Objective FP7-ICT-2009-5-257448/D.B.3

Future Networks

Project 257448

"SAIL – Scalable and Adaptable Internet Solutions"

D.B.3
(D-3.3) Final NetInf Architecture

Date of preparation: **2013-01-14**
Start date of Project: **2010-08-01**
Project Coordinator: **Thomas Edwall**
Ericsson AB

Revision: **1.1**
Duration: **2013-01-31**

Document Properties

Document Number:	D.B.3
Document Title:	**Final NetInf Architecture**
Document Responsible:	Dirk Kutscher (NEC)
Document Editor:	Bruno Kauffmann (FT), Jean-François Peltier (FT), Patrick Truong (FT)
Authors:	Bengt Ahlgren (SICS), Matteo D. Ambrosio (TI), Elwyn Davies (TCD), Anders E. Eriksson (EAB), Stephen Farrell (TCD), Hannu Flinck (NSN), Claudio Imbrenda (NEC), Bruno Kauffmann (FT), Gerald Kunzmann (DOCOMO), Dirk Kutscher (NEC), Anders Lindgren (SICS), Ian Marsh (SICS), Luca Muscariello (FT), Hugo Negrette (EAB), Börje Ohlman (EAB), Karl-Ake Persson (EAB), Petteri Pöyhönen (NSN), Jarno Rajahalme (NSN), Mohammed Shehada (DOCOMO), Miguel Sosa (EAB), Dirk Staehle (DOCOMO), Ove Strandberg (NSN), Janne Tuononen (NSN), Vinicio Vercellone (TI)
Target Dissemination Level:	PU
Status of the Document:	Final
Version:	1.1

Production Properties:

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

Document History:

Revision	Date	Issued by	Description
1.0	2012-11-30	Bruno Kauffmann	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.
All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.
For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

Abstract:

This deliverable on the "Final NetInf Architecture" provides an update on the NetInf architecture that resulted from recent prototyping and evaluation activities in the SAIL project. Based on experiences from prototyping individual components, interoperability tests, integration into systems and scalability and performance evaluations as well as community feedback, the NetInf architecture has been advanced. In addition, more aspects of the systems such as specific Convergence Layers and routing mechanisms have been developed and specified.

Keywords:

Network of Information, NetInf, Information-Centric Networking, Internet, SAIL, Architecture, Prototypes, Evaluation

Executive Summary

This document is a public deliverable of the Scalable and Adaptive Internet Solutions (SAIL) EU-FP7 project [1]. It describes the updated Network of Information (NetInf) architecture and the prototyping and evaluation results that led to this update. 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 SAIL NetInf system for Information-Centric Networking provides a highly scalable network architecture with particular support for robustness and reliability. NetInf is designed to be enable multi-technology/multi-domain interoperability and to facilitate initial deployment and migration from today's networks to ICN.

Earlier deliverables have described the NetInf Architecture and Applications and the NetInf Content Delivery and Operations mechanisms. Based on these foundations, 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 March 2012 and that has been advanced further since then.

We have used these prototypes to test features of the system and to evaluate certain technologies such as transport protocols, local ICN communication and name resolution. Different evaluation efforts have been undertaken. Therefore this document has three main contributions: 1) an architecture summary, with a description of recent updates, 2) the description of different prototype developments that implement the architecture and the NetInf specifications, and 3) evaluation results for different aspects of the system.

With respect to the NetInf architecture, the core design principles remained valid and useful, and most of the work has focused on incremental changes and extensions such as the advancements of the NetInf naming scheme and the different Convergence Layer specifications.

The prototyping results include the NetInf Open Source Software implementing key elements of the system for different environments as well as individual partner components (router, router modules for transport and routing, name resolution systems, mobile device applications, and Delay- and Disruption-Tolerant Networking (DTN) prototypes). These components have been integrated into a common scenario — the NetInf *Event with Large Crowd* scenario that feature content generation and delivery in high-density networks. Moreover, the prototyping work includes the self-contained Global Information Network (GIN) NetInf system.

The evaluation results include the following topics:

Throughput, latency and other performance metrics – measuring and characterising the performance of NetInf mechanisms and the overall system, including throughput and latency. The results of this kind of evaluation are contributing to showing the advantages of NetInf for the end-user, and to showing the feasibility of particular mechanisms.

Scalability – simulating and analysing the scalability properties of central NetInf mechanisms, in particular name resolution and routing, to establish the feasibility of the NetInf system for a global scale network.

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

Event with Large Crowds (EwLC) project-wide scenario – showing several aspects of the feasibility and benefit of NetInf in a complex, project-wide, scenario that is difficult to realise with current technology.

Increased efficiency and reduced costs – showing and estimating the increased efficiency of NetInf in terms of reduced need for network capacity, other resources and the resulting lower cost. The results of this kind of evaluation are largely contributing to showing the advantages of NetInf for network operators.

Contents

1 Introduction

The SAIL NetInf system for Information-Centric Networking provides a highly scalable network architecture with particular support for robustness and reliability. NetInf is designed to be enable multi-technology/multi-domain interoperability and to facilitate initial deployment and migration from today's networks to ICN. This deliverable provides an updated NetInf architecture description as well as insights from our recent prototyping and evaluation activities of NetInf.

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 have been documented in the project deliverable D.B.2, which has formulated the architecture invariants more specifically and which has advanced NetInf technologies such as routing and forwarding, transport protocols and caching.

Key NetInf elements have been published as specifications, such as the NetInf protocol specification [2] — a conceptual specification of a NetInf Node-to-Node communication protocol that includes an object model for Named Data Objects (NDOs), a detailed description of the Convergence Layer approach, as well as the specification of HTTP and UDP Convergence Layers. 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 March 2012 and that has been advanced further since then.

We have used these prototypes to test features of the system and to evaluate certain technologies such as transport protocols, local ICN communication and name resolution. Different evaluation efforts have been undertaken. For example, we have built testbeds for running larger numbers of nodes implementing the SAIL Event with Large Crowds (EwLC) scenario, with the objective to analyse the system in different configurations for popularity distribution, object size, cache displacement strategies, mobility models etc. and to analyse possible performance gains over existing web-based communication.

This deliverable The experiences from the prototyping and evaluation activities have been fed back to the architecture and mechanism design work — which has led to a few evolutionary changes and to clarifications in and extensions of the corresponding protocol specifications. For example, this deliverable provides the updates on naming and the NetInf protocol and presents the new *HSkip* NetInf name resolution technology that we have developed and evaluated. It also describes a new

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

Bloom filter based approach for name resolution aggregation and the NetInf architecture support for migration, i.e., its ability to enable a smooth migration from today's networks to ICN.

The merits of an architecture and the derived protocol specification can only be assessed in experiments with *running code in different environments*. Hence, this deliverable describes the different recent NetInf prototype and testbed developments, covering a broad range of developments, including NetInf routers, name resolution systems, applications endpoints, and emulation frameworks.

For *evaluating* systems such as NetInf different aspects of the systems have be to assessed. For example, *scalability* is often referred to as one of the crucial properties of an ICN system. Scalability is important in different ways: in general this refers to the ability of the system to grow to large numbers of nodes, NDOs, and users. One important property to this regard is name resolution and routing scalability, i.e., the ability of the system not only to provide a useful service in smaller, limited deployments but to also manage routing and forwarding in the on the global Internet scale, i.e., with 10^{15} NDOs and more. Routing scalability pertains to several properties: the fundamental ability to deal with the corresponding routing or name resolution state in a way that keeps lookup and forwarding latency within acceptable limits, or, for example, the possibility to build individual network nodes that can store and process the required information fast enough. Furthermore, there are qualitative properties that are especially important for NetInf's ICN approach, for example its resilience against network partitions and its support for intermittent connectivity and long communication delays.

This document is structured as follows: We are describing the corresponding *architecture and NetInf technology updates* in Chapter 2. Chapter 3 provides details on the different prototype developments, and Chapter 4 presents the evaluation results. Chapter 5 concludes this deliverable.

2 Architecture Update

This chapter provides a description of recent developments related to the Network of Information (NetInf) architecture since the previous deliverable [5].

As will be seen, the architecture has not changed significantly over the period, but has mainly stood the limited test of the small amount of elapsed time and prototyping in the past six months. Most effort has been devoted to iterative elucidation of the already-designed NetInf architecture via protocol development and prototyping.

The *NetInf naming scheme* and the associated security services have remained stable, but has been extended to support different name representations and multiple pluggable crypto suites (amongst other features). The *NetInf protocol* has been specified in more detail, especially with respect to the different Convergence Layer specifications. Work on name resolution and routing has progressed – we have advanced the Internet routing mechanisms on explicit aggregation, and we have developed a new DHT-based name resolution approach as well as an NetInf name aggregation mechanisms for exchanging name resolution information in NetInf domain peering scenarios.

In Section 2.1 we present a brief overview of the overall architecture, its design principles and different components. Section 2.2 describes the updates to the naming scheme, and Section 2.3 highlights the recent developments for the NetInf protocol and its Convergence Layers. Section 2.4 presents the design of the HSkip Name Resolution System (NRS) and the name aggregation approach. Finally, in Section 2.5 we explain how the NetInf architecture and the described components can enable initial deployment and smooth migration from today's networks to a future Information-Centric Network.

2.1 Architecture Summary

NetInf is a *networking approach* that provides *access to Named Data Objects (NDOs)* as a first order networking primitive, i.e., the primary service of *NetInf nodes* is the forwarding of NDO requests and the transfer of the corresponding objects (or pointers to copies). This fundamental service model is common to many Information-Centric Networking (ICN) approaches. In this section we summarise the NetInf architecture described previously in Deliverable 3.2 [5] and by Dannewitz *et al* [6]. We describe a set of NetInf design principles and an example message flow in a simple NetInf network.

One of the key NetInf design goals is evolvability and extensibility — in order to enable migration from initial deployments to broader usage. It is important to understand that, from this perspective, it is pointless to design a closed, clean-slate future Internet approach that can only be used in lab environments. For NetInf, we found it more useful to design a system that is based on a set of fundamental ICN design principles, and that provides good support and benefits for todays networks as well as the necessary flexibility to enable migration to more native deployments.

On a related note, it is important to avoid mistakes from ivory tower research in future networking, such as assuming that all networks are the same with respect to properties such as connectivity models, size, stability, latency, throughput, policy etc. We have to acknowledge that today's Internet is already fundamentally heterogeneous with respect to these properties, and *connecting the next billion users* will extend this further.

The NetInf architecture, unlike many other ICN research approaches, addresses these requirements through several design principles. For example, the *Convergence Layer* approach that is mentioned below enables a pluggable support for different networks technologies, overlays and underlays based on one common conceptual model for providing access to named data as a network service. NetInf nodes can implement the same request and response forwarding logic, transport and caching strategies for different networks that they are attached to.

It is important to understand that this does not restrict NetInf to overlay deployments only (such as running NetInf over HTTP). Instead it enables NetInf developers and users to use the technology now, to prototype and advance components such as routers and name resolution services in order to enable a more credible analysis and evaluation of the system's properties through running code.

2.1.1 Design Principles

Accessing Named Data Objects Accessing named data as a first order principle implies that NetInf nodes generally don't communicate on the basis of network or host addresses, but instead use NDO names to identify objects independent of network location. This unique, location-independent naming enables ubiquitous replication and caching of NDOs in the network. An NDO consists of its name in a common format and the actual object in a common data structure. The main service is to forward NDO requests to appropriate copies and transfer objects back.

Minimal common node requirements To broadly apply to different types of networks and deployments, NetInf only makes minimal node requirements:

Naming format: There is one common NDO *naming* format that all nodes understand.

Object model: There is one format for *representing* NDOs and optional metadata. All nodes can process these structures. The format allows for, e.g., application-specific extensions; not all nodes need to understand all extensions or metadata formats.

NetInf protocol: There is one simple protocol that all nodes implement. The NetInf protocol is message-based; it provides requests and responses for `PUBLISH`ing, `GET`ting, and `SEARCH`ing for NDOs. These requests and responses employ the common naming format and the common object model.

Generic NetInf nodes Based on these minimal requirements, NetInf nodes can provide different functions such as forwarding requests and responses, caching, and name resolution. The same functionality can be provided by specialized infrastructure nodes for a global network as well as by user NetInf nodes, e.g., in a local context.

Flat-ish namespace NetInf can employ a flat namespace for NDO names, i.e., there need be neither topology- nor organization-related hierarchy in names. However, NetInf names can also contain hierarchy in the authority component, so there may be something to explain here.

NetInf names are flat when considering name comparison. That is "ni" name matching only takes into account the hash algorithm and hash-value and none of the other bytes in the name URI. This kind of comparison is useful for example when checking for a cache-hit.

NetInf names are not flat when considering routing of messages however, in that case it is entirely reasonable for routing schemes to take into account the content of the authority part of the URI (and possibly other parts of the query-string, such as a scope parameter), when considering how to route requests or do name resolution.

