The candidate locations may either be already running service instances or servers on which the service program is to be started. The optimal selection of the service instance location depends on the content locations such that consequently the selection of the service instance location also implies a selection of content locations from the retrieved set of locators. The optimal selection of the service depends on multiple factors like costs for starting a service, costs for transporting content, but also the effect that network and server parameters like available bandwidth, latency, or load may have on the service quality. The general service placement problem is described in detail in Section 4.4 where also a generic example is included.

Following the placement decision, the SPM controller communicates with the SPM interfaces running on the SPM servers. (3) The SPM controller transmits the service script including concrete content and service instance locators to the SPM interface. If the service instance is not yet running in the local service platform, (4) the SPM interface is responsible for fetching the service program and starting it on the service platform. The service program is obtained relying on NetInf since it is a normal file. Finally, the SPM instance calls the service program with the parameters as specified in the service script. These parameters may include content name plus locator, client address, or addresses of other service instances the local component is connected to. In the example of the watermarking service this means that the watermarking service is called with name+locator of both movie and watermark. (5) The service instance fetches watermark and video using NetInf, adds the watermark to the video and (6) returns the processed content, i.e. the watermarked video, to the client.

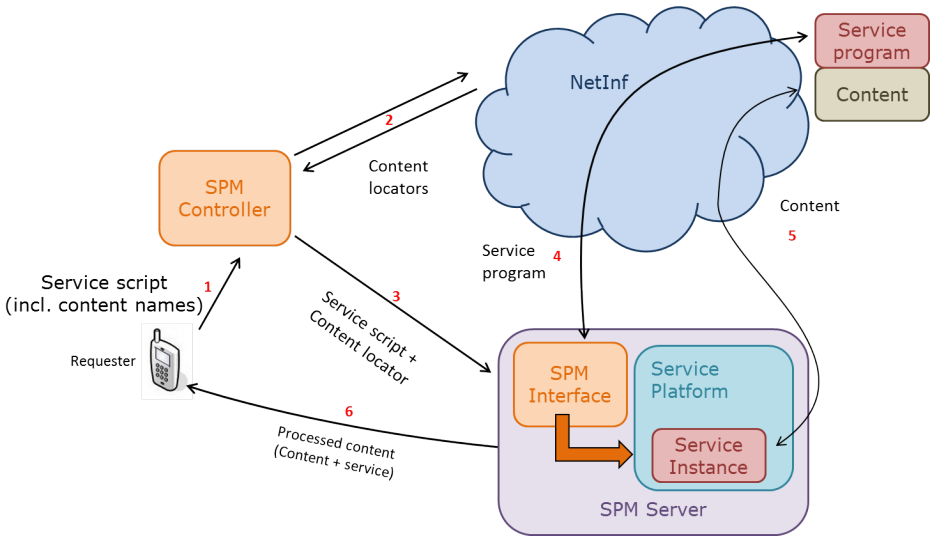


Figure 4.1: Basic SPM Architecture and Functionality.

4.4 Service Placement

The scope of service placement mainly focuses on the following questions [35]:

1. Where in the network are service instances to be placed?
2. How many service instances are required for "optimal" operation?
3. When should the current configuration of a service be adapted?

In contrast to that, the request-routing system is responsible for redirecting user requests to the "optimal" replica, again based on certain metrics and costs. The request-routing is not considered in this section. However, when placing services the system must be aware of the implemented request routing algorithm, as both algorithms will impact each other.

In general, service placement can be designed in three different ways [35]:

a. using a central entity coordinating the whole system (the central entity itself can be a fixed node, or the node taking over the role of the central entity can be dynamically decided),

b. based on a distributed consensus between all or adjacent nodes, or

c. local decisions by each node taking into account locally available information.

In this remainder of this section, the service placement problem is formulated and a solution to it is given at a sample service. First, goals of service placement are defined, followed by a list of different cost aspects. Next, different time scales of service placement decisions are discussed. It is followed by the actual formulation of the service placement problem, starting with assumptions made in our work, the cost function, a description of the considered system, constraints for the optimization, and parameters used. How to solve the service placement optimization is shown at a simple sample system and service.

4.4.1 Goals

In a very general speaking, the goals of service placement are to reduce costs of the provided service, while maintaining or improving the quality of the service. The specific goals to be optimized for a

given service vary based on the service itself as well as policies defined by the service provider and operator.

For example, depending on the service and use case the following variants or alternatives of the service placement exist [35]:

- Reduce bandwidth consumption (network traffic) by placing multiple copies of the service in the network, i.e., trading storage (for instantiating the service on multiple servers) for bandwidth. This scenario makes sense, e.g., for today's mobile core networks, where a significant increase in mobile video traffic is putting a high burden on the deployed network infrastructure.
- For time-critical applications, e.g., deploy as many service instances as possible in order to reduce service access times

Therefore, different weights are assigned to the different cost types listed below. We distinguish between *real* (monetary) costs and *virtual* costs. Real costs are, e.g., storage and roaming costs. Virtual costs comprise amongst others quality metrics, like the response time of a service, or indirect costs, like the available battery lifetime on a mobile device. Virtual costs are used similar to real costs. For example, the response time of a service is proportional to its virtual cost, i.e., the higher the response time, the higher the costs. Another example is to use virtual costs for load balancing the system: less loaded nodes are "cheaper" than highly loaded nodes. Both virtual and real costs are considered similarly in optimizing the service placement.

4.4.2 Costs

Costs are generated in the following actions:

- Service is migrated or replicated to another node
- Service is collecting relevant data from the (multiple) content servers
- Signalling and data exchange
  - between subservices of a composite service
  - between service instances for synchronization, update of state, and configuration
- Processing of the content at all involved nodes
- Answering the service request, i.e., serving the user

The above costs may include the following real (R) and virtual (V) costs:

- Transportation related costs (including migration of service instance)
  - Routing and roaming/transit costs (R)
    - proportional to network distance, e.g., hops, number of traversed AS
    - proportional to amount of transmitted data
  - Temporal distance, e.g., round-trip-time, delay, jitter (V)
  - Available bandwidth / throughput on the traversed paths (including links + nodes) (V)
- Processing and service related costs (including instantiation of service)
  - Required CPU cycles (R)
  - Required storage, including memory and mass storage (e.g. cloud storage) (R)

- Current server utilization, e.g., less loaded nodes are cheaper (V)
- Battery lifetime of mobile nodes (V)
- Available storage capabilities (V)

4.4.3 Timing of Placement Decisions

The migration or replication of a service instance creates costs for a) network traffic for transporting the service instance (i.e. the source code or a binary file and potentially including a huge state), b) the instantiation of the service at the new location, and c) virtual costs describing the delay for the migration. The delay until the new service instance is ready to be used may take seconds to hours, depending on the network connectivity and the size of service plus state. Hence, a small peak in the service demand may not be worth or justify a costly and protracted migration process. This also implies that a service presumably will not be migrated based on a single user request, but, more likely, the service placement will adapt the service configuration following general trends in service demand or a changed network topology. Moreover, this decision may also be content and service specific, i.e., a start-up delay of 60 seconds may still be reasonable when applying a media manipulation service to a 2 hours high quality video file, but is not acceptable for short video clips. These limitations and considerations need to be taken into account when adapting the placement problem to a specific service.

4.4.4 Assumptions

In the focus of this work we assume that:

- Physical and virtual network topologies are more or less stable (in contrast to (Mobile) Ad Hoc Networks).
- Target function is to reduce monetary routing and processing costs.
- *Monolithic services*, i.e., services may not be composed of subservices, and identical service instances are migrated/replicated in the network.
- *Distributed services* without global state, i.e., independent service instances may be created, but no synchronization among them is required.
- *Node-independent services*: The service is not tied to a specific node, but may be installed and executed on any server in the network.
- *Global knowledge*: A central entity (the "SPM server") is globally coordinating and managing the service placement, i.e., executing a centralized algorithm for solving the service placement problem. The SPM server is running on a fixed, infrastructural node and is not assigned dynamically.
- Service discovery is redirecting a client-request to the optimal (here: closest in network topology view) server running an instance of the requested service.
- Content locations are known (or can be retrieved from the ICN) and change on a larger time scale than the processing time of the service placement algorithm. A specific content may be located at several content servers.

4.4.5 Formulation of the Service Placement Problem (including content placement)

Cost function: Based on the different cost types stated above, we define the following cost function for a single service:

$$\begin{aligned} f(i,j) &= \text{storage of data} + \text{storage of components} + \text{transport of data} + \\ &\quad + \text{processing} + \text{migration to new service configuration} \\ &= F^{\mathrm{D}} + F^{\mathrm{C}} + F^{\mathrm{T}} + F^{\mathrm{P}} + F^{\mathrm{M}} \end{aligned} \tag{4.1}$$

We then try to find the placement of data and components that minimizes this cost function:

```
Minimize f(i,j)
```

System description: The following system consisting of nodes, data sources, and service components is defined in order to perform the optimization:

- Nodes N
  - N_1: user devices — high costs for storage and processing
  - N_2: base station
  - N_3: P-GW
  - N_4: dedicated server — low costs for processing
  - N_5: dedicated cache — low costs for storage
  - N_6: external sources — no costs for storage (see CSTR 1)
- Data sources D
  - D_1: output of components
  - D_2: external sources
  - D_3: internal caches/replicas of data
- Service components S
  - S_1: components of the service
  - S_2: application on the client device

Note: we assume that user nodes (i.e. client devices) also are potential data sources (e.g. user generated content with the built-in camera of the device) and can be used to run service components locally.

Constraints: There are a number of constraints (CSTR) for the system configuration that need to be considered in the optimization:

CSTR 1 Certain data sources only store certain data (not more or less), e.g. external data sources (N_6) only provide certain data

- Data provided by a source must be on the source node (but may additionally be replicated to other nodes)
- External sources cannot store additional data (besides the content they provide)

CSTR 2 Storage and processing capabilities of each node are limited (different limits per type of node may be applied).

CSTR 3 Data sources of type "D_1" must be on the corresponding component (but may additionally be replicated to other nodes).

CSTR 4 Each data item must be available at least at one node in the network.

Parameters: For each service to be evaluated we need to define the following parameters prior to running the optimization. Thereby, i, j are instances of components and data sources, respectively. After defining the parameters, an example is given using a simple sample service. Costs are given in monetary units (MU), size of data is described in data units (DU), processing is abstracted in processing units (PU).

$S = [s_{i,j}]$	Description of the service, i.e. describing I data flows between J components and the order of the component calls	$I \times J$
$L = [l_{x,y}]$	link costs (between nodes x and y) of the underlying network with a total of N nodes	$N \times N$
w^{o}	Size of the components (in DU)	$1 \times I$
w^{s}	Size of data (in DU)	$1 \times J$
w^{t}	Costs for storing 1 DU at each node (in MU)	$1 \times N$
w^{p}	Amount of processing done at each component (in PU)	$1 \times I$
w^{i}	Costs for instantiation of the components (in PU)	$1 \times I$
w^{d}	Costs for deinstallation the components (in PU)	$1 \times I$
w^{q}	Costs for processing 1 PU at each node (in MU)	$1 \times N$

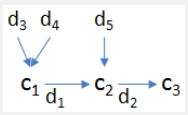
For solving the optimization we define the following additional parameters:

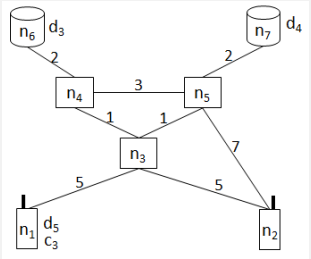
C	Candidate configuration of components on nodes	$I \times N$
D	Candidate availability of data on nodes	$J \times N$
E	Current configuration of components on nodes	$I \times N$
F	Current availability of data on nodes	$J \times N$

Example: The example shows a sample system with $N = 7$ nodes (2nd figure) and a service consisting of $I = 3$ components using $J = 5$ data sources (1st figure). The network consists of 2 databases (n_6, n_7), 2 mobile devices (n_1, n_2), and 3 core nodes (n_3 to n_5). The first component c_1 is using input d_3 and d_4 and its output is d_1. Component c_2 is retrieving d_1 and processes it with d_5 (user generated content from n_1). Component c_3, the player on the mobile device n_1, is finally receiving and displaying the output from component c_2.