This flat-ish approach does however limit the ability to aggregate names based on a hierarchy, e.g., for routing or name resolution. Explicit aggregation [7], separate from the proper name, is

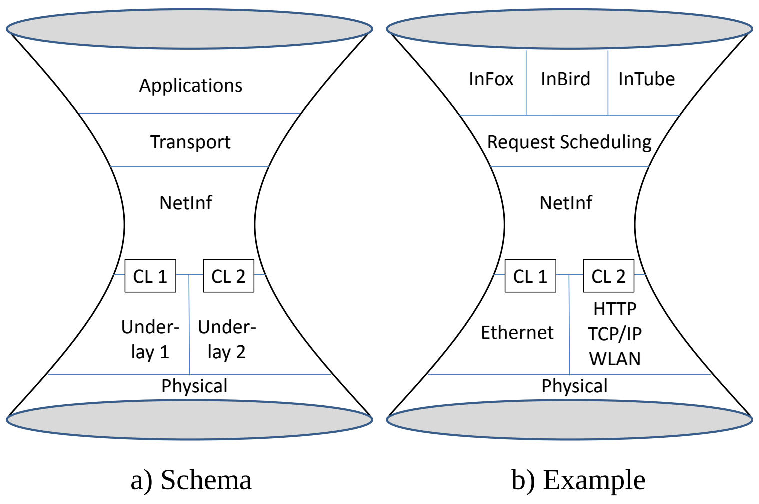


Figure 2.1: NetInf protocol stack, assuming a node with two convergence layers over two different underlays

therefore used to circumvent this problem. On the other hand, for many hierarchical namespaces, the hierarchy is based on aspects like organizational structures, folder structures, etc. Compared to such hierarchical namespaces, a flat-ish namespace provides better name persistence as it eliminates such fixed interdependencies. Otherwise, e.g., organizational changes would result in name changes. In addition, a flat-ish namespace also has the advantage of separating tussle over trademarks from unique data naming. [8]

With properly designed distributed algorithms for name construction, name management is significantly simplified even without a naming authority to assign names. Naming can rely on statistical uniqueness; rare name collisions can be handled as an error, e.g., by the NRS and can in any case be easily avoided through the use of collision resistant hash functions.

Name–data integrity Name–data integrity validation is NetInf's fundamental object security service for both static and dynamic objects. The common naming format and object model enable data-integrity validation by requesters (or any other node). Name–data integrity validation can be performed without infrastructure support like a Public Key Infrastructure (PKI) by employing hash function outputs (and/or public key digests as part of NDO names in the case of dynamic objects). Not all nodes are required to perform validation. Section 2.2 provides an update of the NetInf Naming Scheme that enables representing NetInf names as well as name–data integrity validation.

Lower layer independence The NetInf protocol specifies messages for node-to-node communication, their semantics, and corresponding node requirements. Different deployments will use different link layers and underlays — with a variation of services and properties. NetInf accommodates this via *Convergence Layers (CLs)* that map the conceptual protocol to specific messages, transactions, or packet exchanges in an existing, concrete protocol. A CL provides framing and message integrity for NetInf requests and responses for communication between two nodes as its main service, but specific CLs can provide additional services as depicted in Figure 2.1.

For example, NetInf-over-IP would require a CL that encapsulates, and potentially fragments and reassembles NetInf messages for transfer in IP packets and validates message integrity (e.g., by CRC). NetInf-over-Ethernet, on the other hand, could be done with a slim CL, but running over layer 2 would restrict the scope of services that a NetInf node could offer, so there are trade-offs when considering which CL(s) to use. Figure 2.1 depicts a schematic and sample NetInf stack for different CLs/underlays; it also shows that between applications and NetInf, additional functions like request scheduling (for implementing flow control, congestion control and other transport layer functions) can be inserted. With CLs, NetInf runs over quite different types of network links, including uni-directional or heavily delay-challenged links.

While specific CLs *can* provide transport protocol functions (reliability, flow control, congestion control), there is a need for transport layer functions across CL links. In NetInf, such functions are implemented on-top of the NetInf layer, e.g., by request scheduling/retransmission decisions. Section 2.3 provides a detailed description of the NetInf protocol and specific CLs.

Routing and name resolution The NetInf protocol's request and response forwarding service requires routing information to decide to which next hop, over which interface a request should be forwarded. For `GET` requests, these decisions are generally based on the name of the requested NDO.

NetInf supports both name-based routing and name resolution, i.e., employing in-network name resolution services that can map NetInf names to network or possibly host identifiers in different namespaces, which we call *routing hints*.

Routing hints indicate where to find copies of the object. Different types of routing hints are supported. Routing hints can be lower-layer host locators that can be directly used to retrieve the object via the underlying network protocol (e.g. TCP/IP), protocol-specific routing hints that can subsequently be used to support name-based routing (e.g., by pointing to the next network that can provide the object), or another NetInf name that allows for indirection.

Notably, NetInf supports a *hybrid* combination of name resolution and name-based routing where at each forwarding step for a given request, either scheme can be freely chosen. Section 2.1.3 describes how this concept would be applied to inter-domain routing.

On-path and off-path caching NetInf supports both on-path caching (on the request/data path) as well as off-path caching. NetInf can integrate any available copy when retrieving data: the original server, redundant copies, as well as replicas stored on user devices (if so permitted). Thereby, NetInf can access the best available copy.

Heterogeneous networks NetInf's flexibility in convergence layers and routing/forwarding schemes allows custom-tailored configurations for different technologies and different administrative domains. As long as NetInf's minimal node requirements are maintained, the NetInf protocol can be mapped to quite different types of network links — from bi-directional message/packet-based point-to-point links to unidirectional broadcast links and intermittently connected links. NetInf also does not assume anything about specific network system architectures — e.g., terminals, access networks, core networks are flexible notions and there is no assumption on *where a network ends*: NetInf user devices can appear as terminals, but they can also provide caching and store-carry-forwarding, thus extending the network or connecting NetInf networks.

Inherent mobility and multi-homing support NetInf inherently makes mobility and multi-homing simpler. Regarding multi-homing, the NetInf architecture puts no limits on the number of interfaces used in parallel to send requests and to receive responses. As all object copies are equal, the order of requests and responses is not critical. To use network resources in an optimized way, several request strategies can be used in NetInf. In some network scenario, e.g., a local setup, broadcasting requests on all interfaces might be optimal. In some other network scenario, an NRS-based solution using only a single interface might be preferred.

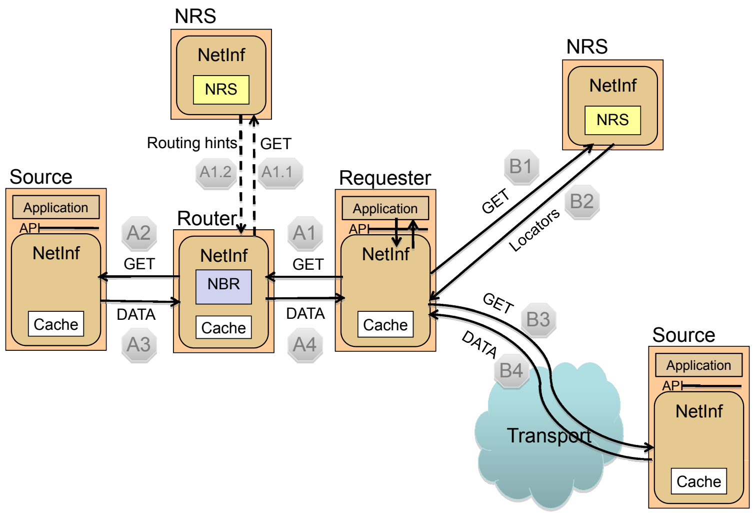


Figure 2.2: Example message flow

There are different types of mobility. For client mobility (i.e., moving requester), there is no need to continue using a specific object copy. Instead, alternative copies close to the client's new location can be used. For server mobility (i.e., moving content), updates of the routing information and/or name resolution information in the network are required to ensure that a data stays accessible.

For traditional point-to-point services (e.g. a voice call) the need for mobility support will be very different in different parts of the network. As NetInf can support different name resolution mechanisms in different parts of the network, these can be selected taking the dynamics of respective network parts into account. While traditional mobility anchor point mechanisms can be used for some parts, very dynamic environments where whole networks are moving (e.g., trains) can rely on mechanisms like Late Locator Construction (LLC) [9].

2.1.2 Sample Message Flow

Figure 2.2 shows an example of name resolution, name-based routing, and the hybrid approach in NetInf. The name-based routing (steps A1–A4) forwards a `GET` request hop-by-hop between NetInf nodes until a cached copy of the NDO is found or the original server (or an alternative server) is reached. If the router does not have enough routing information to perform name-based routing in step A2, it can perform a name resolution step (steps A1.1–A1.2) before forwarding the request (step A2) based on the retrieved routing hints, which illustrates the hybrid mode. On the return path, the object can be cached in intermediate nodes for subsequent requests. Alternatively, the initial requester can query an NRS (steps B1-B4) via a `GET` message to resolve the object name into a set of *routing hints*, in the example simply lower-layer host locators. Subsequently, the routing hints are used to retrieve the object via the underlying transport network, e.g., a legacy IPv4 network.

2.1.3 Name resolution and Routing

NetInf forwards and resolves requests for objects as well as forwards the response messages. Different parts of the network can have different routing requirements and, thus, will need different routing protocols, just as we have multiple routing protocols for Internet Protocol (IP) today. Hence, NetInf supports different request/response routing/forwarding mechanisms, such as Open Shortest Path First (OSPF) for local domains. NetInf implementations should, thus, provide a way to easily plug in new routing, name resolution, or forwarding schemes.

NetInf uses a hybrid request routing/forwarding scheme, integrating pure name-based routing (in the sense of "routing on flat-ish names") with name-resolution-based aspects. The name-based routing could, e.g., simply match names with default routes via prefix or pattern matching. For example, requests from inside a network can reach an "edge" with external access, at which point name resolution is likely to be used. The integration of name resolution and name-based routing also supports, e.g., *late binding*: In heterogeneous domains or in a NetInf Delay- and Disruption-Tolerant Networking (DTN) island, it is often not possible to resolve the object name *at the requester side* into a locator that *has meaning* at the requester side. Instead, the name might be resolved into some routing hint that leads only *towards* the final destination. In this case, the resolution into the final destination locator can be performed by an intermediate NetInf node along the path to the destination. This combination with name resolution enhances scalability and performance of name-based routing.

Routers performing name-based routing (or default route forwarding) can be configured to do request aggregation (similar to CCN's pending interest management).

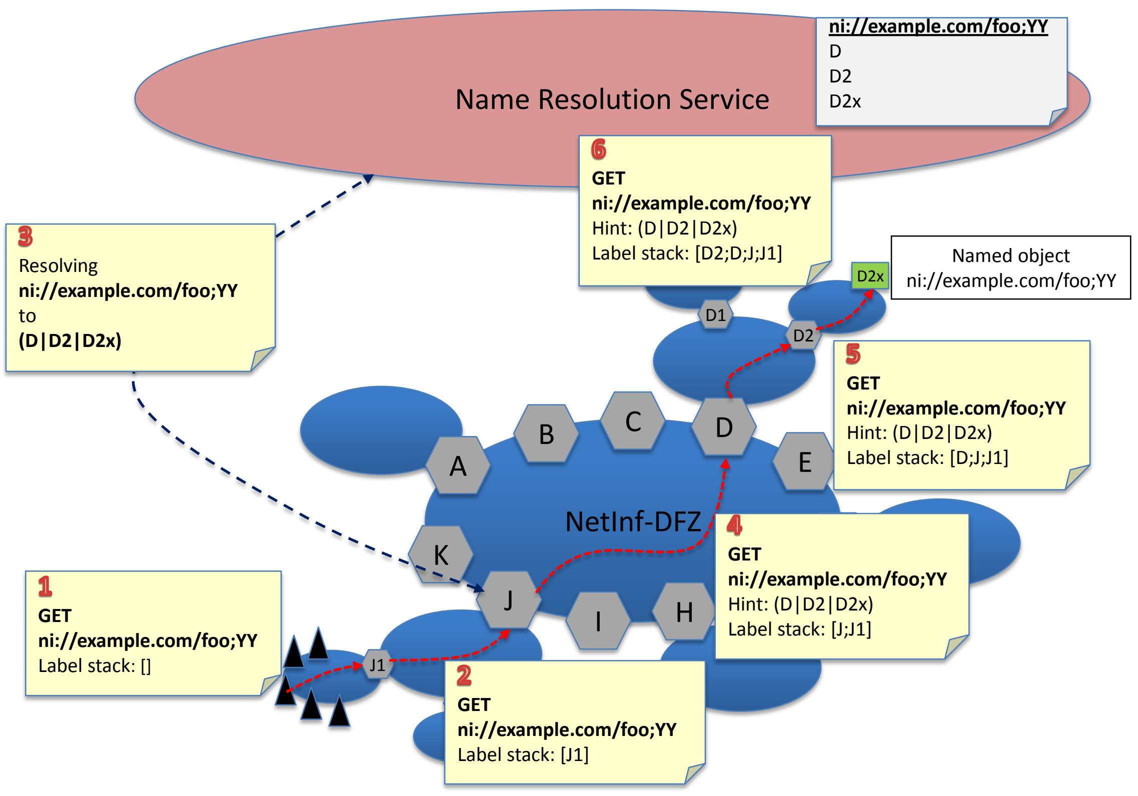


Figure 2.3: NetInf inter-domain scenario

At the global level, we expect only one routing scheme, akin to Border Gateway Protocol (BGP) for the Internet. Figure 2.3 shows an interconnection of different NetInf domains, connected via a central Default Free Zone (DFZ). As in today's Internet, edge domains can decide internally on NetInf routing/forwarding, adapted to a domain's needs. In the DFZ, the NetInf BGP routing system is used, a variant of today's BGP combined with a global resolution system for aggregation.