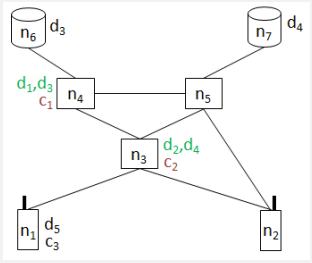
$$S = \begin{array}{c} \begin{matrix} d_1 & d_2 & \dots & d_5 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} c_1 \\ c_2 \\ c_3 \end{matrix}$$

$$L = \begin{bmatrix} 0 & 9 & 5 & 6 & 6 & 8 & 8 \\ 9 & 0 & 5 & 6 & 7 & 8 & 9 \\ 5 & 5 & 0 & 1 & 1 & 3 & 3 \\ 6 & 6 & 1 & 0 & 2 & 2 & 4 \\ 6 & 6 & 1 & 2 & 0 & 4 & 2 \\ 8 & 8 & 3 & 2 & 4 & 0 & 6 \\ 8 & 9 & 3 & 4 & 2 & 6 & 0 \end{bmatrix}$$





Current service configuration:



$$E = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$F = \begin{bmatrix} d_1 \\ d_2 \\ \dots \\ \dots \\ d_5 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

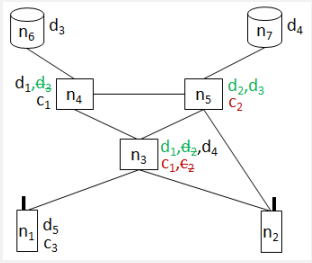
Storing related:

$$w^{\mathrm{o}} = \begin{bmatrix} 0.1 & 0.1 & 0.2 \end{bmatrix}$$

$$w^{\mathrm{s}} = \begin{bmatrix} 1.0 & 1.0 & 2.0 & 1.0 & 1.0 \end{bmatrix}$$

$$w^{\mathrm{t}} = \begin{bmatrix} 5.0 & 5.0 & 3.0 & 1.01.0 & 0.0 & 0.0 \end{bmatrix}$$

Candidate service configuration:



$$C = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} d_1 \\ d_2 \\ \dots \\ \dots \\ d_5 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Processing related:

$$w^{\mathrm{p}} = \begin{bmatrix} 1.0 & 3.0 & 2.0 \end{bmatrix}$$

$$w^{\mathrm{i}} = \begin{bmatrix} 0.2 & 0.3 & 0.5 \end{bmatrix}$$

$$w^{\mathrm{d}} = \begin{bmatrix} 0.1 & 0.1 & 0.1 \end{bmatrix}$$

$$w^{\mathrm{q}} = \begin{bmatrix} 5.0 & 5.0 & 1.0 & 2.0 & 2.0 & \mathrm{Max} & \mathrm{Max} \end{bmatrix}$$

The costs for each candidate solution (i, j) are calculated as follows:

Storage of data F^{D} Storage of data j on node x: ${f^{\mathrm{D}}}_{j,x} = d_{j,x} \cdot {w^{\mathrm{s}}}_j \cdot {w^{\mathrm{t}}}_x$
In our example this means:

$$u^{\mathrm{d}}{}_1 = \begin{bmatrix} 0 & 0 & 6 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$u^{\mathrm{d}}{}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The total storage costs in the candidate service configuration then sum up to:

$$F^{\mathrm{D}} = \sum_{j=1}^{J} \sum_{x=1}^{N} {f^{\mathrm{D}}}_{j,x} \quad , \text{with } {f^{\mathrm{D}}}_{j,x} = d_{j,x} \cdot {w^{\mathrm{s}}}_j \cdot {w^{\mathrm{t}}}_x \tag{4.2}$$

Storage of components F^{C} Similar to the cost for storage of data, we can derive the total storage costs for the considered candidate service configuration:

$$F^{\mathrm{C}} = \sum_{i=1}^{I} \sum_{x=1}^{N} {f^{\mathrm{C}}}_{i,x} \quad , \text{with } {f^{\mathrm{C}}}_{i,x} = c_{i,x} \cdot {w^{\mathrm{o}}}_x \cdot {w^{\mathrm{t}}}_x \tag{4.3}$$

Processing F^{P} Processing costs for the considered candidate service configuration are the following:

Processing of component i on node x: ${f^{\mathrm{P}}}_{i,x} = c_{i,x} \cdot {w^{\mathrm{P}}}_i \cdot {w^{\mathrm{q}}}_x$

The total storage costs in the candidate service configuration then sum up to:

$$F^{\mathrm{P}} = \sum_{i=1}^{I} \sum_{x=1}^{N} {f^{\mathrm{P}}}_{i,x} \qquad \text{, with } {f^{\mathrm{P}}}_{i,x} = c_{i,x} \cdot {w^{\mathrm{P}}}_i \cdot {w^{\mathrm{q}}}_x \tag{4.4}$$

Transport of data This is the most complex computation in this optimization. As there may be multiple nodes providing/caching the same data, the following steps are necessary.

For each d_j replace all zeros with MAX_INT:

$$d'_j \quad = \quad ((1 - d_j) \cdot (\text{MAX\_INT} - 1)) + 1 \tag{4.5}$$

For example, this would result in:

$$d'_1 = \begin{bmatrix} \text{Max} & \text{Max} & 1 & \text{Max} & \text{Max} & \text{Max} & \text{Max} \end{bmatrix}$$
$$d'_3 = \begin{bmatrix} \text{Max} & \text{Max} & \text{Max} & \text{Max} & 1 & 1 & \text{Max} \end{bmatrix}$$

Next, we need to "expand" each vector d_j to a matrix, such that there is only at least one single entry 1 or Max in each line. We do this, by multiplying the vector with the identity matrix $\mathbf{E}$:

$$\mathbf{D}_j = d_j \cdot \mathbf{E} \tag{4.6}$$

The result is a NxN matrix with 1 or Max values in its diagonal and 0 elsewhere. For example:

$$\mathbf{D}_1 = \begin{bmatrix} \text{Max} & & & \cdots & & & \\ & \text{Max} & & & 0 & & \\ & & 1 & & & \cdots & \\ & & & \text{Max} & & & \\ \cdots & & & & \text{Max} & & \\ & 0 & & & & \text{Max} & \\ & & \cdots & & & & \text{Max} \end{bmatrix}$$

$$\mathbf{D}_3 = \begin{bmatrix} \text{Max} & & & \cdots & & & \\ & \text{Max} & & & 0 & & \\ & & \text{Max} & & & \cdots & \\ & & & \text{Max} & & & \\ \cdots & & & & 1 & & \\ & 0 & & & & 1 & \\ & & \cdots & & & & \text{Max} \end{bmatrix}$$

The generic costs for transporting data from node x to node y are given by matrix $\mathbf{L}$, whereas D_j states whether content is available at a certain node (value "1") or not (value "Max", as content cannot directly be retrieved from that node). Multiplying these matrixes gives the costs for transporting data item j from node x to node y.

$${\mathbf{K}^{\mathrm{d}}}_j = \mathbf{L} \cdot \mathbf{D}_j \tag{4.7}$$

As content may be available from multiple nodes, we assume we can get the data from the optimal node, i.e. the one with lowest transport costs. This can be expressed as:

$$k^{\mathrm{d}}{}_j = \min(\mathbf{K}^{\mathrm{d}}{}_j) \quad \text{, vector of size } 1 \times N \tag{4.8}$$

$$\mathbf{K}^{\mathrm{d}} = [k^{\mathrm{d}}{}_1; k^{\mathrm{d}}{}_2; \ldots; k^{\mathrm{d}}{}_J]^{\mathrm{T}} \tag{4.9}$$

Looking at the example we get:

$$\mathbf{K}^{\mathrm{d}}{}_1 = \begin{bmatrix} \cdots & & \mathrm{Max} & & \cdots \\ \cdots & & \mathrm{Max} & & \cdots \\ 5 \;\; 5 & 0 & 1 & 1 & 3 \;\; 3 \\ \cdots & & \mathrm{Max} & & \cdots \\ & & \cdots & & \\ & & \cdots & & \\ \cdots & & \mathrm{Max} & & \cdots \end{bmatrix}$$

$$\mathbf{K}^{\mathrm{d}}{}_3 = \begin{bmatrix} \cdots & & \mathrm{Max} & & \cdots \\ & & \cdots & & \\ & & \cdots & & \\ \cdots & & \mathrm{Max} & & \cdots \\ 6 \;\; 6 & 1 & 2 & 0 & 4 \;\; 2 \\ 8 \;\; 8 & 3 & 2 & 4 & 0 \;\; 6 \\ \cdots & & \mathrm{Max} & & \cdots \end{bmatrix}$$

$$\mathbf{K}^{\mathrm{d}} = \begin{bmatrix} \min(\mathbf{K}^{\mathrm{d}}{}_1) \\ \min(\mathbf{K}^{\mathrm{d}}{}_2) \\ \ldots \\ \ldots \\ \min(\mathbf{K}^{\mathrm{d}}{}_5) \end{bmatrix} = \begin{bmatrix} 5 & 5 & 0 & 1 & 1 & 3 & 3 \\ 6 & 6 & 1 & 2 & 0 & 4 & 2 \\ 6 & 6 & 1 & 2 & 0 & 0 & 2 \\ 5 & 5 & 0 & 1 & 1 & 3 & 0 \\ 0 & 9 & 5 & 6 & 6 & 8 & 8 \end{bmatrix}$$

Finally, looking at matrix $\mathbf{S}$ defining the service composition and the candidate service configuration $\mathbf{C}$ we can calculate which data item j will be transported to which node x:

$$\mathbf{T} = \mathbf{C} \cdot \mathbf{S}^{\mathrm{T}} \quad \text{, matrix of size } J \times N \tag{4.10}$$

$$\mathbf{T}' = \begin{array}{c} \begin{matrix} n_1 & n_2 & n_3 & \ldots & n_7 \end{matrix} \\ \begin{bmatrix} \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \end{bmatrix} \end{array} \begin{matrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{matrix}$$

Then, the total transportation costs in the candidate service configuration sum up to:

$$F^{\mathrm{T}} = \sum_{j=1}^{J} \sum_{x=1}^{N} f^{\mathrm{T}}{}_{j,x} \quad \text{, with } f^{\mathrm{T}}{}_{j,x} = t_{j,x} \cdot w^{\mathrm{s}}{}_j \cdot K^{\mathrm{d}}{}_{j,x} \tag{4.11}$$

Migration to new service configuration M In addition to the costs generated at the candidate service configuration, the costs for reaching a new service configuration must be considered in the optimization. These costs can be split into the following parts:

$$\begin{aligned} F^{\mathrm{M}} &= \text{transporting data to new locations} + \\ &\quad \text{transporting components to new locations} + \\ &\quad \text{instantiiation of components at new locations} + \\ &\quad \text{deinstallation of components at old locations} \\ &= M^1 + M^2 + M^3 + M^4 \end{aligned} \tag{4.12}$$

with

$$M^1 = \sum_{j=1}^{J} \sum_{x}^{N} {m^{\mathrm{s}}}_{j,x} \cdot {w^{\mathrm{s}}}_{j} \cdot {k^{\mathrm{d}}}_{j,x} \tag{4.13}$$

$$M^2 = \sum_{i=1}^{I} \sum_{x}^{N} {m^{\mathrm{i}}}_{i,x} \cdot {w^{\mathrm{o}}}_{i} \cdot {k^{\mathrm{c}}}_{i,x} \tag{4.14}$$

$$M^3 = \sum_{i=1}^{I} \sum_{x}^{N} {m^{\mathrm{i}}}_{i,x} \cdot {w^{\mathrm{i}}}_{i} \cdot {w^{\mathrm{q}}}_{x} \tag{4.15}$$

$$M^4 = \sum_{i=1}^{I} \sum_{x}^{N} {m^{\mathrm{d}}}_{i,x} \cdot {w^{\mathrm{d}}}_{i} \cdot {w^{\mathrm{q}}}_{x} \tag{4.16}$$

whereby M_{s}, M_{i} and M_{d} indicate which data or components must be moved, instantiated or deinstalled, respectively.

$$M^{\mathrm{s}} = (D \oplus F) \cdot D \tag{4.17}$$

$$M^{\mathrm{i}} = (C \oplus E) \cdot E \tag{4.18}$$

$$M^{\mathrm{d}} = (C \oplus E) \cdot C \tag{4.19}$$

Note: We assume there are no costs for deleting data.

In our sample service configuration components 1 and 3 will be transported and instantiated at a new location:

$$M^{\mathrm{i}} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The variable ${k^{\mathrm{c}}}_{i,x}$ (matrix K^{c}) in equation 4.14 is derived similar to matrix K^{c} described in the previous paragraph (equations 4.5 to 4.9):

$$c'_i = ((1 - c_i) \cdot (\mathrm{MAX\_INT} - 1)) + 1 \tag{4.20}$$

$$\mathbf{C}_i = c_i \cdot \mathbf{E} \tag{4.21}$$

$${\mathbf{K}^{\mathrm{c}}}_{i} = \mathbf{L} \cdot \mathbf{C}_i \tag{4.22}$$

$${k^{\mathrm{c}}}_{i} = \min({\mathbf{K}^{\mathrm{c}}}_{i}) \tag{4.23}$$

$$\mathbf{K}^{\mathrm{c}} = [{k^{\mathrm{c}}}_{1}; {k^{\mathrm{c}}}_{2}; \ldots; {k^{\mathrm{c}}}_{J}]^{\mathrm{T}} \tag{4.24}$$

Putting everything together Coming back to the service placement function in equation (4.1):

$$f(i,j) = F^{\mathrm{D}} + F^{\mathrm{C}} + F^{\mathrm{T}} + F^{\mathrm{P}} + F^{\mathrm{M}} + M^1 + M^2 + M^3 + M^4$$

with

$$F^{\mathrm{D}} = \sum_{j=1}^{J}\sum_{x=1}^{N} d_{j,x} \cdot {w^{\mathrm{s}}}_j \cdot {w^{\mathrm{t}}}_x \tag{4.2}$$

$$F^{\mathrm{C}} = \sum_{i=1}^{I}\sum_{x=1}^{N} c_{i,x} \cdot {w^{\mathrm{o}}}_x \cdot {w^{\mathrm{t}}}_x \tag{4.3}$$

$$F^{\mathrm{P}} = \sum_{i=1}^{I}\sum_{x=1}^{N} c_{i,x} \cdot {w^{\mathrm{p}}}_i \cdot {w^{\mathrm{q}}}_x \tag{4.4}$$

$$F^{\mathrm{T}} = \sum_{j=1}^{J}\sum_{x=1}^{N} t_{j,x} \cdot {w^{\mathrm{s}}}_j \cdot {k^{\mathrm{d}}}_{j,x} \tag{4.11}$$

$$M^1 = \sum_{j=1}^{J}\sum_{x=1}^{N} {m^{\mathrm{s}}}_{j,x} \cdot {w^{\mathrm{s}}}_j \cdot {k^{\mathrm{d}}}_{j,x} \tag{4.13}$$

$$M^2 = \sum_{i=1}^{I}\sum_{x=1}^{N} {m^{\mathrm{i}}}_{i,x} \cdot {w^{\mathrm{o}}}_i \cdot {k^{\mathrm{c}}}_{i,x} \tag{4.14}$$

$$M^3 = \sum_{i=1}^{I}\sum_{x=1}^{N} {m^{\mathrm{i}}}_{i,x} \cdot {w^{\mathrm{i}}}_i \cdot {w^{\mathrm{q}}}_x \tag{4.15}$$

$$M^4 = \sum_{i=1}^{I}\sum_{x=1}^{N} {m^{\mathrm{d}}}_{i,x} \cdot {w^{\mathrm{d}}}_i \cdot {w^{\mathrm{q}}}_x \tag{4.16}$$

This results in:

$$\begin{aligned} f(i,j) = \sum_{x=1}^{N} \Bigg(& \sum_{j=1}^{J} \left({w^{\mathrm{s}}}_j (d_{j,x} \cdot {w^{\mathrm{t}}}_x + t_{j,x} \cdot {k^{\mathrm{d}}}_{j,x} + {m^{\mathrm{s}}}_{j,x} \cdot {k^{\mathrm{d}}}_{j,x}) \right) + \\ & + \sum_{i=1}^{I} \left(c_{i,x} ({w^{\mathrm{o}}}_x \cdot {w^{\mathrm{t}}}_x + {w^{\mathrm{p}}}_i \cdot {w^{\mathrm{q}}}_x) + {m^{\mathrm{i}}}_{i,x} ({w^{\mathrm{o}}}_i \cdot {k^{\mathrm{c}}}_{i,x} + {w^{\mathrm{i}}}_i \cdot {w^{\mathrm{q}}}_x) + {m^{\mathrm{d}}}_{i,x} \cdot {w^{\mathrm{d}}}_i \cdot {w^{\mathrm{q}}}_x \right) \Bigg) \end{aligned} \tag{4.25}$$

with

$$\begin{aligned} \mathbf{K}^{\mathrm{d}} &= [{k^{\mathrm{d}}}_{j,x}] && : \text{Minimum link costs for transporting data in candidate placement} \\ \mathbf{K}^{\mathrm{c}} &= [{k^{\mathrm{c}}}_{j,x}] && : \text{Minimum link costs for transporting components in cand. placement} \\ \mathbf{T} &= [t_{j,x}] && : \text{Required transport of data in candidate placement} \\ \mathbf{M}^{\mathrm{s/i/d}} &= [{m^{\mathrm{s/i/d}}}_{i,x}] && : \text{Data or components to be moved, instantiated or deinstalled} \end{aligned} \tag{4.26}$$

Using the above equation it is possible to calculate the optimal placement decision, i.e., which content and services should be moved to, replicated to or removed from which nodes given the current service configuration and service access pattern. We provide a very general definition

the problem space, which then has to be adapted to the specific service, system parameters, and operator policies.

This optimization can be used to calculate the optimal service placement decision and to evaluate the influence of certain service/system parameters on the behaviour of the service placement. The next step is to provide a practical realization of this optimization that can be executed in the decision taking entity under real-time conditions. This is the basis for performance evaluations of the optimization for specific services with realistic user behaviour and system/service parameters.

4.5 Summary

In this chapter, the concept of in-network services was introduced. The goal of in-network services is to keep not only content close to the user as in in-network caching, but to also move the processing close to the user. The benefit of this concept is to use localized cached content for personalized composite services that are otherwise provided by centralized servers and content stored in CDNs. The proposed SPM architecture shows how in-network services may be implemented relying on NetInf as the means to obtain multiple locators for requested content and to retrieve content and service programs. The architecture described in Section 4.3 presents the most light-weight solution including NetInf only as a means to retrieve content. The decision for this first architecture was taken in order to develop a SPM prototype on top of the current NetInf prototype. A future extension of the architecture will also see NetInf as a means to locate running service instances.

The key problem in placing services in the network is the economic decision whether to install service programs in the network and where. The content and service placement problem is formulated in a general way that captures different aspects of content placement (pre-filling and locating caches) and service placement (installing service programs or components). Service placement can be done in a proactive or reactive way. Proactive service placement is based on the statistical usage of a service and a service component is placed in the network before the service is actually executed. The proactive service placement is optimized together with cache and content placement. Proactive service placement takes place on larger time scales and falls into the domain of network planning and management. Reactive service placement is done on a much shorter time scale and the service is installed in the network as a reaction of the request of a single customer. The described SPM architecture focuses on reactive service placement and the optimization takes place in the SPM controller after it retrieves the locators from NetInf. This means that there is no possibility to place content, such that the service and content placement problem degrades to a pure service placement problem with reduced degrees of freedom for the optimization.

In general, the formulation of the content and service placement problem as a linear program (as described in Section 4.4) achieves two objectives. First, it allows to evaluate the economic benefits of the in-network service placement concept depending on the composite service, the underlying network properties, and the costs for transport, storage, and processing resources. The complexity of the problem and probably also the degree of detail as specified in the program cannot be fulfilled in a real network implementation. As a consequence, the prototype will be built using much simpler and less complex heuristics for solving a reduced optimization problem. However, the generally formulated optimization problem provides an idealistic benchmark solution for evaluating these heuristic-based solutions.

The future work on in-network services will be a prototype implementation of the described SPM architecture. The content and service placement problem will be used for evaluating the economic aspects of in-network services and to identify a promising example service and network scenario as well as a good heuristic to be used for implementing the prototype and demonstrating the benefits of the in-network services concept.

5 Demonstration Scenario: Event with Large Crowd

To highlight the operation of NetInf networks and how different parts of the architecture work together, a specific scenario has been selected to demonstrate this – the Event with Large Crowd (sometimes referred to as Event with Large Crowds (EwLC) in the rest of this chapter). The scenario details the technical use of NetInf functionalities and shows how the specific scenario can be deployed. The Event with Large Crowd scenario is also an overall scenario for the whole SAIL project and some aspects of the scenario are described in deliverable D.A.9 [36]. There is a clear objective to align the prototyping activities with the scenario work.

In this chapter, we will give an overview of the scenario and explain some of the problems that exist with the operation of current Internet architectures in that scenario. We provide a conceptual outline of the different components that are required to realize this scenario and that NetInf can provide. We then explain how the prototyping activities of different project partners fit into this scenario by implementing the required functionality and integrating with other prototypes. Finally, a plan on how to evaluate the NetInf architecture and protocols under the given scenario to show the benefits of NetInf is outlined.

5.1 Scenario Overview

When operators dimension deployments of cellular networks, they base the design on regular demands and load on the network during peak hours. There is however a limit to how much capacity can be allocated to a single location (in particular for radio communication where the available frequency spectrum is a limiting factor), and operators do not want to spend more money on deployments than is typically required. There are however situations where large crowds gather in a relatively small area during a relatively short period of time (on the order of tens of minutes to hours), creating a very high load on the cellular network. This is typically the case at large sports arenas during matches and other sports events, or during planned or impromptu large gathering of people in urban areas (such as during a marathon, the Notting Hill carnival in London, or the recent royal weddings in Sweden and the UK).

Common for all these scenarios is that they occur during some event that gathers a large crowd who are interested in accessing data from the network. This large gathering of people creates a demand on the network that is higher than what the network infrastructure is dimensioned for, causing the user experience to deteriorate. As the people in the crowd are in the same area for the same event, it is also reasonable to expect that they will have similar interests that drive their data access patterns (e.g., at a football match, it is likely that a large fraction of the crowd want to view a replay of a goal). Thus, we believe that there are great potentials for using NetInf in this type of scenario as NDOs can be cached close to users like in local access points, but also in the mobile nodes themselves to serve other nearby mobile nodes, reducing the load of the network. Additionally, user generated NDOs can be distributed either via infrastructure caches or via local peer-to-peer communication techniques to minimize a mobile node's outbound bandwidth consumption.

5.2 Conceptual Description of Required Components

To realize the scenario described in the previous section, there are a number of different technical components and functions that need to be present in the system. This section provides an overview on a conceptual level of the different components that are required to make the scenario possible. More details on how these will be implemented within the boundaries of the SAIL project will be described in the next section.

5.2.1 Content Generators

We envision having two types of content available in this scenario. One is the centrally produced, "global" type of content. This could be content that arrives from the global Internet or content that is centrally produced by the event organizer (such as the official replay clips of the latest goal). In addition, there should also be user generated content in the system. This could for example be content such as video clips of interesting events taken from the smart phones of users. This content should also contain location based meta-data so that it is possible to determine where it originates from and make a selection of which content to view based on location.

5.2.1.1 Content Directory

In order to make both "professional" and user content usefully accessible it needs to be keyed into a directory, and this needs to be able to be retrieved and displayed by users.

As an example let's consider a football match. Let us further suppose that this is being covered by a professional broadcaster fielding multiple cameras, sophisticated video recording and control equipment plus a team of commentators, match statisticians, web bloggers and technical assistants.

The interesting time span of the event will generally be between 120 minutes and 200 minutes, to cover some build up, the main body of the match, half-time, post-match celebrations and, for some events, extra time and/or penalty shoot-out.

Now if a viewer wants to examine some aspect of the match (let's call it an *incident*), the primary consideration will be "when did it happen?" Thus the key dimension in the directory is time. Events recorded in the directory need to be keyed for time and duration. To help make it easy to find what are the relevant times (start and finish) for an incident, it would be really convenient if there was some text that could be searched to help locate the incident. In practice, many professional broadcasters already have an online presence and deliver (in greater or lesser detail) a blog for "significant events" (c.f., Formula 1, UK Premier League matches) together with voice commentaries. The directory application could leverage this work and add a little to it. It is possible that some technical application support could help of course.