Figure 2.3 shows an example for inter-domain routing that relies on NetInf's hybrid, name-based/name-resolution-based object retrieval. The hexagons represent NetInf routers. The messages exhibit type and additional parameters (e.g., requested NDO name, label stack, routing hints). The example assumes that no cached copy is available at first. In step 1, a client in *J1*'s access network sends a *GET* request for the NDO `ni://example.com/foo;YY`. The request is forwarded to router *J* via a default route (step 2). While forwarding the request, the names of involved NetInf routers are added to the label stack in the request (here router *J1*) to construct a source route. Router *J* lacks routing information. Therefore, it consults an NRS, which resolves the NDO name into a set of routing hints ($D|D2|D2x$) (step 3), which are added to the request. The routing hints typically do not name the end-point destination node of the request but are rather used to forward to the next hop; node *D* in our example (step 4). Node *D* belongs to a different provider with its own routing/forwarding scheme, e.g., based on the provided routing hints (step 5). Finally, the request reaches node *D2x* that holds a copy of the requested NDO (step 6). Then, the information collected in the label stack is used to forward the NDO back to the requester (not shown in Figure 2.3).

We are designing and evaluating several name resolution and routing mechanisms that can be plugged in to the just described framework. Name resolution mechanisms, both for global (in the DFZ) and local deployments, are covered further below in Section 2.4, including the new DHT-based HSkip system. The GIN/REX system, which is described in Section 3.4, is employing the MDHT system for intradomain routing and resolution, and the REX system for global exchange of authority name to network ID mappings. The global routing with routing hints, summarised in Section 3.1.1.5, can as an alternative use DNS for resolving NDO aggregates into routing hints in a similar fashion as REX.

2.2 “Named Information” Naming Scheme

NetInf uses a common naming scheme that supports multiple “pluggable” cryptographic algorithms and representations. The “ni” and “nih” Uniform Resource Identifier (URI) schemes proposed by Scalable and Adaptive Internet Solutions (SAIL), have now been approved as a Proposed Standard by the Internet Engineering Task Force (IETF). At the time of writing, the specification [2] is in the Request For Comments (RFC) editor queue awaiting finalisation of normative references and will be issued as an RFC in due course (probably early in 2013 due to a dependency on the updated HTTP/1.1 specifications). The “ni” and “nih” URI schemes have already been registered as permanent URI schemes by Internet Assigned Numbers Authority (IANA)[1] and are therefore available for use immediately.

To briefly recap, “ni” URIs allow for the inclusion of hashes in URIs in a structured manner, so that name-data integrity can be verified. The “ni” URI scheme was designed for NetInf and its use as part of the NetInf ICN approach has been described previously. During the standardisation process, additional uses for these URIs were identified, amongst which were naming objects for the now-closed DECADE Working Group (WG) of the IETF and for naming keys and/or nodes for the CoAP protocol. That latter case also threw up a requirement for a compact binary form of name and also an unambiguously human-speakable equivalent to an “ni” URI, and so the “nih” scheme was derived from the “ni” scheme, essentially by removing all optional features and ensuing that the remaining structure was unambiguous when spoken. So for example, rather than use the base64url encoding of the hash an “nih” URI uses an ASCII-HEX encoding with optional spacing characters. Many other comments were also received, including from the World Wide Web Consortium (W3C)'s Technical Architecture Group (TAG) which indicates that the ideas behind the “ni” URI do recur in various contexts. This, and other expressions of interest are hopeful signs that the “ni” URI standard may be of real utility both for ICN and elsewhere. Figure 2.4 gives examples of the various forms of URI.

[1] `http://www.iana.org/assignments/uri-schemes.html`

```
+-------------------------------------------------------------------+
| URI:                                                              |
| ni:///sha-256;UyaQV-Ev4rdLoHyJJWCi11OHfrYv9E1aGQAlMO2X_-Q         |
+-------------------------------------------------------------------+
| .well-known URL (split over 2 lines):                             |
| http://example.com/.well-known/ni/sha256/                         |
| UyaQV-Ev4rdLoHyJJWCi11OHfrYv9E1aGQAlMO2X_-Q                       |
+-------------------------------------------------------------------+
| URL Segment:                                                      |
| sha-256;UyaQV-Ev4rdLoHyJJWCi11OHfrYv9E1aGQAlMO2X_-Q               |
+-------------------------------------------------------------------+
| Binary name (ASCII hex encoded) with 120-bit truncated hash value |
| which is Suite ID 0x03:                                           |
| 0353 2690 57e1 2fe2 b74b a07c 8925 60a2                           |
+-------------------------------------------------------------------+
| Human-speakable form of a name for this key (truncated to 120 bits|
| in length) with checkdigit:                                       |
| nih:sha-256-120;5326-9057-e12f-e2b7-4ba0-7c89-2560-a2;f           |
+-------------------------------------------------------------------+
| Human-speakable form of a name for this key (truncated to 32 bits |
| in length) with checkdigit and no "-" separators:                 |
| nih:sha-256-32;53269057;b                                         |
+-------------------------------------------------------------------+
| Human-speakable form using decimal presentation of the            |
| algorithm ID (sha-256-120) with checkdigit:                       |
| nih:3;532690-57e12f-e2b74b-a07c89-2560a2;f                        |
+-------------------------------------------------------------------+
```

Figure 2.4: Examples of "ni" and "nih" URIs

2.3 NetInf Protocol

NetInf employs one conceptual protocol providing *accessing named data objects* as a first-order principle. Thus there is one simple protocol that all nodes implement. The NetInf protocol is message-based; it provides requests and responses for `PUBLISH`ing, `GET`ting, and `SEARCH`ing for NDOs. These requests and responses employ the common naming format and the common object model.

This section briefly recaps the architectural decisions and protocol design that were described in SAIL Deliverable D.B.2 [5] and summarizes the developments that have taken place in the specification of the NetInf protocol since the writing of that deliverable.

2.3.1 NetInf Protocol Summary and Update

In the previous SAIL Deliverable D.B.2 [5] (Section 2.1) the NetInf architecture was characterized in abstract terms as follows

- NetInf enables access to 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 DTN networks.

This characterization remains valid and we have continued to work on the NetInf protocol described in Section 2.2 and Appendix 1 of SAIL Deliverable D.B.2. Since the publication of D.B.2, the specification of the ni URI scheme [2] has been approved for publication as a Proposed Standard RFC and, based on implementation experience, an extended version of the protocol document reproduced in Appendix 1 of D.B.2 containing a specification of the abstract protocol has been published as an Internet Draft (I-D) [10].

The CL concept has been refined and realized in a number of ways using Hypertext Transfer Protocol (HTTP) over Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and the DTN Bundle Protocol (BP) as transports for NetInf messages. To summarize, the abstract protocol and the CLs that have been developed are all based on the use of three pairs of messages and 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 Multipurpose Internet Mail Extension (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 combination of the NetInf protocol layer with choices of appropriate CLs offer a number of advantages over existing Content Delivery Network (CDN) approaches:

- transport of the content can be controlled from the receiver;
- applications can receive additional information (metadata) about the content;
- content can accessed across domain boundaries where different network technologies are in use (e.g., DTN and IP); and
- transport can be optimized for the network type and conditions that prevail (e.g., coping with intermittent connectivity and mobility).

The published specification [10] contains outline specifications for both the HTTP and the UDP based CLs. As envisaged in D.B.2, Section 2.2.2, the object model for the CLs makes extensive use of MIME and JavaScript Object Notation (JSON) has been used to encapsulate any affiliated data that is carried in CL messages.

The following sections 2.3.2 and 2.3.3 give an outline of the HTTP and UDP CLs.

2.3.2 HTTP CL

The HTTP CL implements the NetInf protocol layered over the HTTP protocol running over a TCP connection. HTTP is designed for use in a client-server paradigm so the CL implementation provides a unidirectional transport for NetInf messages from a client node that originates messages (GET, PUBLISH or SEARCH) that are sent to a server node that manages a cache of NDOs. The server processes the received message and sends appropriate responses back to the client over the same connection. If the node running the client also wishes to maintain a cache of NDOs that can be accessed via the NetInf protocol, it will also have to run an HTTP server that can accept NetInf messages from other client nodes. The situation for a symmetric arrangement of two nodes is shown in Figure 2.5. If appropriately implemented the server component can handle connections from an arbitrary number of other nodes in parallel, subject to system constraints.

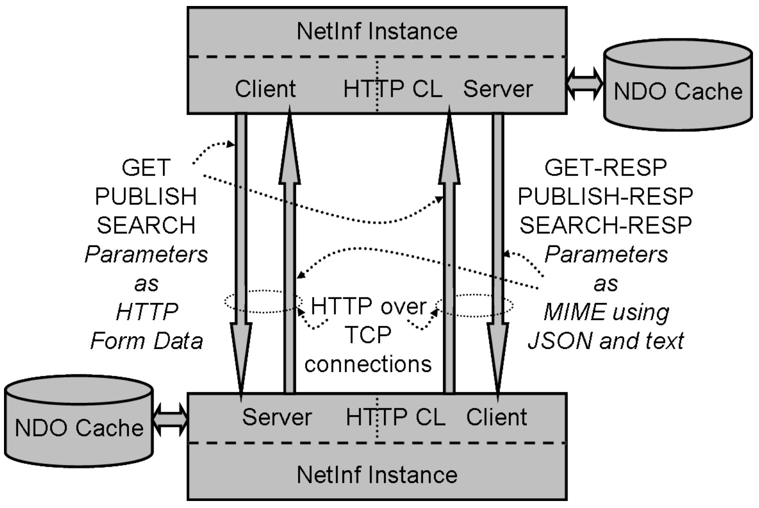


Figure 2.5: NetInf protocol HTTP convergence layer

The combination of HTTP over TCP used for the connections provides all the capabilities for a NetInf CL specified in Section 3 of the NetInf protocol specification [10] without the CL having to implement any additional facilities, since TCP provides reliable in-order delivery of messages and HTTP provides the necessary message boundaries.

The rest of this section provides an outline of the HTTP CL; for more details please refer to the full specification given in Section 6.1 of the protocol specification [10].

NetInf requests are encoded in HTTP forms and Multipart MIME HTTP response bodies are used for the corresponding NetInf responses. These choices have been made because they allow web browsers to interact easily with NetInf and because there are many tools available that make implementation relatively easy. However, the HTTP CL is also intended for use between NetInf 'infrastructure' nodes without human users or conventional web servers. Implementations have been constructed that use web browsers and forms as the front-end, integrated with a web server (Apache), and as stand-alone components for use in non-interactive applications.

Extensive use of JSON objects is made to provide relatively compact encoding of 'affiliated data' that is carried by the NetInf protocol messages. The term 'affiliated data' is used in the protocol specification to cover both 'metadata' that describes attributes of the NDOs being transmitted and metadata of the protocol. Any JSON objects carried consist of sets of unordered name/value pairs at the top level. As is normal with JSON, the values themselves may be arbitrarily complex nestings of objects or arrays of objects.

If the server is unable to successfully process a message for any reason, it returns an HTTP error response with a suitable error code. An HTTP 200 response is sent when processing is successful together with a response body that is described in the following sections.

2.3.2.1 GET operation

The form for the GET message has two mandatory fields:

URI the name for the NDO to be retrieved, typically an ni: scheme URI [2].

msgid the message IDentifier (ID) (must be unique per CL hop and request/response pair).

and one optional field:

ext Provided for future extensions and not currently used in our implementations.

The format for a GET-RESP message depends on whether the server is returning both the content octets of the NDO and its affiliated data, or just the affiliated data if the server's cache does not contain the content octets and the request has not been propagated further. In the second case the response body consists of a single MIME component of type application/json encoding the affiliated data. If both parts are returned, the response body is a two part multipart/mixed MIME type, consisting of the affiliated data as before and the NDO content using the appropriate MIME type for the content.

The affiliated data JSON object can contain the fields described below. Note that this remains a work-in-progress and is liable to change as the implementations mature and are further developed so the reader should expect a few minor inconsistencies between this description and the Internet-draft and the code. As one might expect, the code is at the bleeding edge.

NetInf String describing the version of NetInf protocol in use (e.g., "V0.2").

ni The ni: scheme URI for the NDO retrieved.

msgid Copy of the msgid from the GET message that resulted in the response.

ts A timestamp. There are no fixed semantics for this as it is mainly for debugging, currently this contains the time at which the response was generated in most cases.

loc A list of locator names from where the NDO might potentially be retrieved.

metadata A JSON object containing content metadata recorded with the NDO when it was published plus information about how it was published.

Some implementations also return additional JSON elements as listed below.

status A code, taken from the HTTP 2xx success response codes indicating what has been returned.

ct The MIME content type of the NDO content, if known.

searches An array of JSON objects describing searches that have flagged up this NDO as matching the search criteria.

2.3.2.2 PUBLISH operation

The form for the PUBLISH message has the same mandatory fields (*URI* and *msgid*) as the GET message (see Section 2.3.2.1) with some additional optional fields:

loc1/loc2 Locators where the NDO might potentially be retrieved.

fullPut Boolean value indicating if the publication operation includes the content of the file or just metadata.

octets If *fullPut* is true, this form field provides the content of the NDO and information about the type of the content.

rform Allows the user to choose a browser friendly Hypertext Markup Language (HTML)-encoded report on the publication or a JSON encoded report suitable for automated processing and/or sending through additional CL hops.

ext A JSON encoded object that can contain a *meta* field. The *meta* field is expected to contain a JSON object that is stored as metadata for the NDO content on the server whether or not the content itself is stored. In future, additional fields can be added to the *ext* object.

If the server receiving the PUBLISH message does not have this NDO stored already, and both policy and resources allow, the server will add the metadata and the content (if supplied) to its cache. Subsequent PUBLISH messages for the same NDO can be used to add the content if the first publication did not supply it or just to add additional or updated information to the affiliated data.