To make this more valuable and enable access to interesting views, some extra spatial information needs to be added to the time dimension already present:

- For any given incident there will generally be a (single) point on/near the pitch that is the focus of attention: the location of a goal scorer, an off-the ball challenge, or even some action in the crowd.
- As a subsidiary factor, there will usually be a key direction relative to that key point, such as the direction of a shot on goal, a pass or another player challenging (whether legally or otherwise).
- There might also be a 'range' where things around the significant point out to this range are also interesting.

The things that viewers may wish to recall are segments of the recorded video etc. regarding some incident. They will also often want to see views from a particular angle to the incident. To make this possible without reviewing all the possibilities, sources need to record

- location of recording device
- direction of view
- location of the recorded event (this can to some extent be derived from the two above items)
- preferably, size of viewing area at the incident location (i.e. wide angle, a few metres, etc.)

Combining this information with the incident specification, it would be possible to request, for example, front on, reverse angle or side views of any incident and possibly request close-up, mid-range or wide-angle for some seconds on either side of the incident.

5.2.2 Network Components

There is a need for both infrastructure and mobile nodes to realize this application scenario. In the infrastructure of the event, we see that there should be NetInf enabled routers that have the ability to cache NDOs along their response paths. These same nodes could potentially also handle the NRS of the system, but that could also be done by separate nodes. The NRS should contain information about all the globally generated content, and also allow users to register the user generated content that they want to make available to all other users there (note: users might choose to not register all their content in the "global" NRS, but only make some content available through the local broadcast based NRS to their mobile neighbours). Additionally, a user might prefer to upload the registered content to an infrastructure node such as NRS, which would then take care of serving other users interested in this content.

The mobile nodes should also be able to cache NDOs in order to serve it to their neighbours using local wireless communication technologies. The nodes should be able to resolve information object names both locally using a scoped multicast/broadcast resolution, and also through the global NRS. The user devices will obviously also need to run some suitable applications, which will be outlined in more detail in the next section. There is currently a couple of different efforts on-going to implement a user device.

In order to enable user generated content to be tagged with location based meta-data, there must be a location service in the system. Depending on the exact characteristics, different types of location services can be used such as GPS for outdoors event, seat number based systems at event on arenas where users are expected to stay in their seats for most of the event, and so on.

5.2.3 User Application and Services

The NetInf functionality in the mobile nodes should be implemented independently of the user applications and accessed by those through a well-defined API that allows new applications to be independently developed. For the demonstration, the nodes should have a main application that is used to access and publish information in the NetInf system. This application should have a GUI that visualizes the content directory and provides a mapping between the directory and locations of produced content in the arena. The user should be able to easily choose to view content (most likely video) from either the centrally generated source or from other users in the same manner.

Further, the user should also have the ability to record videos and other types of content and publish it into the NetInf system. The user should be able to decide if the content should be registered in the NRS to be available to everybody in the arena, or if it should only be available locally to the nodes in the local wireless neighbourhood (using the broadcast/multicast name resolution).

5.3 Scenario Implementation Plans

In this section we describe the prototyping activities taking place at different partners to support the EwLC scenario and explain how these components fit into the scenario and which functionality they implement. The prototypes communicate using the NetInf protocol defined in Chapter 2 and the ni: naming scheme specified in [37] for naming NDOs.

5.3.1 NEC NetInf Router Platform (NNRP)

NNRP is NEC's NetInf prototype that implements, as modules, all the components needed for the implementation of the scenario:

- Full support for the ni: naming scheme, including verification of the hash.
- HTTP Convergence Layer, to exchange objects with other nodes.
- HTTP client interface, to allow legacy clients to interface with NetInf.
- Object cache, to cache objects on the path.
- Support for PUBLISH operations.
- Support for name resolution.

The use of any desired naming resolution and publication strategy can be configured through a configuration file.

NNRP can be used in different contexts: it can be used for example on Access Points and base stations, as local node for accessing the NetInf network, or as NRS frontend. Its implementation allows many nodes to run on the same host, e.g. for testing purposes.

Since it supports the HTTP Convergence Layer, NNRP can seamlessly interoperate with other NetInf implementations which also use it. Closer interoperation with other components is possible in the form of modules: an API allows third parties to write and integrate additional modules in NNRP. Modules can either be either written in C, as shared libraries, or as independent processes which communicate with the core over a TCP or Unix socket, allowing modules to be written in different languages and/or to run on different hosts.

Modules are arranged in chains that determine the actions that are applied to a passing message; chains are described in the per-node configuration file; a command-line interface allows for full control of the node at run-time.

5.3.2 RVS

RendezVous Server is NSN's NetInf prototype, which provides NRS and rendezvous functionality for ni: Uniform Resource Locators (URLs) and Well-Known URLs (WKUs) and implements the HTTP CL. The functionality implements an infrastructure support for the EwLC scenario as shown in Figure 5.1 and Figure 5.6.

Namely, a ni: client can publish an NDO locator(s) or an NDO itself to the RendezVous Server (RVS) using the PUBLISH method. NDO locators are stored in RVS information structure and copies of NDOs are stored in local storage (off-path cache), implemented as part of the RVS. Clients can query for an NDOs from the RVS using the GET method. If an NDO or its locator information exists in the RVS, it will be returned in a GET-RESP message.

RVS has been written in Java, tested in the OS X and Windows 7 operating systems and integration tested with existing Java, Python and Ruby tools provided as open source software available

from the NetInf repository on Sourceforge[1]. As a side note, RVS also adopts a role of normal HTTP proxy for regular HTTP traffic.

5.3.3 EAB Mobile Node

The user device developed by EAB for the EwLC scenario implements the applications and services that the end user will be using to publish and access content in the NetInf network. The device is implemented using Android mobile nodes. These devices will use the NetInf protocol to communicate with the infrastructure or directly to each other. The communication with the infrastructure uses 3G as underlying radio technology, the direct communication uses Bluetooth or WiFi.

The Android client currently has the following functionality:

- Assign a ni: name to an object, store this object in the local file system and register the ni: name in the local name resolution database.
- Establish a Bluetooth connection to a nearby Android NetInf client and send a GET OBJECT request.
- Receive GET OBJECT requests from nearby Android NetInf clients, check in the local name resolution database and reply found or not found.
- Retrieve an object from a nearby Android NetInf client after a positive GET reply.

In addition to the functionality that is already present in the current implementation, there are plans for additional features in the prototype. The WiFi Direct standard will be leveraged as a second transport technology besides Bluetooth for exchanging content between mobile nodes. To fully take advantage of the NetInf benefits, full support for caching of NDOs in the mobile nodes will be implemented. There are also plans to integrate the NRS that are part of the NEC NNRP and NSN RVS systems as a key component for multi-peer communication.

The work done so far has been focused on porting the OpenNetInf code to the Android platform. Some parts of the OpenNetInf code need modification and new code has to be written. The reason for this is that some parts of the OpenNetInf code is not compatible with Android Java and also that OpenNetInf does not use the same naming schema and communication protocols as defined in the current SAIL work.

5.3.4 TCD Mobile Node

The aim of TCD is to develop a mobile tablet device that uses DTN (via the Bundle Protocol - [12]) and NetInf protocols as its primary means of communication with other nodes and can be used in the EwLC scenario. A tablet was chosen as the physical device as its interface limits the configuration options available to the end user.

Given the time and resources available the scope of the prototype has been limited to the following functional areas: search in and access NDOs via a filesystem, discovery and access to various system aspects of the tablet via the /proc and /dev filing systems.

The tablet hardware offers WiFi (802.11b/d/g) connectivity as is conventional on tablets.

The functionality currently being integrated includes:

- DTN2 Reference Implementation Bundle Protocol Agent providing an NDO cache via the bundle store provided by DTN2,
- ni: naming of the bundles using the BPQ extension [16] to the Bundle Protocol,

[1] http://sourceforge.net/projects/netinf/

- publication and retrieval of NDOs over a NetInf DTN Bundle Protocol Convergence Layer
- access to the NDO cache as a filing system via the ni: names,
- textual search in the cached NDOs using the Apache Lucene[2] search engine,
- local and remote searching via a Firefox web browser plug-in, and
- access to operating system information using ni: names to refer to items in /proc and /dev on the tablet.

TCD are also intending to provide the ability to act as a gateway between the HTTP- and DTN-based convergence layers using the NetInf nilib code that has been developed as part of the SAIL project. TCD will endeavour to have the EwLC directory application running on this device in due course.

5.3.5 Cross-WP: OCons DTN Routing for Events with Large Crowds

Within work package C of the SAIL project, work on DTN routing in disconnected and infrastructure-less settings has been carried out. As part of the cross-work package work in the project, Tecnalia has considered how this work could be used within the auspices of the EwLC scenario. This is described in more detail in D.C.3 [38]. This work will be considered for the discussions around the scenario, but as the DTN routing prototype does not fully support the NetInf protocol and architecture as described in this document, it will not be used for the technical evaluations.

5.3.6 Integration Examples

In this section we provide a couple of examples of how the different components developed by the project partners can be integrated to address the situations that arise in the scenario. In the examples, the term MN refers to Mobile Node and can be the EAB mobile node, the TCD mobile node, or some other mobile client.

5.3.6.1 Publish NDO to Infrastructure Service

In this section, we illustrate examples of how to use the NetInf protocol and ni: naming scheme in the EwLC scenario to publish NDO to RVS and to query and fetch published content from NNRP and RVS.

Figure 5.1 shows how a Mobile Node (MN) has created a new content (*data1*) to be published and then publishes the ni: names of the content (*ni://data1_id*) together with the content itself (message 1). Once the next-hop NNRP receives the PUBLISH message, it forwards the message to the RVS (message 2), which then creates a new local mapping with the received parameters and caches the content. After this, the RVS sends a response message back to the origin MN via the intermediate NNRP (message 3 and 4).

For a MN to retrieve *data1*, it queries the infrastructure by sending a GET message towards the RVS indicating interest in the *data1* with the *ni://data1_id* reference. The NNRP on the path towards the RVS intercepts the GET message and looks for potential cached identical data as in the request. In case there are no copy available in the NNRP the GET message is forwarded towards the RVS as message 2 in Figure 5.2. The RVS replies (see message 3) with the actual data and the downpath NNRP can cache the *data1* in its local cache. The final delivery of the data is from the NNRP to the MN with message 4.

[2] http://lucene.apache.org/

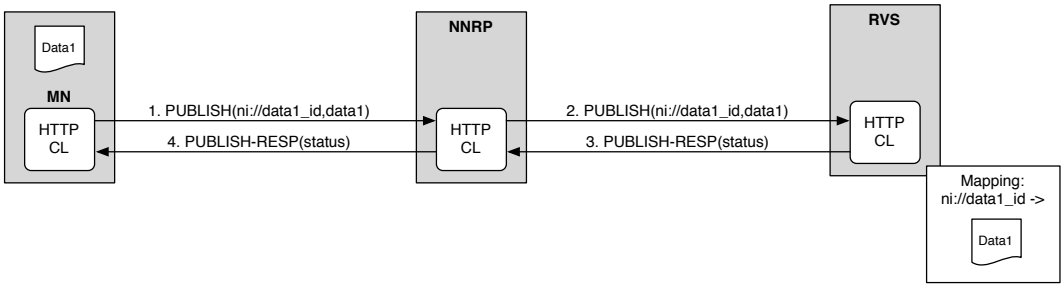


Figure 5.1: Example of how MN publishes NDO to infrastructure service (RVS).

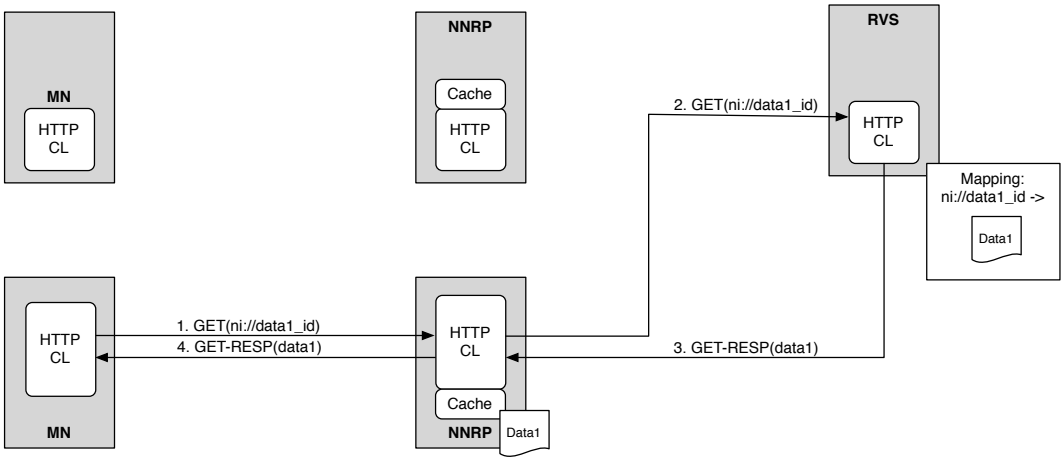


Figure 5.2: Example of how MN queries the published content from infrastructure service (RVS).

Finally, Figure 5.3 describes the situation where the actual requested *data1* (message 1) is available at an upstream NNRP. The GET message is received at the NNRP and the cached *data1* is delivered with message 2.

5.3.6.2 Publish Locator(s) to Infrastructure Service

In this section, we illustrate examples of how to use the NetInf protocol and ni: naming scheme in the EwLC scenario to publish locator(s) to RVS and to query and fetch published content from NNRP, RVS and MN.

Figure 5.4 shows how a MN has created a new content (*data1*) to be published and then publishes the ni: names of the content (*ni://data1_id*) together with the locators (*loc1* and *loc2*) via which the content can be queried (message 1). Once the next-hop NNRP receives the PUBLISH message, it forwards the message to the RVS (message 2), which then creates a new local mapping using the received parameters in the PUBLISH message. After this, the RVS sends a response message back to the origin MN via the intermediate NNRP (message 3 and 4).

Once a MN has published its content, other MNs can query it as illustrated in Figure 5.5, where the MN queries the content with its ni: name (*ni://data1_id*) (message 1). Once the message has been received by the next-hop NNRP, it is then forwarded to the RVS (message 2), where the local mappings are searched for the given ni: name and the resulting locators (*loc1* and *loc2*) are then replied back to the MN via the intermediate NNRP (messages 3 and 4). After this, the MN selects the best candidate locator, which would be *loc1* in this case, and sends another query using the selected locator (message 5). The next-hop NNRP intercepts this and forwards it to other NNRP that is a next-hop node towards the destination MN (message 6). The receiving NNRP

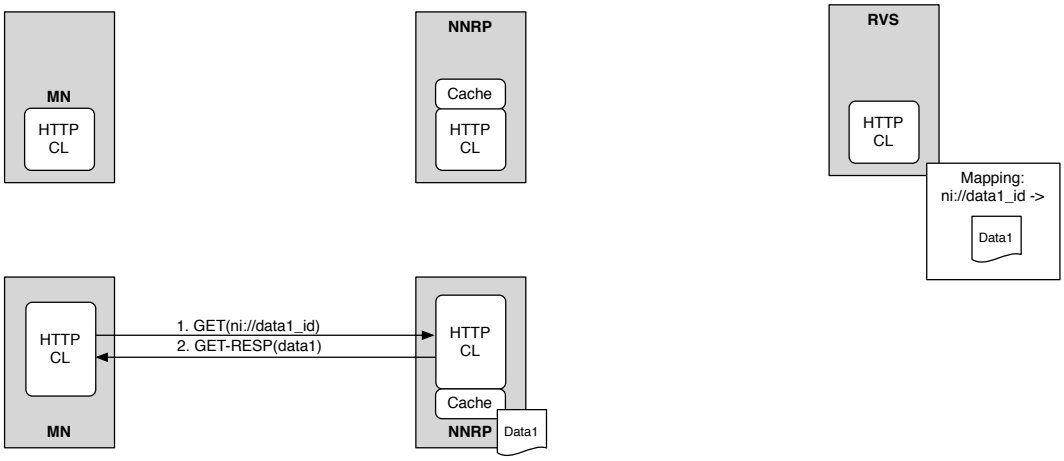


Figure 5.3: Example of how MN queries the published content that is cached in NNRP from infrastructure service (RVS).

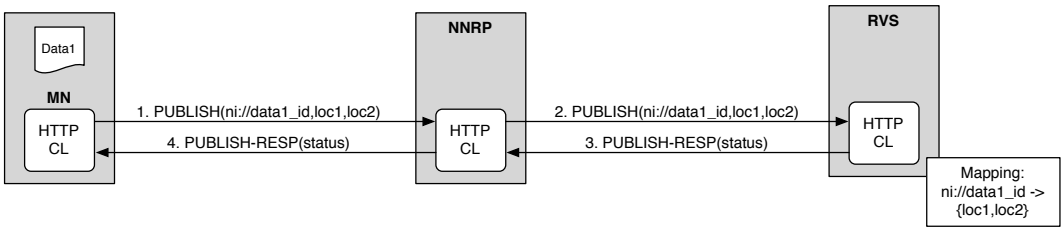


Figure 5.4: Example of how MN publishes locator(s) to infrastructure service (RVS).

then forwards the message to its destination MN (message 7). The destination MN checks that it really has the requested content and after that it sends a response towards the query origin MN with the requested content (*data1*) (message 8). The next-hop NNRP receives the message and decided to cache the content in the response and then forward it to the next-hop NNRP (message 9). Similarly, also other NNRP decides to cache the content from the message and after that it forwards the message to its destination MN. Afterwards, the requested content (*data1*) is cached in both NNRPs so any subsequent request for the same content can be served from NNRP caches as represented in Figure 5.6.

5.4 Evaluation

After integrating the prototypes of project partners to realize the EwLC scenario, we also want to evaluate the performance of doing this with the NetInf architecture. In doing so, there is a need for a framework within which the evaluations take place (in particular as the characteristic of the scenario is that of a very large scale that makes it difficult to assemble in a lab environment), as well as guidelines for which particular use cases that are of interest to study to determine what the potential benefits of using NetInf are. In this section, we describe the simulation and emulation framework that we will be using for our evaluations and enumerate the basic use cases that will be studied.

5.4.1 Simulation and Emulation Framework

In order to actually confirm the benefits of using NetInf in the scenario outlined above, it is important to be able to actually test the protocols and prototypes in such a scenario. Even though

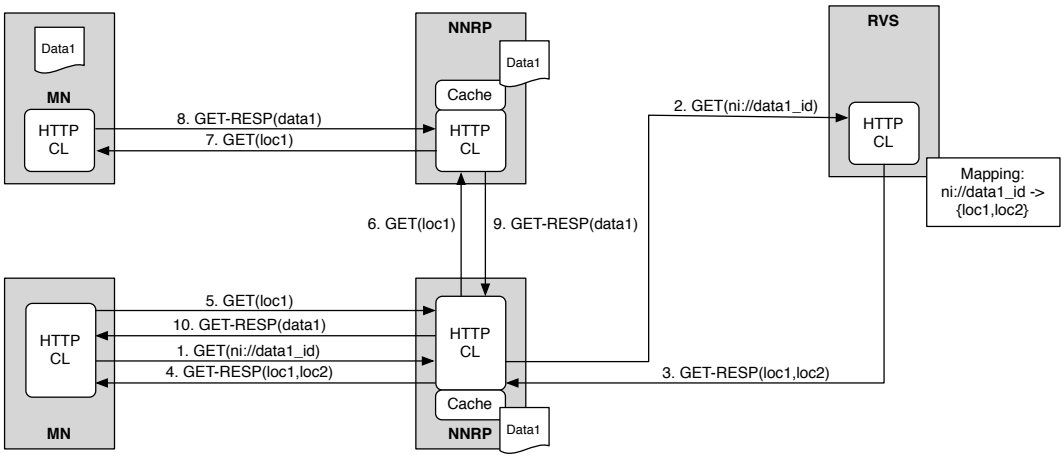


Figure 5.5: Example of how MN queries the published content from infrastructure service (RVS) to get the locator and then retrieves the NDO from the publishing MN.

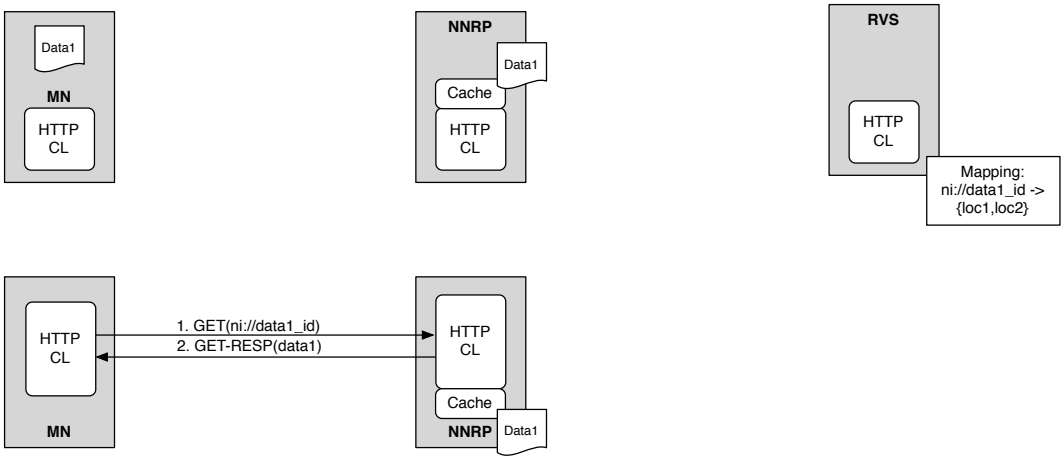


Figure 5.6: Example of how MN queries the published content that is cached in NNRP and that is directly served from there.

most components required to realize the scenario are implemented (or have defined plans for their implementation), a key feature of this scenario is that it requires a large number of nodes. This makes it impractical to set up a real deployment of the scenario, both in terms of hardware costs and logistics. Therefore, we are planning on using a simulation and emulation framework based on the ns-3 network simulator[39] and some work done in the N4C FP7 project [40] to facilitate the use of real software in the simulated environment. This environment allows us to simulate a network using the popular and commonly used ns-3 network, while using a light-weight virtualization to allow us to run real prototype software on the nodes in the simulated network. This makes it easier to set up and test scenarios that contain a large number of nodes. The framework also makes it possible to connect applications running on real hardware to the simulated environment. This feature will be used to confirm the functionality of mobile devices, but in terms of large-scale evaluations, we will mostly rely on simulated and emulated nodes.

5.4.2 Evaluation Use Cases

To evaluate the performance and benefits of NetInf under the EwLC scenario, we also define a number of evaluation use cases that showcase the benefits of using NetInf over the traditional Internet architecture. When evaluating a network architecture, it is important that the concrete use cases are realistic, but at the same time it is vital to keep the complexity low so that the results can be easily interpreted. We have chosen to focus on some use cases that we believe are likely to occur in an event with a large crowd and that show some specific, quantifiable benefits of NetInf.

We will also look into more advanced mobility and data traffic generating models for potential larger-scale tests. The main focus of the evaluations will however be on the use cases below:

Global event, official coverage A ”globally” (within the arena) interesting event occurs. This could for example be a goal being scored at a football game. A large percentage of users want to access the official instant replay movie clip of this event (the content is generated by a central entity, so it can be assumed to be located in infrastructure). This creates a high demand in the network – the use of caching in the network will help reduce the overall load and improve performance. All users will not request the content at the same time, but requests will be somewhat delayed (accesses distributed over the next few minutes range or so after the event based on an exponential distribution). This will showcase the benefits of NetInf and the inherent “time shift” capabilities provided by the architecture over traditional multicast solutions.

Global event, customized coverage When a global event occurs, there may be different content covering the object. There might be different official camera angles of the event, and in addition there is likely to be a plethora of user-generated content available from event attendees with smartphones (i.e., everybody recording the event from their particular location and location-tagging that content). Users want to see a different view of the event and based on the distributed map/content directory requests official and/or user generated content from other parts of the arena.