The affiliated data stored for an NDO records information about the NDO needed to fill in the JSON response object described in Section 2.3.2.1. This object is sent back as the response body if the user selects the JSON response format. Otherwise a web browser (and human) friendly report is sent back encoded as HTML.

2.3.2.3 SEARCH operation

The form for a SEARCH has two mandatory fields, a *msgid* field as used in the GET message plus the *tokens* field described here:

tokens A search query string appropriate for the search mechanism used,

and optional fields *rform* and *ext* as used in the PUBLISH form (see Section 2.3.2.2).

On receiving a SEARCH message, the server carries out the search using the selected mechanism and returns a response body, in the format selected by the user using the *rform* parameter, enumerating the ni: scheme names for the NDOs flagged by the search mechanism and additional information as appropriate. The search may be carried just among locally cached NDOs or extended to other nodes according to local policy at the server.

2.3.3 UDP CL

The UDP CL implements the NetInf protocol using a UDP datagram service, i.e., all NetInf messages are mapped to individual UDP messages. The purpose is to provide a light-weight datagram-based CL that can be used to implement NetInf transport protocols on top and that can provide efficient communication for querying NRSs, and request broadcasting/multicasting. The UDP CL provides no hop-by-hop flow control, retransmission and fragmentation/re-assembly.

The UDP CL has two sending modes: 1) send to specified destination IP address and 2) send to the well-known IPv4 multicast address 225.4.5.6. For both unicast and multicast the UDP port number is 2345. All request and response messages are JSON objects, i.e., unordered sets of name/value pairs.

For UDP CL messages, the following JSON names for name/value pairs are defined (not all objects have to be present in all messages):

version the NetInf UDP CL protocol version — currently "NetInfUDP/1.0".

msgType the message type (e.g., GET).

msgId the message ID (must be unique per CL hop and request/response pair).

locators an array of locators.

instance an UDP CL speaker identifier (must be unique per IP host, e.g., process ID and per process ID.

This version of the specification defines the GET request and the corresponding GET response only.

A GET request provides the following objects:

```
version:  "NetInfUDP/1.0"

  msgType:  "GET"

  uri:      name of the requested \ac{NDO}

  msgId:    message ID (see above)
```

A GET response provides the following objects:

```
version:  "NetInfUDP/1.0"

msgType:  "GET-RESP"

uri:      name of the requested \ac{NDO}

msgId:    message ID (see above)

locators: a list of locator strings
```

2.4 Name Resolution

Name resolution and name-based routing are essentially complementary alternatives in NetInf. Especially during migration, name resolution has several advantages. It does not require global adoption right from the start but can grow from the edges by first providing NRS capabilities in some edge networks that can make objects available inside the edge networks. NetInf name resolution supports today's Uniform Resource Locators (URLs), hence, any already existing object can be used in NetInf right away. URLs pointing to already existing copies can be registered as object locators in edge NRSs and objects can be retrieved via existing protocols like HTTP. NetInf nodes that route NetInf messages are not in fact required for a pure name resolution approach.

NDOs are advertised to an NRS using the `PUBLISH` message that contains the object name, a set of routing hints, and optionally some metadata. By supporting different kinds of routing hints like lower-layer locators (e.g., IP, URLs), indirection to other NetInf names, and other protocol-specific routing hints, this approach offers a flexible mechanism to adapt data retrieval to heterogeneous underlying networks. To retrieve the advertised information from an NRS, the `GET` message is used, or locators (also URIs) can be used according to the URI scheme present (e.g., "http:").

NetInf does not dictate a single global NRS but can support multiple NRSs which all support the same NetInf interface. Different types of NRSs are possible, e.g., a local broadcast NRS for local ad hoc networks, a global NRS, and local network NRSs. Operating an NRS in their local network allows network providers to improve traffic engineering by selecting appropriate object copies during the resolution process to reduce and control network traffic and, potentially, load at the caches and data servers. This gives network operators a strong incentive to invest in NetInf NRSs, thereby supporting the NetInf migration process.

In general, there are three operational modes for routing hint selection. First, the NRS can return all its known routing hints and the requester selects one or more routing hints to retrieve the data. Second, the NRS can return a pre-selected set of routing hints based on its network knowledge, resulting inter-domain traffic, server load, internal policies, etc., giving the NRS full control of the selection process. In a hybrid approach, the NRS can return a list of routing hints. This leaves the final selection to the user while still making use of the internal knowledge of the NRS.

For a fully functional, world-wide Network of Information, at least one global NRS is required. NetInf offers multiple options to interconnect separate local NRSs into a global NRS infrastructure, including the Multilevel DHT (MDHT) system [11] and the Hierarchical SkipNet (HSkip) system [12]. Both systems form a global, hierarchical NRS that is topologically embedded in the underlying network to improve scalability, latency, and locator selection for efficient information dissemination. The hierarchical, topological embedding of both systems combined with their registration and retrieval scheme ensures that close-by locators are automatically preferred over farther away locators during name resolution. To reduce load at the global level, caching of NRS entries can be used at lower-level NRS nodes. Details can be found in reference [12].

The NRS can also collect statistics about object popularity. The NRS is well suited for that as it aggregates resolution requests by many users. In a topologically embedded NRS, dedicated NRS subsystems are responsible for separate subnetworks. Thereby, the statistics collected by a local NRS automatically represent local popularity, making caching even more efficient when combined with *local* caches.

We have described MDHT in previous deliverables. In this deliverable, in Section 2.4.1 we first describe the HSkip NRS approach in more detail before we present an aggregation optimisation for NRSs for NetInf names in Section 2.4.2.

2.4.1 A Hierarchical Extension to SkipNet: HSkip

The goal of name resolution in an ICN architecture is to map a *name* of an object to a *locator*. A locator is some sort of identifier that can be passed to some other (typically, non-ICN) network layer and serve as an address in this network; the typical example would be an IP address. When the object names are non-hierarchical, a common approach is to use Distributed Hash Tables (DHTs).

But despite the names being non-hierarchical (for good reasons), two other aspects are likely to remain hierarchical to some degree: (1) the structure of networks as imposed by the commercial structure of network providers, (2) usage patterns. Point (1) is justified by the still existing structure of smaller local providers and larger interconnection providers (although there is some flattening of the Internet going on). Point (2) is justified by human behavior, e.g., people are more likely to be interested in local affairs, with some scoping taking place based on language, cultural commonalities, municipalities and nations, etc. For example, an analysis of the Domain Name System (DNS) traffic [13] at the University of Paderborn has shown that, depending on the observed network, between 40% and 70% of the total DNS queries have requested locally registered domains (i.e., uni-paderborn.de and related domains). A study by McDaniel and Jamin [14] has shown similar results. There is also some very rough correlation between these two effects in that network providers still somewhat relate to such cultural boundaries.

Supporting a flat name space on such a hierarchical network with hierarchical usage patterns by a standard DHT is certainly possible; it does, however, squander optimization options. The main intuition is to structure the DHT such that name-locator mappings for objects that are more likely to be requested in some parts of the network are made available in these very parts; the goals are to reduce lookup latency and lookup traffic.

To this end, we have developed several approaches to structure DHT systems such that the network hierarchy and usage patterns are taken into account. In previous work, we have developed a multi-level approach — Multilevel DHT (MDHT) — where each level corresponds to a network level and allows administrative independence. In this section, we extend the SkipList-based SkipNet system into a hierarchical version: HSkip.[2] Evaluation results are described in the evaluation chapter, Section 4.1.1.

We wish to emphasize, however, that we are still working on a flat-ish namespace. For the example of the ni naming convention, this means that the hash itself serves as the "flat name" in, e.g., HSkip; hierarchies come from the network topology, not the object names.

2.4.1.1 Requirements for hierarchical name resolution

Apart from usual requirements — low latency, scalability, deployability, agility when adding/remove objects, control scoping of name bindings — we would like to emphasize one extra requirement here: *resolution path locality*. If a local object copy is available for retrieval, resolution for this object ID should happen locally (i.e., within the same subnetwork) (called *resolution locality*) and the resolver should return the local object locator (called *content locality*). If the two communication endpoints (i.e., sender and receiver) are in the same network domain, the *resolution path* should also be contained within that network (called *resolution path locality*). The same is obviously also desirable for the data path, which is, however, potentially not under the control of the NRS.

2.4.1.2 Background: SkipNet

SkipNet is a DHT architecture based on the idea of Skip Lists. In SkipNet, each object is named with a *string ID* and each node is named with a similar string ID and, in addition, with a *flat numeric ID* chosen randomly from a uniform distribution. The string IDs can be hierarchical. SkipNet can do routing/forwarding in both namespaces with an average latency of $O(\log n)$ (n number of nodes). When forwarding within the string namespace, SkipNet supports the two properties *resolution path locality* and *binding record locality*. Binding record locality makes it possible to define the (subset of) node(s) on which an object should be stored.

String forwarding: SkipNet constructs a double-linked ring of all nodes, sorted by string ID. Each node has $2 \cdot \log N$ (N = no. of nodes) pointers that leap over an exponentially increasing number of nodes, somewhat similar to Chord's [15] finger table. When forwarding to a certain node ID, SkipNet follows these pointers and leaps over as many nodes as possible without surpassing the destination ID until the destination is reached.

[2]The remainder of this section is text from a submission to the Elsevier journal of Computer Communications.

Numeric forwarding: SkipNet constructs multiple double-linked rings by dividing the main ring into two rings, which are each subsequently divided again, and so on. The binary, numeric node ID determines in which ring a node participates. Forwarding happens by finding the next node in the main ring that shares the same first numeric digit with the requested numeric ID. SkipNet then changes into the next ring containing this node and repeats the same process with the next digit until no more progress can be made. Then, the node with the closest numeric ID is chosen.

Constrained load balancing: SkipNet uses a combination of string forwarding and numeric forwarding to achieve a load balancing mechanism constrained to a subset of nodes (called Constrained Load Balancing (CLB)). To achieve this, SkipNet divides each object ID into two parts. First, the *subgroup part* that specifies the subset of nodes, i.e., all nodes with this common string ID prefix. Second, the *object part* that serves as unique object ID within this subgroup. The *hash(object part)* is used to determine the specific node within the subgroup of nodes that is responsible for this object ID. CLB forwarding is performed in two steps. First, finding any node that is part of the subgroup via string forwarding as described above. Second, finding the responsible node within this subgroup by numeric forwarding. Here, the above described numeric forwarding mechanism is slightly modified to make sure that the subgroup of nodes is not exited during forwarding. This is done by reversing the forwarding direction when the boundary of the subgroup (as specified by the common string ID prefix) is reached.

One might think that the original SkipNet architecture can directly be used to provide a hierarchical NRS for flat namespaces. First, SkipNet provides some kind of hierarchy (hierarchy of rings in the *numeric namespace*). However, the numeric IDs cannot be used to construct a topologically-embedded hierarchy as needed by our NRS as the numeric SkipNet IDs have to be uniformly distributed. Second, SkipNet provides resolution path and binding record locality only for the *string IDs*. However, because of the required topological embedding in the underlying hierarchical network topology, the string IDs have to be hierarchical, contradicting our requirement of a flat namespace. Moreover, SkipNet generally does not provide any means to discover the nearest object copy. Hence, we have to find a way to provide resolution path locality and binding record locality for a flat namespace, combined with a topologically-embedded hierarchy to find the nearest object copy. For this reason, we have developed the Hierarchical SkipNet (HSkip).

2.4.1.3 HSkip architecture

HSkip is a *homogeneous* hierarchical DHT based on SkipNet. HSkip differs in two main aspects from SkipNet. First, we construct a nested hierarchy of resolution domains, embedded in the underlying network topology. Second, we provide binding record and resolution path locality while naming objects with the *numeric IDs*. This is essential to perform our hierarchical registration/request scheme. In addition, thanks to naming objects with *numeric* IDs, transparent remapping of objects to a new resolution domain is possible, which is not possible in SkipNet as the domain is encoded in the object's string ID.

Hierarchy of Resolution Domains: Our hierarchical design is based on a single SkipNet DHT. SkipNet makes it possible to use a hierarchical string namespace for naming SkipNet nodes. We make use of this to construct our hierarchy of nested, topologically-embedded domains. Each HSkip node is named using a string ID chosen from a hierarchical namespace corresponding to the underlying network topology. For example, DNS-like names (e.g., *country.provider.organization.node*) can be used. All nodes with a common name prefix are grouped together into one domain. For example, all nodes with the name prefix *countryX* are part of the domain *countryX*. All nodes with the name prefix *countryX.providerJ* are part of the domain *countryX.providerJ*. By allocating names in this way, the nested hierarchy is constructed automatically when nodes join the HSkip network, using the standard SkipNet join process [16]. The numeric node IDs are chosen randomly from a uniform distribution without special constraints, similar to the original SkipNet design.

Figure 2.6 shows an example HSkip resolution domain hierarchy. The leafs of the tree (circles) represent physical nodes. The complete string ID of a node is given by following the path from the root of the tree to the node, i.e., the left-most node has the ID *x.j.a.1*. All other tree nodes (squares) represent higher-level resolution domains. Each resolution domain consists of all nodes in their subtree. For example, the resolution domain *x.j* contains the nodes *x.j.a.1*, *x.j.b.1*, and *x.j.b.2*. Note that the HSkip nodes are not explicitly connected in this hierarchical way. Rather, the hierarchy is only given implicitly by the nodes' string IDs which are used by our forwarding protocol to traverse the HSkip network in this hierarchical way as described subsequently. It is important to note that this hierarchical structure imposed by the string IDs has no correspondence to the tree-like SkipNet levels created via the numeric IDs.

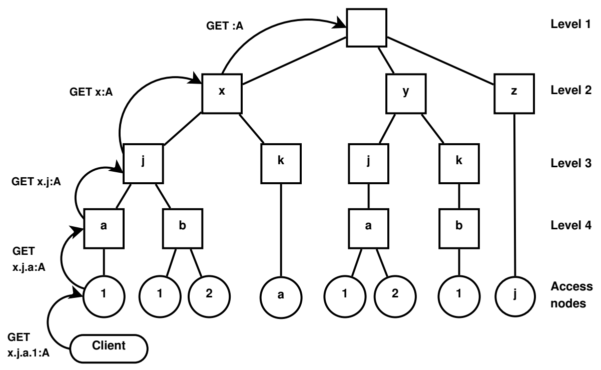


Figure 2.6: HSkip hierarchy with GET request for object *A*

Locality Properties with Numeric IDs: The original SkipNet design supports binding record locality and resolution path locality. A major SkipNet drawback for us is that it only supports those locality properties when objects are named with the string IDs. SkipNet achieves those properties via CLB. The name of the resolution node/domain where the entry should be stored is attached to the object name. In our case, where the underlying network topology is hierarchical and, consequently, the resolution domains are constructed and named in a hierarchical fashion, this approach would result in hierarchical object names. However, we require an NRS for a flat namespace. Therefore, HSkip names objects with the flat numeric namespace. To preserve binding record locality and resolution path locality for the numeric namespace, we introduce our own intra- and inter-domain forwarding protocol.

Intra- & Inter-Domain Routing/Forwarding: Our intra-domain forwarding protocol is based on SkipNet's CLB mechanism. The routing tables are constructed like in the original SkipNet design. Intra-domain forwarding at different hierarchical levels can be done with the same set of routing tables, i.e., HSkip requires only a single routing table per node. Therefore, we characterize HSkip as *homogeneous* hierarchical DHT.

As shown in Figure 2.6, a request for object ID *A* is started by first querying the local Access Node (AN) *x.j.a.1*. Note that Figure 2.6 shows the alphanumerical string ID of the HSkip *node* *x.j.a.1* (which is hierarchical as described previously) separated by colon from the *flat* object name *A*. The hierarchical node ID is not part of the flat numerical object ID and is only used to perform SkipNet's CLB forwarding as described below.

If resolution fails at the local AN, resolution continues at the next level *x.j.a*. If resolution also fails here, the node responsible for this ID at level *x.j.a* forwards the request to the next higher level *x.j* until resolution succeeds or fails at the top level. At each level, resolution is constrained to the specific resolution domain via SkipNet's CLB forwarding. For example, in the resolution domain *x.j.a*, resolution is restricted to nodes with the prefix *x.j.a*. Levels can be bypassed if required, e.g., due to problems at a certain level, by continuing directly with a smaller prefix.

As explained in Section 2.4.1.2, the CLB mechanism consists of two steps in SkipNet: (1) forwarding the request to a node that is part of the constrained domain (*string forwarding*), and (2) forwarding to the responsible node within the CLB domain (*numeric forwarding*). Thanks to the way that a request is started and forwarded through the hierarchical resolution domains in HSkip, the request path by definition starts at a node that is part of the current restricted CLB domain. Hence, string forwarding of the CLB mechanism can be omitted. Only the numeric forwarding has to be carried out.

SkipNet's numeric forwarding requires $O(\log n)$ steps at each level. HSkip's inter-domain forwarding works similar to the *entangled* MDHT forwarding, i.e., the forwarding steps performed at level i bring the request closer to the responsible node at level $i-1$. Hence, even assuming the worst case that all levels have to be consulted, HSkip only requires $O(\log n)$ resolution steps in total.

2.4.2 Peering with Flat Name Spaces

There are situations where the hierarchical name resolution can be and should be optimized. A *peering relation* between network domains (as explained in Section 2.5.3) needs a scalable solution in addition to the hierarchical resolution structure that follows certain resolution path. This is because domains that are peering at the content level would like to share their cached content or parts of it depending on peering agreements to their peering partners only. The locally cached content does not follow any previously set structure but appears flat as opposed to globally published content. Thus we need a mechanism to efficiently check if any peering partners may have a copy of a requested content in their cache(s) before using the "upstream" providers in the resolution path. To facilitate flat content distribution among the peering partners and related distributed name resolution needed for content peering Bloom Filters (BFs) is introduced as a vehicle for aggregation to achieve support for horizontal agreements and related name resolution step between (name resolution) domains. The BFs are used to announce only the content that a peering partner makes available in its caches. BFs are not used to announce all NDO in a global NetInf network (assumed to be order of 10^{15}), but only those NDOs that are cached by a peering partner. Therefore the upper limit of entries announced by a BF is limited to the size of largest cache used for peering. Note carefully, that a peer may announce more than one BF to its peers, e.g., one BF per cache or even multiple BFs for a cache.

BF is a convenient probabilistic data structure to represent group membership within a given set of objects. The size of a BF, the number of entries and hash functions define probability of false positives:

$$P_{err} = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - \mathrm{e}^{-\frac{kn}{m}})^k \tag{2.1}$$

, where k is the number of hash functions, m is size of BF vector and n is the number of inserted elements.

BFs have been also used for content delivery purposes, for instance, Publish-Subscribe Internetworking Routing Paradigm (PSIRP) used it for packet multicast forwarding [17], but here we use BFs for their original purpose to test group membership.

BF structure between resolution points Each NRS domain wishing to advertise cached publications for content peering purposes as an aggregate to its peers creates one or more BFs. The association between the NRSs to exchange aggregate announcements is created as part of a (content) peering agreement. The announcements can be implemented by reusing NetInf registration messages or a routing protocol (e.g., BGP) to distribute the advertisements. The frequency at which the announcements are exchanged should be a configurable parameter and the announcements must be rate limited. Based on BGP announcements we recommend an interval of not less than 10 seconds between two consecutive advertisements or BF NetInf registrations. During this time interval the advertiser aggregates all new registrations within its local domain into a single or multiple BFs.

A BF is an aggregate of a number of resolvable flat names/IDs such as ni-names that are made accessible to name resolution requests from peering partners. The aggregate does not contain any other publishing info. Each NRS can advertise multiple BFs, where BFs could have additional properties such as the name/ID space type or scope they represent, the publishing authority, etc. The BF advertisement includes an identifier (locator) of the announcing NRS so that an external NRS can contact it to resolve the locator for the content.

When a NRS receives a request, e.g., GET, it first checks whether it has a local match for the requested content. If not, then it checks if some of its peers may contain a copy of the content by testing if the ni-name matches any of the BFs from the peering partners. Note that there could be more than one matching BF. If one or multiple matches are found the NRS contacts those NRSs that had announced the BFs to complete the name resolution for the original request. Based on the responses from the matching NRSs, the original NRS has adequate information to select a best candidate location to be returned to the content requestor. For the selection of best candidate location, there may be multiple criteria that specify how a best candidate location could be elected, e.g., load of the hosting node, popularity of data, distance between the client and a data in the routing space and so on. Of course, if there is only a single copy of the published data, then this selection process is trivial.

Once the best candidate location has been selected, the NRS has several options depending on what kind of delivery model we have in use. The first option corresponds to 'query mode' where the NRS sends a reply back to the client in order to instruct it on how and where to fetch the requested data. This is similar how conventional NRSs operate in the Internet. For a second option, the NRS acts as a proxy and fetches the requested data on behalf of the client and sends it back to the client. This is similar functionality to the operation of web proxies. A third option is closer to name-based routing where the NRS just sends the original request towards the next hop node, which then continues resolving/routing the request.

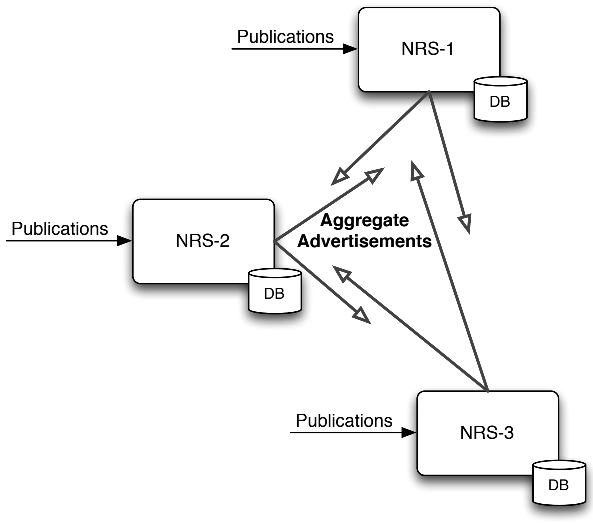


Figure 2.7: An example of BF based aggregation advertisements between resolution points

Typically BFs are created with a default size for which the preferred false positive probability is estimated. There are two cases when BF's capacity needs to be increased; 1) a NRS receives a non-matching resolution request for its BF, i.e., because of a false positive match where a peer requests a NDO that was not cached, and 2) capacity of BF needs to be increased to maintain the desired false positive probability ratio, e.g., more entries in a peering cache or cache size is extended. In both cases, there could be specific algorithms used to derive a new BF size based on Equation (2.1) relating the BF vector size, number of entries added and number of used functions used. However, it should be noted that a BF independently of its utilization rate requires the same amount of space, i.e., both empty and full BF that are dimensioned in the same way require the same amount of memory. One key property of BF is 'false positivity' meaning that BFs with too high utilization rate should be avoided. Therefore, it is important that these aspects are captured by the policies governing how BFs are constructed, dimensioned and resized.

The NRS advertising incorrectly configured BFs will either get unnecessary requests that it cannot resolve or it may miss an opportunity to service a request even if it had a local copy of the content. Therefore it is in the interests of the creator of a BF to offer a minimal probability of false positives by adjusting the BF parameters (m, n and k in Equation (2.1)) or dividing the cached content ni-name space between multiple shorter BFs.

2.5 Migration

For migration to be effective, both technical and business aspects should be taken into account. In SAIL deliverable D.B.2 [5] we defined Architecture Invariants, enabling NetInf migration through:

- Compatibility with current infrastructure;
- Compatibility with existing protocols and networks;
- Compatibility with existing content.

The NetInf architecture supports multi-domain environments where each domain deploys its own network technology including domains of challenged networks, such as Delay-Tolerant Networking (DTN) networks. NetInf interconnects a variety of networks with different address spaces, different network technologies, and different owners through NetInf routers at the boundary of a (technology) domain. The Convergence Layer (CL) adapts the NetInf protocol to different types of underlying networks on a per hop basis.CL forms the main concept for compatibility to existent infrastructure. The NetInf naming scheme follows URI scheme and moreover provides a means to embed hash based names into URLs (Section 2.2). Therefore NDOs are accessible not only via NetInf protocols, but also through legacy protocols as well other ICN protocols.

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 (see D.B.2 [5] Section 2.2.8 for C-language binding). NetInf API allows for both content and application compatibility.

There is a huge existing infrastructure dedicated to name resolution in the current Internet; the DNS system. It is important to take advantage of the current DNS infrastructure at the early stage of a migration, a DNS server may point a request to the nearest NRS node similarly how CDNs redirect name resolutions to surrogate nodes.

The rest of this section is organized as follows: first the technical tools offered by NetInf are described to give the technical basis, followed by a study on how migration based on these tools could actually take place following incentives for unilateral adoption that then grows towards multilateral adoption enlarging the circle of impacted parties. Content caching opens a possibility to establish a new type of peering relationship, namely peering at the content level. We also discover that for transit operators content caching is not aligned with the current business incentives that are based on pure volume of transferred bytes. Therefore, new content centric business models need to be developed for Tier-1 providers.

Migration to a new network architecture or even introduction of a new protocol into an operative real life environment has multiple obstacles to overcome. The benefits of the new approach need to outweigh the associated operational and investment costs to provide positive net value at each step of the migration. More importantly the distribution of benefits and costs need to match the impacted actors. Incremental deployability enables reasonable initial investments and lower risks but easily leads into interworking arrangements with their own complexity and scalability issues that may stall the subsequent steps. Migration challenges can be scoped by considering the number of impacted actors starting from a case where one actor decides to deploy a technology unilaterally and then extend the considerations to a multiple actor case where multiple parties need to cooperate. Of course, the scale of multilateral arrangements can vary from a pair of actors to the complete value chain at the global scale. The adopted CL approach of NetInf ensures reusability of the existing infrastructure and complies to the Architecture Invariants of D.B.2 [5]. In the next section we will first look into the technical tools offered by the NetInf architecture and then in the subsequent sections discuss how these tools would be taken into use and what kind of incentives will guide their adoption.

2.5.1 Technical Tools for NetInf Adoption

An instantiation of a NetInf architecture requires a name resolution system and content stores. Note that content storage can also be in the hosts. The NRS node redirects ni-request to content stores that serve requests. Clearly, a DNS server may also provide locators for NDOs in small instantiations, but DNS can not meet the need of massive amounts of name to locator bindings and subsequent locator updates in any larger set ups, nor it can not provide the ni-name integrity check at the content publishing stage.

The NetInf protocol (Section 2.3) can be layered on top HTTP or any other message passing protocol by the use of the CL. Since the ni-scheme is URI-based, the use of HTTP is straight forward. Neither the client software nor the original HTML-pages do not need to be updated to begin with. It is enough that the client can follow redirections issued by the DNS pointing, maybe through an Service (SRV)-record to an authoritative NRS node. Unmodified clients without NetInf support will suffer from the burden of additional redirections and related transaction delays. They cannot take any benefits from the name content bindings to ensure consistency of the content and accessing same content from multiple sources.