Local event An event that is only of interest for other users in the same local area of the arena occurs. This may create an increase in the network traffic locally, but we will also be able to show that this exchange can potentially be done without having to involve the infrastructure. By using a local multi-cast based convergence layer and localized multi-cast/broadcast name resolution system, it is possible for mobile nodes to locate the content that is available in the local neighbourhood and exchange that data using local wireless technologies. If multiple such events occur throughout the arena, the effects will thus be contained locally and not cause extra strain on the infrastructure.

Viral content Viral content based on recommendations by users (e.g., in a chat application or discussion forum).

5.5 Summary and Next Steps

In this chapter we have defined the Event with Large Crowd scenario which will be used for showcasing the technical use of NetInf functionalities. We provided an overview of the scenario and some of the problems that exist with the operation of current Internet architectures in that scenario. The chapter has also explained how the prototyping activities of the project partners fit into this scenario and which of the required functionality each component implements. Some examples of how the different components will be integrated to provide a complete and fully functioning system are also given.

The work described in Section 5.3 is ongoing and the implementation and integration efforts will continue to create a fully functional and integrated NetInf system that can be deployed in an EwLC scenario. The plan is the use the evaluation framework in Section 5.4 to set up some representative instances of the EwLC scenario and use that framework to demonstrate NetInf functionality and investigate its benefits over other networking technologies.

6 Conclusion

NetInf is an ICN approach that is being developed by nine partners, three vendor companies, three telecom operators, one research institute and two universities. This deliverable provides a current snapshot of this work and points to more detailed material that documents the results so far.

Summary We have described the NetInf foundations that are based on a selected set of invariants, outlining important guidelines for how the NetInf ICN should work in general. With these invariants as corner stones, we have defined the NetInf Protocol as the general *conceptual* protocol that is supported by *NetInf nodes*, i.e., the entities in NetInf networks. The NetInf Protocol provides the *Convergence Layer* concept as a key element, which is one enabling element for the multi-technology and multi-domain support in NetInf. Another enabling element is the *Global Connectivity and Inter-Domain Interfaces* approach.

One of the interesting aspects of ICN from a network perspective is that it provides new, sometimes more adequate, ways to achieve required functions such as resource control. The possibility for in-network caching as a network node feature and the request/response semantics promote a more receiver-driven approach that can affect the design of transport protocols/strategies but that can eventually also affect the way that some network management functions are realised and the way that networks are finally operated. To build networks that are not only different but also provide the promised performance benefits the design and validation of new content transport and placement mechanisms is necessary.

First of all it is important to understand the implications and opportunities of the NetInf Convergence Layer approach that we have analysed in Section 3.2.1. Secondly, NetInf needs transport protocols that can provide good throughput and reasonable latency in a network with potentially many per-hop caches and Convergence Layer links. We have analysed the performance of a receiver-window and delay-based transport protocol for such networks. For caching, and the way that caching is done in detail – both considering a single node and a network of caching nodes – ICN provides different options for fine tuning and for overall network design. We consider those the truly interesting *network management* aspects of ICN and have developed a concept for realising corresponding functions in NetInf.

For learning how to manage caching in ICN, we have analysed the effectiveness of in-network caching in access network environments, taking the performance and availability of current storage technology into account. One lesson learned was that for collaborative caching LRU may be a better displacement strategy than LFU. However, performance and complexity considerations may advocate the simpler RANDOM displacement strategy, especially for higher layer caches that serve many aggregated requests.

Reactive caching in ICN is by itself not particularly different from web caching (or similar caching applications). In order to leverage the true potential of ICN, it is useful to exploit specific mechanisms of the ICN concept in general and of the NetInf protocol in particular. We have therefore developed an approach by which the network (and its caches) can be *informed* about NDO properties, specifically about indications of NDO popularity in order to adapt caching behaviour. A first analysis of our approach – *Forward Meta Data* – showed promising results when compared against LFU and LRU, so we plan to extend our work on this. We see this as the NetInf way of network management, i.e., leveraging the NetInf object model and the NetInf protocol for request and NDO transfer *to inform the network for doing something useful.*

When analysing caching performance in a specific NetInf setup such as the GIN system, collaborative caching can be enabled by leveraging the properties of a hierarchical name resolution service. We have undertaken a detailed analysis of where caches could and should be placed to achieve optimal performance in terms of hit ratio, latency and on-net traffic ratio. This very detailed analysis illustrates how caching strategies and cache placement decisions can be made – leveraging specifics of the ICN network and its name resolution mechanisms.

Whereas ICN for content distribution is an acknowledged concept, there has not been much work done so far on analysing the feasibility of using ICN for hosting in-network services and on reasoning about optimal placement decisions. We have started first conceptual and analytic work on this topic and have formulated the *service placement problem* for finding optimal service locations considering certain cost functions for storage and communication resources. While this work is still at an early phase, it can be seen as step towards understanding ICN network optimisation potential – which could be used to assess real-world protocol performance.

However, when building solutions for real-world networks and especially for evolving the Internet, it is indispensable to validate research ideas and analytic results by experiments with real implementations, i.e., running code. Calculations and simulations can help to understand performance characteristics and trade-offs of specific algorithms, but this work is of limited value if it cannot be implemented in a system context. In SAIL NetInf feedback from prototyping and experiments has helped to increase the overall quality of the design.

We are using the *Event with Large Crowd* demonstration scenario for that purpose. This work is unique in two ways: 1) it involves the definition of a non-trivial real-world application scenario that requires many of the NetInf protocols and algorithms that have been presented in this document; and 2) it is actually being implemented in a significant integration effort of different partner components. What may seem as unnecessary effort is actually a *required* proof-of-concept that helps us to advance the NetInf protocol specifications and the different optimisation algorithms because we can run them on a test-bed with configurable and reproducible properties – which greatly adds to the quality and credibility of the overall approach.

NetInf in SAIL NetInf is one of three "Scalable and Adaptive Internet Solutions" approaches in the SAIL project. The two other ones are Open Connectivity Services [41] and Cloud Networking [42].

Open Connectivity Services (OConS) provides a framework of resource allocation and management in different types of networks. One of the approaches there is to collect knowledge about the network and its current state (e.g., utilisation) to optimise network attachment and resource usage decisions. NetInf can benefit from such services, for example for supporting multi-interface handling on NetInf node, in particular for doing multi-path decisions. On the other hand, it is also interesting to contrast the OConS vision from the NetInf vision: Whereas in OConS, information about a larger scale network status is explicitly collected, processed and finally used to perform "intelligent" decisions about resource usage, NetInf is actually rather trying to leverage *implicit* information, for example delay information in transport protocols, heuristics about NDO popularity for cache control and local, per-node observations for rate control and interface selection. One objective of NetInf in that sense is to simplify network operation by avoiding the necessity of having that explicit information.

Cloud Networking (CloNe) is based on the ideas to integrate virtual networking and cloud computing by providing single management abstraction of combined resource "slices". This can, for example, be used to provide cloud resources on demand, with guaranteed computing, storage and communication capacity. Such a system can be used for many applications, including content distribution. Assuming network virtualisation is perfect, NetInf can certainly run on-top of a CloNe network, and, assuming that NetInf routers can run on CloNe execution platforms, it would be

possible to dynamically create complete NetInf networks with CloNe. First experiments towards this are being worked on in SAIL. Again, on the other hand, CloNe and NetInf can also be seen as two alternative approaches to address dynamically changing demand for content distribution resources. By re-active and pro-active caching a NetInf system would also be able to increase its resource usage on demand and actively placing NDOs into the network would automatically lead to a different network utilisation as nodes start directing requests to newly created replicas that get announced through the name resolution or name-based routing system.

In SAIL, we are investigating both the complementary aspects of the approaches and the differences in achieving the same objective.

Next Steps The NetInf architecture and protocol have reached a sufficiently stable state, and we expect only incremental changes. We will still be working on specific Convergence Layers, Inter-Domain Interface details, and of course on additional transport and caching mechanisms.

Particular emphasis will be put on the integration and evaluation work. As we move forward with the Event with Large Crowd scenario, we expect valuable feedback for the architecture and content distribution mechanisms. We also plan to evaluate NetInf's applicability and performance characteristics for other, non-content-distribution applications, such as interactive real-time services.

Appendices

SAIL has defined the following NetInf specifications so far:

- the NetInf Protocol (see Appendix A) that specifies the conceptual NetInf protocol and the Convergence Layer concept;
- the ni: Name format specification (see [3] and [4]);
- the HTTP and UDP Convergence Layer specifications that are described in Chapter 2 and that will be published later; and
- the Bundle Protocol Query Extension Block specification (see [43]) that specifies elements of a DTN Convergence Layer.

Appendix A: Specification of NetInf Conceptual Protocol

The following text will be published as an Internet Draft and submitted to the newly formed Information Centric Networking research group of the Internet Research Task Force (IRTF).

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 21, 2012

D. Kutscher
C. Imbrenda
NEC
S. Farrell
E. Davies
Trinity College Dublin
C. Dannewitz
Universitaet Paderborn
December 19, 2011

The NetInf Protocol
draft-netinf-proto-00

Abstract

This document defines a conceptual protocol and corresponding node requirements for NetInf nodes in a NetInf network. A NetInf network offers an information-centric paradigm that supports the creation, location, exchange and storage of Named Data Objects (NDOs). NetInf nodes can provide different services to other NetInf nodes, e.g., forwarding requests for information objects, delivering corresponding response messages, name resolution services etc. This (abstract) protocol is intended to be run over some "convergence layer" that handles transport issues. A "wire" format is not (yet) provided.

Status of this Memo

This Internet-Draft is submitted in full conformance with the

provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 21, 2012.

Table of Contents

1. Introduction

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. [RFC2119]

Syntax definitions in this memo are specified according to ABNF [RFC5234].

2. Principles and Assumptions

A NetInf network provides an information-centric networking (ICN) environment in which units of data content can be identified and accessed using a URI-based naming scheme. NetInf nodes in a NetInf network support the creation, location, exchange and storage of these

units of content. In order to support interoperable implementation of the NetInf design, [ref.netinf-design] the following assumptions are made here:

- all nodes can take on all NetInf roles (but do not have to);
- as necessary, nodes may access a Name Resolution System (NRS) and/or a (possibly name based) message routing infrastructure for NetInf messages; and
- the NetInf protocol can be used directly to access content.

The NetInf protocol operates on Named Data Objects (see [ref.netinf-design]) referred to as NDOs or just DOs. An NDO is an ordered collection of octets associated with a name. The NetInf protocol is designed to cache, locate and transmit complete NDOs.

The NetInf protocol is specified so that NDOs can in principle be retrieved from nodes anywhere in the network to which messages can be routed. This routing is intended to be driven by the names of the NDOs, with the option to use an NRS, but this specification does not discuss how routing, nor calling to an NRS, is carried out. Routing will also depend on the underlying Convergence Layer protocol (see Section 3) in use at that node.

Nodes offering NetInf services may return locators in some cases, but locators may not be usable outside of some (possibly hard-to-characterise) domain, or for more than a limited period of time, due to mobility of nodes or time limited access through 'pinholes' in middleboxes such as firewalls and Network Address Translators (NATs). Accordingly, a design goal is to enable preferential use of names, with locators mostly as hints to improve efficiency. For this reason one can argue that locators ought not be made available to applications using the NetInf protocol in a form that would allow them to try to use the locator outside the NetInf protocol. NDOs may have multiple locators in order to indicate specific interfaces or to reflect attachment to multiple addressing domains. Locators also typically map to a specific instance (copy) of an NDO residing at a given host.

NDO names will often be based on hash-function output values, and since the preferred hash-function will change over time and may change depending on location, this implies that NDOs can also have more than one name. There may also be cases where truncated hash values are desired (e.g. in cases where packets must be kept below some small size and fitting an entire request into one packet is required), and in such cases collisions will occur, or can easily be generated by bad actors. There are also cases where it is desirable to use a name to refer to some "dynamic" NDO, whose octets change (e.g. perhaps the current weather report) and there are cryptographic methods for doing this. This all means that there is no strict 1:1 mapping between names and NDOs, however, we do expect that for most objects, for most ICN deployments, there will in practice be one NDO that is named by each name. That is, each name usually does refer to just one object, and this protocol is designed to work best for that case.