The next step in adding NetInf support into client software and as well as to HTML-pages (e.g. the server side) would be to apply the mapping between ni-URIs scheme and URLs ([18] or Hypertext Transfer Protocol Secure (HTTPS) [19]) by making use of the .well-known URI [20] by defining an ni-suffix as provided in ni-scheme specification (see Section 2.2). Such “a mapped URI” would look like `http://<netloc>/.well-known/<hash-alg-name>/<digest>`. Caches can respond to HTTP GET ni-requests with content .well-known URLs. Further the NRS can redirect the request to off-path caches that it knows containing an associated copy of the content. In the longer term browsers ought to be able to use the NetInf protocol directly. This would allow the browsers to take advantage of alternative sources for the content without having to do anything extra and being able to validate the name content binding.

Use of ni-scheme and NetInf caching impacts also search engines. Integrating NDO content identified by the ni-URI scheme into the indices created by web search engines is likely to need extensions to the search strategy used by the associated “crawlers" together with possible alterations to the way they would access this content for indexing. As crawlers prefer visibility of the content type in order to classify it properly and control their search strategy, supporting information in addition to basic ni-URI is needed. This additional information for content classification can be made available in various ways, including

- links containing an ni-URI can include the optional query string “`?ct=<content-type>`”;
- references within other documents could encode this data in a standardized way;

- using a crawler enhanced to use the NetInf protocol to access the content metadata via GET;
- providing a specific extension to the NetInf protocol GET that returns just the content type and some summary material.

Crawlers usually access the content in order to be able to display some title or summary information related to the content and this information is also used as input to the page ranking algorithms used by search algorithms: ni-URI links without available summary information would almost certainly be relegated to a very lowly position in the page ranking. Accordingly the crawler will normally need the support of an NRS to provide locators(s) and/or routing hints for a cache containing the NDO when accessing the actual content or metadata.

During the initial deployment of the NetInf scheme, crawlers could make use of the ".well-known" mapping of ni-scheme URIs to HTTP scheme URLs if they need to access the actual content or its metadata so that it will not necessarily require immediate integration of NetInf into crawlers.

To ensure that NDOs appear reliably in search results, it would be desirable to get locators of NetInf NRSs into the seed search tree of search engine crawlers. The NRS should be capable of providing the crawler with a listing of the NDOs that it knows about (e.g., via HTML pages); organizing this so that the crawler can obtain incremental updates would lead to increased efficiency. With the deployment of NetInf it may be possible to use the SEARCH capability of NetInf to feed the crawler Section 2.3. One could also envisage that metadata could be used both to provide improved indexing, via keywords to facilitate page ranking and other content indexing, and using 'related documents' fields to provide additional search paths.

2.5.2 Unilateral Adoption of NetInf

The most usual and most successful form of migration is unilateral adoption of a network technology as an application level overlay. For example, this was how BitTorrent [21] became widespread. Application overlay needs a client side application support and a set of accessible Web servers and no investment into the network infrastructure. An overlay doesn't require involvement from any other party than from the interested end users to install the needed client software and from the application provider to set up the servers. Naturally the application provider wants to maximize the customer base. For this purpose the distribution of the client side software should be efficient and automated without any manual configuration. The Web application developers have addressed this through downloadable browser plug-ins, e.g. Adobe Flash, Quick Time etc. The first generation NetInf implementation relied also on a browser plug-in [22] approach. OpenNetInf plug-in acts on ni-names in a web-page and provides client side support for the name resolution as well as implements the (early version) of the NetInf protocol. The plug-in approach has the drawback that for each browser platform an own variant may be needed. This, however, is changing with the increased capabilities of Web browsers (more specifically through HTML 5) and wide adoption of JavaScript [23]. This allows application developers to provide code that is downloaded to the end users when the service is accessed first time. Application segments with special content integrity and/or confidentiality requirements for content delivery can benefit from the NetInf naming scheme where the name is bound to the content itself and the server providing either the cached copy or origin NDO is under control of the NetInf name resolution system.

Application overlays have a set of known drawbacks. They introduce (processing and bandwidth) overhead and create performance limitations. The overall protocol design gets easily complex if the application level overlay needs to get access to network topology and other network resident state information to optimize its performance. When an overlay needs assistance from the infrastructure it become tied with the infrastructure that slows down the deployment. More importantly the overlay network may contradict traffic delivery policies of the underlay operator creating a negative economical incentive to the underlay to support the overlay. In such not so rare cases, the underlay operator is likely to throttle or even filter out traffic of an application overlay especially if the amount of carried traffic becomes significant. Application level overlay offers a quick starting point to experiment, tailor and innovate with new name bindings, caching and proprietary content routing schemes in a given application context (such as fleet management, etc.). However, it offers limited performance beyond the original target segment. Even worse, it usually results into uneven distribution of benefits that is likely to become a burden both for the application and network providers in cases the traffic amounts and mixes grow. Support for increasing amounts of traffic require new investments to bandwidth and transport capacity. Without a positive net value these investments will not happen and all parties will suffer.

Another form of unilateral migration is a case where an operator starts to deploy NetInf technology inside his or her own network domain transparently to improve the efficient and timely use of network resources. The motivations are very similar to deployment of transparent caches, or Network Address Translation (NAT)-functionality that do not need any actions from the users or from content providers. Currently operators are leaning towards two approaches to match the capacity demand with the cost of delivery: transparent caching and CDNs. Caching is a local optimization to improve Quality of Experience (QoE) and to lower transit costs by bringing the content closer to the edge of the network. It does not create any business relationship between the access network operator and the content provider. Beyond the intra-domain traffic and cost optimizations operators are developing their own CDNs to monetize the Over-the-Tops (OTTs) traffic and to get a role in the content delivery value chain. CDNs are meant for primary content for which the end users are willing to pay for and there is a service contract between the CDN operator and the content provider. CDNs may use caches but more importantly they use proprietary request routing to locate the appropriate source of content delivery. NetInf can combine caching and Telco CDNs into a one single solution that an operator could package into a service. Here NRS mediates between the local caching system and the CDN. NRS participates to CDN request routing and coverts its redirection requests into NetInf messages.

The key concept of NetInf is the content based naming scheme that decouples naming from content location. Instead the names are cryptographically bound to the content itself. This makes content replication and management significantly more scalable and efficient: any node with a cached copy can provide the corresponding content. Content from external sources can be renamed with content based name when entering into the operator's domain, republished in the local NRS or DNS and replicated in the local caches to meet operators policies and potential Service Level Agreements (SLAs) requirements. To implement this NEC NetInf Router Platform (NNRP) Section 3.1.1 or NRS Section 3.1.2 would need to intercept the original HTTP-requests a replies and to translate them into NetInf addresses and actions. For that purpose NNRP has an "input_http" module, which translates HTTP-messages into a NetInf message. Similar renaming was implemented for NRS with a functionality to position content in selected caches.

This is how the content can be acquired unilaterally into a local NetInf deployment. The republishing phase takes into account content renaming, traffic engineering, network conditions, subscription profiles, provider SLA and other service level attributes when selecting the caches to store the content. When subsequent requests for the same content arrive a simple name look-up from the NRSs can check if the content is already within the local scope after which the request is redirected either by DNS or HTTP redirect to the proper cache that will apply all the relevant service attributes to the content delivery. Here the NetInf protocol is used between the NRS and NNRP elements and caches, the clients would not need to change their functionality.

NRSs offers a control point at the request routing layer with a fine-grained control how to allocate storage in servers and how to use transport capacity within the network topology. Allocation of network storage (caches, web proxies and surrogates/content delivery nodes), dimensioning of the transport capacity, and dynamic load balancing are all related and taken into account at the local publishing phase. Without a combined view of these functions an OTTs request routing decision can be in conflict with the transport policies. Optimal transport level load balancing needs visibility into content attributes: type of the content (static, streaming, live, coding, etc.) when determining the scheduling of transport capacity between contending requests.

2.5.3 Multilateral Adoption of NetInf

Since the Internet consists of interconnected, but independently owned and operated *autonomous systems*, it is evident that common agreement by the network owners is needed for any updates affecting all the network domains. There are two principal options to overcome this hurdle: Firstly, make sure that the changes are obviously beneficial to all network domains, or, secondly, re-engineer the design so that only the parties benefiting from the changes need to participate. [3]

The commercial and voluntary structure of the Internet stresses the importance of operational incentives, the motivations each independent domain has for network interconnection [25, 26, 27]. The most basic motivation comes from customers paying for the network connectivity they want. Such paid connectivity to third party networks is called transit service. More subtle incentives based on mutual benefit drive bilateral interconnection (*peering*), both between backbone networks, and between any other networks [28]. While the backbone networks have to peer in order to provide the universal transit service, the typical incentive for peering among other networks comes from the desire to limit their transit service charges, which are typically volume-based [29, 30]. The inter-domain structure of the Internet that arises from the interconnection incentives — networks paying service provider networks for transit service, the provider networks peering to be able to provide the universal transit service, and the mutual peering on all levels to limit the transit costs.

To understand the impact of NetInf to the Internet we need to look into how Internet is consisted and what are boundary constraints that can not be changed by any technology since they are stemming from the purpose or mission of the Internet. The Internet topology divides into two kinds of well-defined regions. Each Autonomous Systems (ASes) governs its own network using suitable *intra-domain* routing protocols and management systems. Contrary to this relative freedom within the domains, *inter-domain* routes are exchanged with the Border Gateway Protocol (BGP [31]).

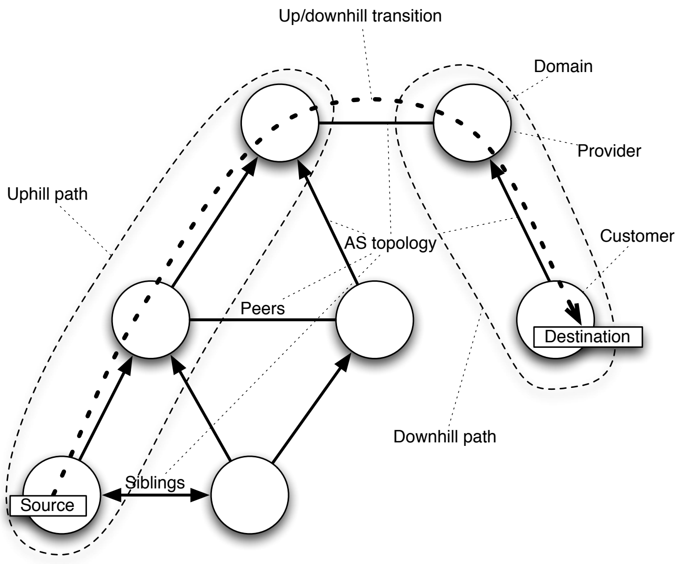


Figure 2.8: A tiered Internet model

[3]This section is based on Jarno Rajahalme's doctoral dissertation [24] and papers therein.

The bilateral contractual relationships between adjacent BGP speakers define the global Internet topology. This structure leads to a tiered Internet model (Figure 2.8), where the ASes at the topmost tier (the *Tier-1*) form, in practice, an oligopoly that does not buy transit from anyone else; all are peering with each other. Bold lines in the figure represent inter-domain links between domain border routers. Solid arrows represent customer — provider relationships, solid lines peering relationships, and two-headed arrows represent sibling relationships between autonomous domains. While Tier-1 networks typically cover large geographical areas, national or regional Internet Service Providers (ISPs) often have a denser regional network footprint, and thus have a natural role in offering regional connectivity services and reselling universal transit service towards those parts of the Internet they have no peering or customer relationships with. These regional transit ISPs are called *Tier-2* networks. Finally, *Tier-3* domains consist of smaller ISPs reselling transit service to customers not running BGP and of the network service users running BGP. The vast majority, about 90%, of the ASes are in the Tier-3 class [32].

In [26], Gao derives an empirical *valley-free* inter-domain path model stemming from the bilateral relationships between ISPs, as deducible from the BGP routing data. In this model, all Internet paths can be divided into three segments (Figure 2.8), *Uphill path* is the sequence of domains from the source domain up to the first provider-to-customer or peer-to-peer link. All the links in this segment are either customer-to-provider, or sibling-to-sibling links. *A peering link in the middle* is a link between two Tier-1 providers, or any peering link between ISPs. *Downhill path* is the sequence of domains from the first provider-to-customer link to the destination domain. All the links in this segment are either provider-to-customer, or sibling-to-sibling links.The valley-free property follows from the ISP incentives reflected in their inter-domain routing policies. Since practically all valid inter-domain paths are valley-free [26], path selection can be safely restricted to valley-free paths.

Policy compliant inter-domain path selection can be performed by selecting a path according to the locally beneficial *path selection incentives* as listed below in order of decreasing preference [24]:

1. A customer path (*for additional revenue*);
2. a path to a sibling's customer (*revenue for the sibling*);
3. a direct peering path (*revenue neutral*);
4. a path via a sibling's peering link (*cheaper than transit*);
5. a direct provider path, and finally, if nothing else is available;
6. a path via a sibling's provider link.

Based on the consideration for voluntary ISP participation, it follows that new architectures start as *overlays*, implemented only on a willing fraction of the global network, using the existing architecture as transport to connect the parts of the new deployment together.

Inter-domain structure itself is a results of routing policies reflecting operator incentives, rather than the physical layer network connectivity. Hence, all new networking architectures are also bound to face a similar policy structure as they are grounded to economical realities. When new kinds of traffic patterns, however, are introduced new kinds of traffic policies might become reasonable. Further this may change the incentives leading to new types of business models see Section 2.5.4 changing the economical landscape.

One motivational factor for ICN networking deployment is the expected more efficient and timely use of network resources. This is due to the object locator (and thus route) information being stored in network nodes as the data publications and requests are processed in the network, which enables sharing the communication and storage resources between multiple recipients. In the case of synchronous transmission, this kind of state and resource sharing essentially results in a multicast service [33]. Caching, in turn, enables *asynchronous sharing*. In general, ICN including NetInf has a similar incentive structure to peering for packet forwarding: savings on transit costs and reduced latency.

The possibility for transparent, architecturally integrated caching is a central piece of NetInf architecture. For the multilateral deployment we need to take the routing policy issues that reflect the economic incentive structure into account: When, exactly, should a NetInf domain cache which piece of data? In other words, who has an incentive to cache content given the current valley-free property of the Internet? It seems clear that Tier-1 domains, in their transit role, would have *no incentive to cache content if the content served from their caches were away from their customer links*, thereby reducing their transit revenue [34].

This generalizes to *all uphill domains* data served from caches will, in general, be away from their customer links, thereby reducing revenue. *There is no point serving data on your customer's behalf, unless the customer pays for the caching service.* However, on the *downhill* paths all domains seem to have an incentive to cache the data, as data served from caches is away from their (internal or external) transit links. The customer network domains, being at the bottom of the downhill paths, have the most obvious incentive for caching, as they can also directly benefit from the reduced latency.

ICN networking introduces a new aspect into the peering arrangements; potential for *content-oriented peering*. In the traditional peering, the links between the peers are kept in balance in terms of exchanged amount of traffic. A peering link is considered imbalanced if one peer is sending significantly more traffic than the other. A peer can impact to the ingress traffic by the means of BGP route announcements that are interpreted as willingness to deliver traffic to the announced addresses. In the content-oriented model this logic is reversed: the sender of the data first advertises data availability, and then receiver initiates the data transfer explicitly, and thus can be considered the benefactor of the ensuing traffic instead of the sender. This role-reversal should be taken into account when creating new policies on top of existing business models. This could even result into new accounting models for peering balance for ICN communication where not only the transferred data volume is taken into account but also the availability of the content from the caches of a peering partner. Volume based settlement fees could be balanced by advertising content availability which then generate more data over imbalanced peering links. The value of the available content compared may be higher to the transfer costs moving the focus from byte transfer to the content itself. Content peering requires that the content request routing (name resolution) is aware of the available NDOs of the peers (see more about a potential implementation in Section 2.4.2). See also more about content peering and business model evolution in Section 2.5.4.

As communication and storage sharing enables higher utilization of the existing peering relationships, some of the overall traffic moves down from the Tier-1s to the lower tiers. This is counter to the economical interests of Tier-1 operators selling pure transfer service. There seems to be a conflict of interest for Tier-1 operators to migrate to NetInf with the current service model because of conflicting incentives at packet transport level vs content level. The use of NetInf would result into less growth, or even a small decline in their packet transit traffic, and therefore less income with the current business model. Tier-1 operators should to apply new business models leveraging information object availability and delivery by participating onto name resolution and content caching.

2.5.4 Migration and Business Model Adoption

WP-A has studied the ecosystem relations for each of the investigated technology areas (NetInf, Open Connectivity Services (OConS) and Cloud Networking (CloNe)) of the project in deliverable D.A.8 [35]. The document presents an evolutionary business model adaption strategy with following steps for NetInf :

1. Internal network optimization;
2. Transparent caching;
3. Telco CDN: Providing commercial CDN services to content providers;
4. Telco CDN with Content Delivery Network Interconnection (CDNI): Extending the footprint of footprint of the Telco CDN service;
5. Virtual CDN: Outsourcing the customer relationships with content providers to a virtual CDN provider;
6. Elastic NetInf deployment: Extending the footprint of the Telco CDN by deploying NetInf caches in virtual machines in the cloud operated by a Cloud Network (CloNe) Provider.

Clearly the first two steps can be implemented unilaterally considering mostly projected capacity needs and related cost of delivery and to certain extend improved QoE in a customized way. The third step building on top of the two earlier ones shifts the focus from resource optimization towards QoE and SLA management of the contracted content from the service providers having an impact to business model. At this step the NetInf architecture should provide efficient content acquisition interfaces as well as monitoring and charging capabilities for premium contracted content (the latter ones have been intentionally scoped out from the project). At this stage the NetInf enabled operator will get a role in the content delivery value chain and the focus is moving from packet delivery to information delivery. The content naming with ni-names would take place at the acquisition interface at the time of publishing. The benefit for the content producer is better traceability and content availability through the accurate search functionality offered by the NetInf. Network operator benefits from more predictable service and network behaviour as it is the operator who positions the content into the caches with operators topology. Also operators delivery infrastructure, i.e. NetInf would integrate aspects of search functionality paving the way for operator to enter into this business area as well. (The specific search functionality, algorithms, ranking and other added value applications are not part of NetInf but applications that can take advantage of the NetInf feature set.)

In the fourth step of migration independent NetInf operator islands will start to federate and cover wide segments of the content delivery value chain. Latest at this stage the multilateral adoption is needed and new modes of peering are expected to emerge as described in the previous section and exemplified by the CDNI working group of the IETF. The last two steps are not any more a pure (technical) migrational steps as they assume wider infrastructure support beyond NetInf so we do not consider them here any further. Interested readers are advised to look more details about those steps in D.A.8 [35].

2.6 Summary

In this chapter we have provided a summary of the NetInf architecture and recent updates to it. Partly, these updates have been motivated by the results from extensive prototyping and evaluation activities that have been undertaken in the project. Overall, the core design principles remained valid and useful, and most of the work has focused on incremental changes and extensions such as the advancements of the NetInf naming scheme and the different Convergence Layer specifications.

In the following Chapter 3, we will describe how the architecture has been leveraged to develop prototypes based on running code and networks of NetInf components. This includes individual components such as Convergence Layer modules as well as NetInf nodes and network emulation frameworks. This will be followed by Chapter 4 providing the actual evaluation results of the different system components.

3 NetInf Prototypes

Prototype implementation of NetInf systems and applications is an integral part of the activity carried out in the work package and, along with the related evaluation activities, has become a major focus in the second part of the SAIL project. NetInf has developed proof-of-concept prototypes and experimental evaluations play a key role in demonstrating the feasibility and assessing the advantages of the emerging information-centric paradigm.

NetInf concepts and functions are validated by performing experiments focusing on specific components and scenarios in order to assess the consistency and viability of the overall architecture design. The experimental development work has also already made the prototype implementation of selected NetInf tools and components available as open-source; the availability of new releases and more prototype components is expected by the end of the project. The detailed description of the currently available NetInf tools released as open-source by the SAIL project can be found in Section 3.2.

The main scenario that has been identified as suitable to demonstrate the benefits of the NetInf approach and integrating experimental work form different partners is the Event with Large Crowds (EwLC) scenario, already introduced in our deliverable D.B.2/D-3.2 [5]. The scenario aims at showing the benefits of the paradigm for mobile clients in such an environment, augmenting current communication infrastructure by effective local NetInf communication. A number of the prototype components described in this chapter have been developed or adapted to eventually contribute to the demonstration of the EwLC scenario. Namely the NEC NetInf Router Platform (NNRP), the NiProxy name resolution server and the NetInf mobile client for Android, will integrate and participate to the EwLC experimental setup. In this regard, the integration of components in the EwLC scenario will be discussed in Section 3.3.1, while Section 3.3.2 will introduce the final demonstration plans.

The NNRP platform, described in Section 3.1.1, is a modular and extensible router prototype that implements several NetInf node functions, including UDP and HTTP convergence layers; it will act as an infrastructure node in the EwLC setup. NNRP is also used by partners for transport experiments. An example is represented by the transport control module described in Section 3.1.1.4, which implements a receiver-based flow control protocol. Moreover, on the routing functions side, the prototype implementation of an extension of the NNRP router, aiming to experimentally verify a global routing scheme based on the routing hints approach, is also introduced in Section 3.1.1.5. The NiProxy prototype (Section 3.1.2) provides name resolution services for NetInf clients communicating via HTTP convergence layer; the Name Resolution System (NRS) component was especially designed to fit into the EwLC scenario, performing two roles: both running in mobile clients for providing local resolution services and acting as an infrastructure NRS node. The Android phone prototype, described in Section 3.1.3, is a newly developed NetInf application for mobile devices, providing local information sharing and infrastructure connectivity, it has been designed for acting as a NetInf-enabled client device in the EwLC demonstration setup.

In addition to those contributing to the EwLC scenario demonstration, other prototypes have been developed in order to show the feasibility of NetInf architecture or focusing on the evaluation of specific issues, they are also described in this chapter. The developed Global Information Network (GIN) prototype introduced in Section 3.4 is a comprehensive node platform prototype for an information centric network based on the technical approach fully described in the GIN Technical Report [36]. It aims to proof-of-concept the feasibility of the approach as a proposal for a target evolutionary architecture able to deal with heterogeneous underlays. The prototype is providing several features like: name based routing and forwarding over heterogeneous networks (IPv4, IPv6, Ethernet), name resolution based on a distributed MDHT infrastructure, in-network registration and storage of named objects, multicast ping of named objects, end-to-end receiver-based multipath retrieval of named objects.

A prototype development focused on enabling NetInf communication in a Delay- and Disruption-Tolerant Networking (DTN) environment is described in Section 3.1.4. A mapping from the NetInf architecture onto DTN has been implemented; adding extra features to the Bundle Protocol (BP) and using some existing ones to meet the requirements of NetInf, the BP and the DTN substrate have been adapted as a useful infrastructure for a NetInf Convergence Layer (CL).

3.1 NetInf Prototype Components

This section describes the different NetInf prototype components that have been developed by SAIL partners: the NEC NetInf Router (Section 3.1.1), a NetInf receiver-based flow control module for NNRP (Section 3.1.1.4), a NetInf global routing module for NNRP (Section 3.1.1.5), a NetInf name resolution service prototype Section 3.1.2, a mobile device application prototype (Section 3.1.3), and a NetInf Delay-Tolerant Networking prototype (Section 3.1.4).

3.1.1 NEC NetInf Router Platform

NNRP is NEC's NetInf prototype that implements several NetInf node functions:

- Full support for the ni naming scheme, including name-content binding validation;
- HTTP Convergence Layer, to exchange objects with other nodes;
- UDP 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.

NNRP is a modular and extensible infrastructure for easy development of ICN networks and testbeds. NNRP is implemented in standard C, for POSIX-compatible operating systems (mostly tested on Linux).

The architecture of NNRP, like many other routing platforms, is centred on chains of modules which operate on the passing of messages (Figure 3.1). Modules can be added to the system at compile-time and at run-time, and the chaining concepts enables flexible creation of module pipelines. Module chains and chain configurations are defined entirely in the configuration file.

Incoming messages are injected in chains by the appropriate convergence layer modules, filtered by the modules, moved around through different chains and eventually forwarded to other nodes. This great flexibility allows NNRP to be used on any node in the demo setup.

3.1.1.1 General NetInf object model and NNRP object model

NetInf model NetInf messages have the structure indicated in Figure 3.2. The *data* and *metadata* parts are not mandatory.

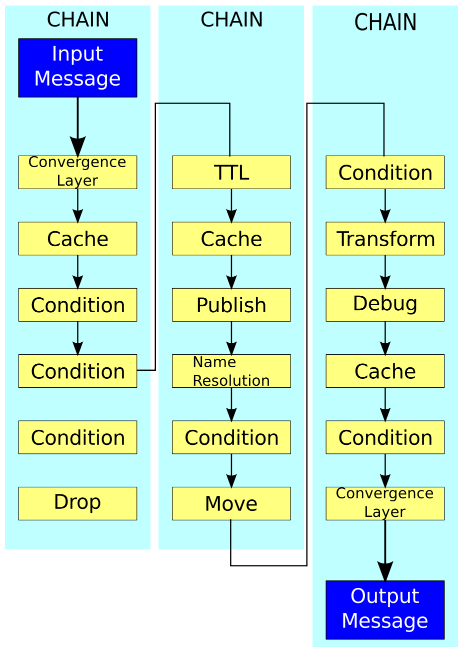


Figure 3.1: Detailed view of NNRP

TYPE
NAME
METADATA *Key: value* ...
DATA *(PAYLOAD)*

Figure 3.2: Abstract structure of a NetInf message

Message types The message type indicates an action or a reply. The names used here are the ones used inside NNRP, some are functionally equivalent to differently named NetInf message types. The supported message types so far are:

GET Request an object.

PUT Publish an object.

CHUNK Carries a chunk of an object around. Usually used as reply to a GET.

ROUTING Not used, planned for future use for routing purposes.

ERROR Indicates that an error occurred. Usually used as a reply to GET or PUT.

NRS Carries a locator of an object. Usually used as a reply to GET. Usually without payload.

SEARCH Not used, planned for future use for semantic searches.

REPLY Indicates a successful request, usually without payload. Usually used as a reply to PUT.

Naming scheme NNRP uses the ni format, but can also operate on opaque strings (if name-data-integrity validation is not needed).

Metadata Metadata is some state attached to a NetInf message. It is a collection of $\langle key, value\rangle$ pairs, where both key and value are plain strings. The keys are considered case-insensitive and are unique in each message.