The following NetInf services are assumed to be implemented on nodes through the NetInf protocol:

- caching of NDOs, both locally originated and acquired through

network operations with the NetInf protocol;

requesting the fetching of an NDO using its name, possibly with the addition of a locator, from elsewhere in the network;

responding to NetInf protocol NDO fetch operations using a name referring to one of its locally known NDOs, which may have been locally generated or acquired from another NetInf node and cached here, by returning either or both of the data named in the operation or locator(s) referring to a node where that NDO is (assumed to be) available;

initiating a search for NDOs matching specified search criteria;

responding to search requests received by determining if any locally known NDOs meet the search criteria according to locally determined algorithms;

NDO publication via sending out the name and, optionally, the associated data to other nodes;

according to locally determined policy, the ability to accept or reject NDO publication requests that are delivered to the node, and to cache either or both of the objects and/or information about those that are accepted;

according to locally determined policy, after carrying out local processing, the ability to forward NetInf messages to other nodes or discard them.

local cache management, driven by local policy and (optionally) whatever cache directives are carried in NetInf messages

3. Convergence Layer Architecture

The idea of the Convergence Layer (CL) is to provide a means to transport NetInf messages between pairs of nodes that offer NetInf services. Any protocol that allows NetInf messages to be passed without loss of information can be used as a NetInf Convergence Layer (NetInf–CL) protocol.

This document does not cover the bit–level specification of any CL protocol. The individual CL protocols will provide their own specification regarding their bit–level format.

Different CLs can be used in the various regions forming a global NetInf network. Where a message has to pass through several intermediate NetInf–capable nodes from source to destination, the NetInf protocol layer at each node is responsible for selecting the appropriate link and CL to forward messages.

Each CL has to offer the following minimal set of capabilities:

- unidirectional point–to–point transport of NetInf messages from source to destination,
- preservation of message boundaries,
- reliable transmission of message octets, and

o in-order delivery of message octets to the destination node.

If an underlying protocol used by a particular CL cannot offer these capabilities natively, then the CL is responsible for synthesising these capabilities by appropriate means, e.g., use of retransmission or insertion of sequence numbers. However, this does not prevent a CL that uses a more capable underlying protocol from implementing additional capabilities, e.g., bidirectional connections that allow a single connection to send NDOs in both directions.

The CL itself does not specify the properties of the messages, how they are interpreted, and the way nodes should interact with them, as that is what is specified in the present document.

The CL architecture is inspired by, and similar to, the concept used in Delay-Tolerant Networking. [RFC4838][RFC5050].

However, in contrast to DTN-CLs, the NetInf-CL concept does not include the handling of fragments of an NDO "above" the CL. This is the main difference between the CL concept as used in DTNs and ICNs. Put another way, a DTN-CL may result in a bundle being fragmented, and those fragments are only re-assembled at the final bundle destination. In the case of an NetInf-CL, if an NDO is fragmented or chunked within the CL, then those fragments or chunks are reassembled at the next ICN node and that fragmentation or chunking is not visible to the ICN protocol. One can also consider that the DTN Bundle Protocol (BP)[RFC5050], which runs over a DTN-CL, can itself, with an appropriate extension such as the "BPQ" extension, [I-D.farrell-dtnrg-bpq] be an NetInf-CL. That is, a concrete instance of this protocol could use the BP with the BPQ extension as an NetInf-CL.

4. The NetInf Protocol - Overview

This protocol assumes that NDOs are named using URIs, and in particular via the "ni" URI scheme [I-D.farrell-decade-ni] which MUST be supported. There are a set of extensions to the "ni" URI scheme [I-D.hallambaker-decade-ni-params] that MAY be supported by nodes. However, other URI forms MAY also be used in the NetInf protocol, in particular as locators, and nodes SHOULD support at least fetching of "http" URLs.

Nodes are assumed to be capable of discriminating between names and locators, based on the URI scheme or otherwise.

The most common operations for a NetInf node will be fetching (using a GET message) an NDO or responding to such queries. The response to the GET message will, if possible, contain the octets making up the specified NDO and MAY contain one or more URIs (typically locators) that could subsequently be used to retrieve the octets of the NDO either via this NetInf protocol or by alternative, locator-specific, means. There are some circumstances in which it MAY be appropriate for the response to the GET message to contain only one or more locators. Examples of this situation occur if the responding node is aware that the object content can be returned more effectively using an alternative protocol or from an alternative source because of bandwidth limitations on the links connecting the responding node.

GET, PUBLISH and SEARCH messages MAY be forwarded by any node that receives them if there is good reason and local policy indicates that

this would not result in excessive usage of network resources.

If a request message is forwarded, then a response message MUST NOT be sent for that request while the overall "transaction" is still in progress. That is, a node that forwards a request does not answer that request itself until it gets an answer from elsewhere.

Response messages MUST be forwarded by routers to the node from which the corresponding request message was received. The routing mechanisms that are used to ensure responses are correctly forwarded in this way are not specified here.

Since this specification does not determine how message routing, nor use of an NRS is done, we do not otherwise specify how or when messages are to be forwarded.

Nodes that want to make a locally stored NDO available with a specific name can use the PUBLISH message to announce that data to the network. This message MAY "push" the octets of the NDO into other nodes' caches. (If those nodes are willing to take them.) The reasoning behind this is that in many circumstances pushing just a name or a locator will not be helpful because the node with the NDO may be located behind a middlebox that will not allow access to the data from "outside." Pushing the complete NDO to a node that is accessible from the originating node but is also accessible from outside the middlebox "interior," can allow global access, e.g., by caching the NDO on a server in the DMZ ("DeMilitarized Zone") of an enterprise network or in a server provided by a home user's ISP.(Internet Service Provider).

As in the case of routing messages generally, this specification does not determine the node(s) to which an NDO can be "pushed."

Finally, NetInf nodes can send a SEARCH message to other NetInf nodes. In response, a NetInf node can perform a local search (i.e., of its local cache) As a response, any of the NetInf nodes that receives the SEARCH message returns a set of "ni" URIs of objects matching the search query. It may also return other types of URI such as "http" URIs. Searching of a node's local cache is the main goal for the SEARCH operation, but if a set of nodes were to forward SEARCH messages, then as a global search (e.g., a Google-like service) service could be offered.

NDOs together with any associated metadata are represented using MIME objects. [RFC2045]. Placing as much of the metadata linked to the NDO in a multipart MIME object along with the octets of the actual object allows for significant specification and code re-use. For example, we do not need to invent a new typing scheme nor any associated registration rules nor registries.

As an example we might have a MIME object of that is multipart/mixed and contains image/jpeg and application/json body parts, with the named image in the former and loosely structured meta-data in the latter. The "ni" scheme parameters draft discusses such examples. This means that the details of the verification of name-data integrity supported by the ni name scheme also depend on the MIME type(s) used.

MIME also simplifies the specification of schemes that make use of digital signatures, reusing techniques from existing systems including Secure MIME (S/MIME) [RFC5751]and the Cryptographic Message

Syntax (CMS) [RFC5652].

Note that (as specified in [I-D.farrell-decade-ni]) two "ni" URIs refer to the same object when the digest algorithm and values are the same, and other fields within the URI (e.g. the authority) are not relevant. Two ni names are identical when they refer to the same object. This means that a comparison function for ni names MUST only compare the digest algorithms and values.

5. Protocol Details

We define the GET, PUBLISH and SEARCH messages in line with the above. GET and PUBLISH MUST be supported. SEARCH SHOULD be supported. Each message has an associated response.

This means that GET and PUBLISH are MUST be implemented and SEARCH SHOULD be implemented. In terms of services, GET and PUBLISH SHOULD be operational but SEARCH MAY be turned off.

5.1. GET/GET-RESP

The GET message is used to request an NDO from the NetInf network. A node responding to the GET message would send a GET-RESP that is linked to the GET request using the msg-id from the GET message as the msg-id for corresponding GET-RESP messages if it has an instance of the requested NDO.

The "ni" form or URI MUST be supported. Other forms of URI MAY be supported.

The msg-id SHOULD be chosen so as to be highly unlikely to collide with any other msg-id and MUST NOT contain information that might be personally identifying, e.g., an IP address or username. A sufficiently long random string SHOULD be used for this.

The ext field is to handle future extensibility (e.g. for message authenticators) and allows for the inclusion of a sequence of type, length value tuples. No extensions are defined at this point in time.

```
get-req = GET msg-id URI [ ext ]
get-resp =  status msg-id [ 1*URI ] [ object ]  [ ext ]

ext = 1*(type length value)
```

Figure 1: GET/GET-RESP Message Format

Any node that receives a GET message and does not have an instance of the NDO referenced in the message MUST either

- forward the message to another node, or
- generate a GET response message with an appropriate status code and the msg-id from the GET message as the response msg-id.

If the message is forwarded, the node SHOULD maintain state that will allow it to generate the GET response message if a matching response message is not received for forwarding within a reasonable period of

time after the GET message was forwarded.

If the node has an instance of the NDO, the response MAY contain zero or more URIs that MUST be either locators for the specified object or else alternative names for that object. If the receiving node has a copy of the relevant object in its cache it SHOULD include the object in the response. Possible reasons for not including the object would include situations where the GET message was received via a low-bandwidth interface but where the node "knows" that returning a locator will allow the requestor faster access to the object octets.

The object MUST be encoded as a MIME object. If there is metadata associated with the object this MUST also be encoded using MIME and integrated with the object in a multipart/mixed MIME object.

If the receiving node does not have a cached copy of the object it MAY choose to forward the message depending on local policy. Such forwarding could be based on name-based routing, on an NRS lookup or other mechanisms (e.g. a node might have a default route).

If an get-resp is received with an object that is not MIME encoded or of an unknown MIME type then that MUST be treated as an application/octet-stream for the purposes of name-data integrity verification.

get-resp messages MAY include extensions as with all others.

5.2. PUBLISH/PUBLISH-RESP

The PUBLISH message allows a node to push the name, and optionally, alternative names, locators, a copy of the object octets and/or object meta-data. Ignoring extensions, only a status code is expected in return.

A msg-id MUST be included as in a GET message.

A URI containing a name MUST be included. The "ni" URI scheme SHOULD be used for this name.

The message MAY also contain additional URIs that represent either alternative names or locators where the identical object can be found. As mentioned in Section 4 it is the responsibility of the receiving node to discriminate between those URIs used as names and those used as locators.

The object octets MAY be included. This is intended to handle the case where the publishing node is not able to receive GET messages for objects. An implementation SHOULD test (or "know") its local network context sufficiently well to decide if the object octets ought to be included or not. Methods for checking this are out of scope of this specification.

A node receiving a PUBLISH message chooses what information from the message, if any, to cache according to local policy and availability of resources.

One way to "fill a cache" if the object octets are not included in the PUBLISH would be for the recipient of the PUBLISH to simply request the object octets using GET and cache those. (There is no point in sending a PUBLISH without the octets and without any locator.) This behaviour is, of course, an implementation issue.

In some cases it may make sense for a (contactable) node to only publish the name and meta-data about the object. The idea here is that the meta-data could help with routing or name resolution or search. Since we are representing both NDO octets and meta-data as MIME objects, we need to tell the receiver of the PUBLISH message whether or not that message contains the full object. We do this via the "full-ndo-flag" which, if present, indicates that the PUBLISH message contains enough data so the receiver of the PUBLISH message has sufficient data to answer a subsequent GET message for that name.

Extensions ("ext") MAY be included as in a GET request.

```
pub-req = PUBLISH msg-id 1*URI [ [ full-ndo-flag ] object ] [ ext ]
pub-resp = status msg-id [ ext ]
```

Figure 2: PUBLISH/PUBLISH-RESP Message Format

The response to a PUBLISH message is a status code and the msg-id from the PUBLISH message and optional extensions.

A node receiving a PUBLISH message MAY choose to forward the message to other nodes whether or not it chooses to cache any information. If this node does not cache the information but does forward the PUBLISH message, it should postpone sending a response message until a reasonable period of time has elapsed during which no other responses to the PUBLISH message are received for forwarding. However, the node MAY send an extra response message, even if it forwards the PUBLISH message, if the sender of the PUBLISH message would have expected the receiving node to cache the object (e.g., because of a contractual relationship) but it was unable to do so for some reason.

5.3. SEARCH/SEARCH-RESP

The SEARCH message allows the requestor to send a set of query tokens containing search keywords. The response is either a status code or a multipart MIME object containing a set of meta-data body parts, each of which MUST include a name for an NDO that is considered to match the query keywords.

```
search-req = SEARCH msg-id [ 1*token ] [ ext ]
search-resp = status msg-id [ object ] [ ext ]
```

Figure 3: SEARCH/SEARCH-RESP Message Format

In the case where the response contains an object, the object MUST take the form of an multipart MIME object where each body part is an application/json object, containing a "name" field with a URI as the value of that field. The other fields in each application/json object SHOULD contain meta-data that is intended to allow the requestor to select which, if any, of the names offered to retrieve.

The URIs included in a search-resp SHOULD be names, but MAY be locators, to be distinguished by the requestor as in the case of GET responses.

The intent of the SEARCH message is to allow nodes to search one

another's caches, but without requiring us to fix the details (ontology) for NDO meta-data. While this main intended use-case does not involve forwarding of SEARCH messages that is not precluded.

As with PUBLISH messages, if a SEARCH message is forwarded, the forwarding node postpones sending an empty SEARCH response until a reasonable time is elapsed to see if alternative node responds to the SEARCH.

SEARCH messages MAY include extensions as for other messages.

6. Security Considerations

For privacy preserving reasons requestors SHOULD attempt to limit the personally identifying information (PII) included with search requests. Including fine-grained search keywords can expose requestor PII. For this reason, we RECOMMEND that requestors include more coarse grained keywords and that responders include sufficient meta-data to allow the requestor to refine their search based on the meta-data in the response.

Similarly, search responders SHOULD consider whether or not they respond to all or some search requests as exposing one's cached content can also be a form of PII if the cached content is generated at the behest of the responder.

Name-data integrity validation details are TBD for some common MIME types.

7. Acknowledgements

This work has been supported by the EU FP7 project SAIL.

8. References

8.1. Normative References

[I-D.farrell-decade-ni]
Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", draft-farrell-decade-ni-03 (work in progress), April 2012.

[I-D.hallambaker-decade-ni-params]
Hallam-Baker, P., Stradling, R., Farrell, S., Kutscher, D., and B. Ohlman, "The Named Information (ni) URI Scheme: Parameters", draft-hallambaker-decade-ni-params-01 (work in progress), April 2012.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

8.2. Informative References

[I-D.farrell-dtnrg-bpq] Farrell, S., Lynch, A., Kutscher, D., and A. Lindgren, "Bundle Protocol Query Extension Block", draft-farrell-dtnrg-bpq-01 (work in progress), March 2012.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[ref.netinf-design] Ahlgren, B., D'Ambrosio, M., Dannewitz, C., Marchisio, M., Marsh, I., Ohlman, B., Pentikousis, K., Rembarz, R., Strandberg, O., and V. Vercellone, "Design Considerations for a Network of Information", Re-Arch 2008 Workshop , December 2008.

Authors' Addresses

Dirk Kutscher
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: kutscher@neclab.eu

Claudio Imbrenda
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: Claudio.Imbrenda@neclab.eu

Stephen Farrell
Trinity College Dublin
Dublin, 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie

Elwyn Davies
Trinity College Dublin
Dublin, 2
Ireland

Phone: +44 1353 624 579
Fax:
Email: davieseb@scss.tcd.ie
URI:

Christian Dannewitz
Universitaet Paderborn
Pohlweg 47–49
Paderborn,
Germany

Phone:
Email: cdannewitz@upb.de

List of Abbreviations, Acronyms, and Definitions

ADU Application Data Unit

AIMD Additive Increase Multiplicative Decrease

API Application Programming Interface

AP Access Point

AS Autonomous System

BGP Border Gateway Protocol

BPQ Bundle Protocol Query

BP Bundle Protocol

CL Convergence Layer

CLA Convergence Layer Adapter

CloNe Cloud Networking

CMS Cryptographic Message Syntax

DECADE Decoupled Application Data Enroute

DHT Distributed Hash Table

DNS Domain Name System

DTN Delay-Tolerant Networking

EwLC Event with Large Crowds

FMD Forward Meta Data

GIN Global Information Network

HTTP Hypertext Transfer Protocol

ICN Information-Centric Networking

IO Information Object

IP Internet Protocol

IRTF Internet Research Task Force

ISP Internet Service Provider

LFU Least Frequently Used

LRU	Least Recently Used
MDHT	Multilevel DHT
MIME	Multipurpose Internet Mail Extension
MN	Mobile Node
MTU	Maximum Transmission Unit
NDO	Named Data Object
NNRP	NEC NetInf Router Platform
NRS	Name Resolution System
NetInf	Network of Information
OConS	Open Connectivity Services
P2P	Peer-to-Peer
PDU	Protocol Data Unit
POP	Point of Presence
QoE	Quality of Experience
RND	Random
RTP	Real Time Protocol
RTT	Round Trip Time
RVS	RendezVous Server
SAIL	Scalable and Adaptive Internet Solutions
SPM	Service Program Mobility
SRV	Service
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VoD	Video on Demand
WKU	Well-Known URL

List of Figures

List of Tables

Bibliography

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

[2] SAIL Project. The network of information: Architecture and applications. Deliverable D-3.1, SAIL EU FP7 Project, 2011. FP7-ICT-2009-5-257448/D-3.1.

[3] Stephen Farrell, Dirk Kutscher, Christian Dannewitz, Boerje Ohlman, and Phillip Hallam-Baker. The Named Information (ni) URI Scheme: Core Syntax. Internet Draft draft-farrell-decade-ni-00, Work in progress, October 2011.

[4] Phillip Hallam-Baker, Rob Stradling, Stephen Farrell, Dirk Kutscher, and Boerje Ohlman. The Named Information (ni) URI Scheme: Parameters. draft-hallambaker-decade-ni-params-00, Work in progress, October 2011.

[5] Z. Shelby, K. Hartke, C. Bormann, and B. Frank. Constrained Application Protocol (CoAP). Internet-Draft draft-ietf-core-coap-09, Internet Engineering Task Force, March 2012. Work in progress.

[6] Jarno Rajahalme, Mikko Särelä, Kari Visala, and Janne Riihijärvi. On name-based inter-domain routing. *Computer Networks*, 55(4):975–986, March 2011.

[7] B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751 (Proposed Standard), January 2010.

[8] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *Proceedings of SIGCOMM'07*, Kyoto, Japan, August 27-31, 2007.

[9] Bengt Ahlgren, Matteo D'Ambrosio, Christian Dannewitz, Marco Marchisio, Ian Marsh, Börje Ohlman, Kostas Pentikousis, René Rembarz, Ove Strandberg, and Vinicio Vercellone. Design considerations for a network of information. In *Proceedings of ReArch'08: Re-Architecting the Internet*, Madrid, Spain, December 9, 2008.

[10] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.

[11] 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.

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

[13] S. Burleigh. Compressed Bundle Header Encoding (CBHE). RFC 6260, Internet Engineering Task Force, May 2011.

[14] M. Demmer and J. Ott. Delay Tolerant Networking TCP Convergence Layer Protocol. Internet-Draft draft-irtf-dtnrg-tcp-clayer-02, Internet Engineering Task Force, November 2009. Work in progress.

[15] S. Burleigh. Delay-Tolerant Networking LTP Convergence Layer (LTPCL) Adapter. Internet-Draft draft-burleigh-dtnrg-ltpcl-03, Internet Engineering Task Force, February 2011. Work in progress.

[16] S. Farrell, A. Lynch, D. Kutscher, and A. Lindgren. Bundle Protocol Query Extension Block. Internet-Draft draft-farrell-dtnrg-bpq-01, Internet Engineering Task Force, March 2012. Work in progress.

[17] A. Farrel, J.-P. Vasseur, and J. Ash. A Path Computation Element (PCE)-Based Architecture. RFC 4655 (Informational), August 2006.

[18] Mike Belshe and Roberto Peon. Spdy: An experimental protocol for a faster web. Technical report, Google Inc., 2011.

[19] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. *SIGCOMM Comput. Commun. Rev.*, 20(4):200–208, August 1990.

[20] Matteo D'Ambrosio, Christian Dannewitz, Holger Karl, and Vinicio Vercellone. MDHT: A Hierarchical Name Resolution Service for Information-centric Networks. In *ACM SIGCOMM Workshop on ICN*, Toronto, Canada, 2011.

[21] Matteo D'Ambrosio, Paolo Fasano, Mario Ullio, and Vinicio Vercellone. The global information network architecture. Technical Report TTGTDDNI1200009, Telecom Italia, 2012.

[22] A. Narayanan and D. Oran. Ndn and ip routing – can it scale? Presentation at ICN side meeting at 82nd IETF, November 2011. http://trac.tools.ietf.org/group/irtf/trac/attachment/wiki/icnrg/IRTF

[23] Luca Muscariello, Giovanna Carofiglio, and Massimo Gallo. Bandwidth and storage sharing performance in information centric networking. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ICN '11, pages 26–31, New York, NY, USA, 2011. ACM.

[24] L. Kleinrock. *Queueing Systems: Volume 2: Computer Applications*, volume 82. John Wiley & Sons, 1976.

[25] John Ardelius, Björn Grönvall, Lars Westberg, and Åke Arvidsson. On the effects of caching in access aggregation networks. Technical report, Swedish Institute of Computer Science, SICS and Ercisson Research, 2012. In submission.

[26] Yannick Carlinet, Bruno Kauffmann, Philippe Olivier, and Alain Simonian. Impact of request correlations on the performance of multimedia caching. Technical report, Orange Labs, France, 2012. In submission.

[27] Cisco visual networking index: Forecast and methodology, 2010-2015, June 2011.

[28] Massimo Gallo, Bruno Kauffmann, Luca Muscariello, Alain Simonian, and Christian Tanguy. Performance evaluation of the random replacement policy for networks of caches (14 pages version). http://arxiv.org/abs/1202.4880, 2012.

[29] Erol Gelenbe. A unified approach to the evaluation of a class of replacement algorithms. *IEEE Transactions on Computer*, 22(6):611–618, 1973.

[30] P. R. Jelenković and X. Kang. Characterizing the miss sequence of the LRU cache. *ACM SIGMETRICS Performance Evaluation Review*, 36:119–121, August 2008.

[31] Giovanna Carofiglio, Massimo Gallo, Luca Muscariello, and Diego Perino. Modeling data transfer in content-centric networking. In *Proc. of ITC23*, 2011.

[32] Lee Breslau, Pei Cue, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and zipf-like distributions: Evidence and implications. In *In INFOCOM*, pages 126–134, 1999.

[33] H. Lundqvist, Z. Despotovic, J. Frtunikj, G. Kunzmann, and W. Kellerer. Service program mobility architecture. In *15th International Conference on Intelligence in Next Generation Networks (ICIN)*, pages 23 – 28, Oct 2011.

[34] Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter Brown, and Rebekah Metz. Reference model for service oriented architecture 1.0. Technical Report wd-soa-rm-cd1, OASIS, February 2006.

[35] Georg Wittenburg and Jochen Schiller. Service Placement in Ad Hoc Networks. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, 33(1):21–25, 2010.

[36] SAIL Project. Description of overall prototyping use cases, scenarios and integration points. Deliverable D-2.9, SAIL EU FP7 Project, 2012.

[37] S. Farrell, D. Kutscher, C. ChristianDannewitz, B. Ohlman, A. Keranen, and P. Hallam-Baker. Naming Things with Hashes. Internet-Draft draft-farrell-decade-ni-03, Internet Engineering Task Force, April 2012. Work in progress.

[38] SAIL Project. Demonstrator specification and integration plan. Deliverable D-4.3, SAIL EU FP7 Project, 2012.

[39] The ns-3 network simulator. `http://www.nsnam.org/`.

[40] Antnio Cunha and Paulo Freitas. D7.3 n4c specification and implementation of integration platform. Technical report, N4C FP7 Project, June 2010.

[41] SAIL Project. Architectural Concepts of Connectivity Services. Deliverable FP7-ICT-2009-5-257448-SAIL/D.C.1, SAIL EU FP7 project, July 2011. Available online from http://www.sail-project.eu.

[42] SAIL Project. Cloud Networking Architecture Description. Deliverable FP7-ICT-2009-5-257448-SAIL/D.D.1, SAIL EU FP7 project, July 2011. Available online from http://www.sail-project.eu.

[43] Stephen Farrell, Aidan Lynch, Dirk Kutscher, and Anders Lindgren. Bundle Protocol Query Extension Block. Internet Draft draft-farrell-dtnrg-bpq-00, Work in progress, November 2010.