3.1.1.2 NNRP model

The NNRP model (Figure 3.3) is a simple extension of the NetInf model, with additional state added to each message. This additional state is only used inside NNRP, it is created for incoming messages and discarded from outgoing messages.

NetInf Message
META-METADATA *Key: value* ...

Figure 3.3: Abstract structure of a NNRP message

This set of NNRP-specific state will be called from now on "meta-metadata", because it is similar to the metadata of the NetInf message, but one layer above it.

3.1.1.3 NNRP-specific basic concepts

The key objects in NNRP are modules, instances and chains.

Modules A module is a plug-in for NNRP. It can perform many functions and can interact with other modules. It can be implemented either as a shared library or as a separate program, interacting with the core over sockets. Some modules will perform a system-wide task on their own, other modules need to be instantiated to perform their actions.

Each module loaded at run time or connected over a socket is given a unique module-id.

Instances An instance of a module is like an instance of an object in an object-oriented environment, where the module represents the class of the object. Instances have parameters to influence their behaviour (like parameters of constructors in object-oriented languages), and can hold a private state.

Each instance is given upon instantiation a unique instance-id, which exists in a different namespace than the module-id.

Filters, sources, sinks Most modules are meant to operate on messages. An instance of a module can be used in up to 3 different ways: as a *filter*, as a *source*, as a *sink*. Filter behaviour is influenced by the contents of the message being processed, including metadata and meta-metadata, parameters passed to the module and global parameters (configuration options).

Filters A filter is an object that transforms a message. It receives a message in input and optionally returns a (potentially modified) message.

Sinks A sink object behaves mostly like a filter, except that it's always the last of any chain, and the output message is dropped after being processed, unless it is moved to a different chain. See paragraph below for more info about chains.

Sources A source object generates messages and put them into the system. Examples of sources are the implementation of the convergence layers, since they receive messages externally and injects them into the system.

Chains Chains are (uniquely) named sequences of filter instances, with an optional source and an optional sink. Messages are generated by sources and then passed on through the chain. Filters can alter or drop the message or cause it to be moved or copied to a different chain. Each chain can only have up to one source and one sink, and any number of filters. Usually only a few chains will actually have a source. The output of each filter is fed to the input of the following filter, new incoming messages generated by source modules are immediately fed to the first module in the chain.

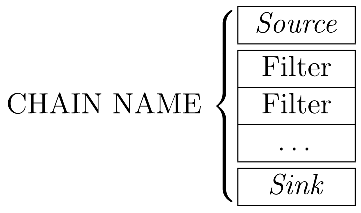


Figure 3.4: Example of chain

Moving, copying and dropping A message can be dropped or moved or copied to a different chain by any filter. When moved or copied to a new chain, messages start to be processed by the filters; sources are skipped.

3.1.1.4 NetInf receiver-based flow control module

A NetInf node may communicate using any CL on top of a complete transport-layer protocol (TCP and HTTP being the most evident among others) and send NDOs on a hop-by-hop basis back to demanding clients. However, this solution means that each NetInf node will have to re-assemble each NDO, leading to both unnecessary latency and CPU consumption in case of large objects. In our deliverable DB2/D-3.2 [5] (Section 3.2.2) and in a published paper [37], a solution is described in the form of a receiver-based flow control protocol, which allows to send only small NDOs through the convergence layers. This section presents a NNRP module which implements this protocol. It also illustrates the modularity of NNRP, which allows to add easily additional or auxiliary functionalities.

Functions The module provides the following functions:

- **Flow control**: the module controls at which rate data are requested from the network, in order to fully utilize the network capacity without creating an overload;
- **Error recovery**: the module guarantees reliable data transfers by re-expressing requests in case of data packet losses;
- **Reassembly**: the module requests data of small pieces, but is able to reassemble the whole object before passing it to the upper layer or application.

Design principles The module manages a Additive Increase Multiplicative Decrease (AIMD) request windows in a similar manner to TCP. It differs from TCP in two aspects:

- it is a receiver-driven protocol, which operates solely at the receiver side; this is possible because data fragmentation is not part of the protocol. One main advantage of receiver-driven protocols is that they allow to connect simultaneously to several sources;
- it allows out-of-order delivery; with the presence of caching and possibly parallel download from multiple sources, there is no guarantee that the NDOs forming an Application Data Unit (ADU) are received in request order. NetInf flow control module hence does not trigger a congestion event when receiving NDOs out-of-order, even repeatedly.

Implementation We assume that when publishing a new ADU, the publisher actually publishes (either explicitly or using a NetInf-aware application) both a set of small NDOs which, once reassembled, build the whole ADU, and a specific NDO which contains the list of these small fragments. The name of the later is by convention the name of the object as seen by the application, with /s0 added as suffix.

Upon request for an object (with name "a") from an application, the module first issues a GET request for this list NDO called "a/s0" in order to obtain the list of the related NDOs. The protocol is then brought in place to carry out the data retrieval, based on the list of NDOs to request. The UDP CL is used for each GET request. Note that this is totally transparent for the application, which just requested an object, and received it as if it were a single NDO.

The NDOs list does not have a specification at the moment. For instance, the example shown in Figure 3.5 simply indicates to replace the s0 suffix with the NDO names preceded by the *irel* keyword. Absolute naming for the NDOs is also supported.

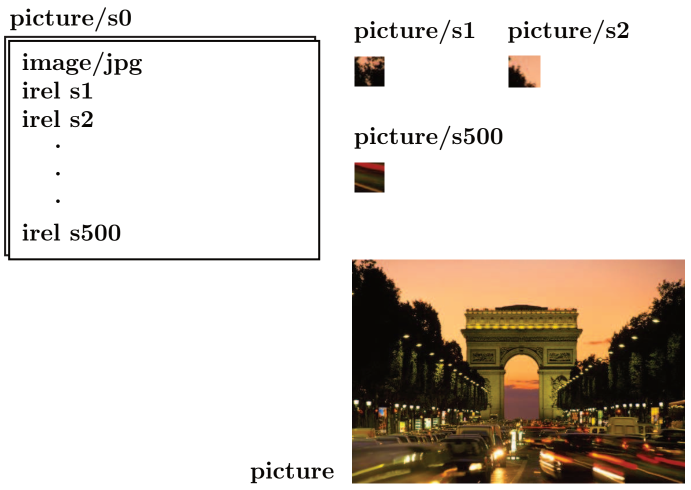


Figure 3.5: Example of NDOs list as a text file, along with NDOs that form the main ADU

Receivers use a request window to manage the transmission. In the same way as the NDOs list is requested, NetInf GET messages are sent to obtain the NDOs. The window is occupied by such pending request messages, and the number of requests may not exceed the current window size W. Next, when a response is received, the window size is increased by $\frac{A}{W}$, where A is a constant factor. Next, in order to take measures against network congestion, the Round Trip Time (RTT) is used. Congestion itself is determined by a timeout mechanism based on received objects' RTT value. Given a current timeout value T, any object inside the current window size scope not received under the said period is considered a congestion notification and the window size is multiplied by a factor $\beta < 1$, reducing its size. The timeout value is stored temporarily and consequent timeouts will not induce another window size reduction for a duration equal to the stored value. After its expiration, window size reductions may take place for new timeout values.

Lastly, by using a constant factor δ, T is determined by the following formula: $T = (1 - \delta)\mathrm{RTT}_{\mathrm{min}} + \delta\mathrm{RTT}_{\mathrm{max}}$, where $\mathrm{RTT}_{\mathrm{min}}$ and $\mathrm{RTT}_{\mathrm{max}}$ are the shortest and longest RTT values measured over a certain number of received NDOs, respectively. By default, last 20 objects' RTT values are stored and used for this purpose. This mechanism ensures that dynamic latency is still taken in account, whilst being able to recover from sudden shifts on the network traffic. The module uses a fixed timeout value before the first object reception. By default, $\beta = 0.5$, $\delta = 0.5$ and $A = 1$ are chosen as window parameters.

The next request to be fed is mainly generated sequentially (i.e. starting with s1 and increased by 1 at each new request generation), even though it may also depend on another mechanism such as an object-priority list, depicting which fragments should be obtained as soon as possible. This is the default behaviour in an ideal setup where no packet loss occurs. On the other hand, NDO losses require other measures to ensure a correct assembly. Firstly, a window reduction will leave some of the late requests undecided; they may be received or lost as well, even though the window will not handle them until the window size reaches its old value once again. Any such request that has not been received so far may be adopted by the window in case the window size increases and the object has not been received yet. Secondly, a congestion signal will add the relevant request position to a discarded requests queue. Unless the loss is actually triggered by a late arrival, it remains there as a candidate for upcoming requests. These supplementary mechanisms are employed when there is no consequent object request left to generate. However, this may also be shaped easily into a different behaviour in case another policy is envisaged to be put in practice for future applications.

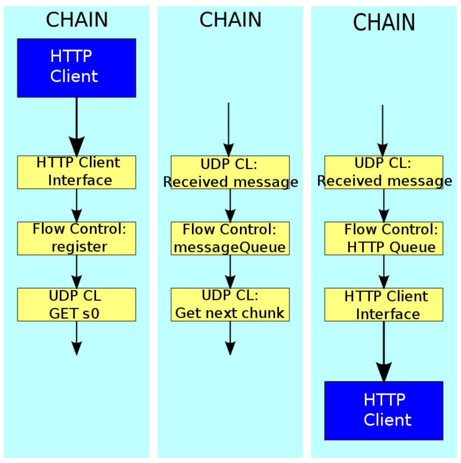


Figure 3.6: Example of chains for flow control module integration

Module configuration Figure 3.6 gives an example of three chains for integrating this flow control module to NNRP. We present here the example of an HTTP client, which on the top left requests an HTTP object. This HTTP GET request is translated into a NetInf GET request by the HTTP Client Interface module. The Flow Controller module then receives the GET request, and initiates a structure to store the requests and data, then send a NetInf GET requests for the list of the required NDOs, using the UDP CL. Any received NDO is then forwarded by the UDP CL to the flow controller module, which both update the window and the list of missing requests, and store the data. When a request receive no corresponding data before a timeout occurs, the module will also reduce the windows size, and add once again the request to the list of missing requests. Finally, once eventually all NDOs are received, the module will re-assemble then into a single NetInf object, which will be returned to the requester (in this case the HTTP Client Interface module, which will then return it to the HTTP client).

3.1.1.5 NetInf global routing using routing hints

We have started to implement a prototype of the NetInf global routing scheme based on routing hints that is described in Deliverable D-3.2 *(D-B.2) NetInf Content Delivery and Operations* [5, Section 2.3]. The implementation is done as a set of modules in the NNRP router.

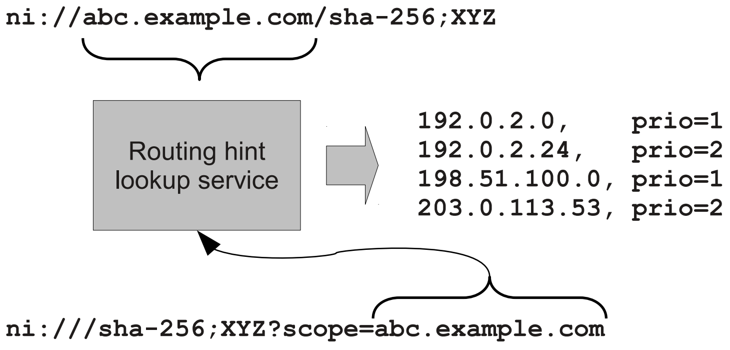

ni://abc.example.com/sha-256;XYZ
Routing hint lookup service
192.0.2.0, prio=1
192.0.2.24, prio=2
198.51.100.0, prio=1
203.0.113.53, prio=2
ni:///sha-256;XYZ?scope=abc.example.com

Figure 3.7: The routing hint lookup service

To recapitulate, 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, as illustrated in Figure 3.7. One or more *scope* URI parameters can also be used to explicitly specify an NDO aggregate that in the same way is mapped into hints as shown at the bottom.
- *NetInf BGP routing system* for the NetInf routers in the default-free zone.
- *Routing hints* that aid global routing by providing aggregation for routing information.
- *Forwarding tables* in each NetInf router that maps ni: URIs and/or routing hints into the address of the next hop to forward the request to.

To start with, we focus on implementing the NetInf router forwarding process in the NNRP router as a set of modules that are configured as part of the NNRP filter chains. The forwarding process includes these steps:

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

These steps depend on the configuration of the particular router. The global NRS lookup step is for instance typically done by an ingress router, as discussed above in Section 2.1.3.

Table 3.1 shows an example routing hint 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.

routing hint	CL-specific next-hop
192.0.2.0	http://global.example.com/netinfproto/get
192.0.2.24	http://edge.example.com/netinfproto/get
10.1.10.1	http://local.example.com/netinfproto/get

Table 3.1: Example routing hint forwarding table

3.1.2 NetInf NRS — NiProxy

NiProxy provides NRS services for NetInf nodes to fetch, publish, search and unpublish data. It implements NetInf protocol [10] with some extensions like support of unpublish operation. NiProxy is constructed as a single node for the EwLC scenario, but also a distributed NRS can be implemented by multiple NRS nodes interconnected through DHT. NiProxy is implemented in Java and its HTTP server implementation is based on Apache Commons^{TM}HTTP modules[1]. Clients communicate with NiProxy using HTTP Convergence Layer (CL). NetInf NRS was made especially for EwLC scenarios, where it has two roles; i) it is running in mobile nodes where it provides NRS services locally to register and unregister local addresses like Media Access Control (MAC) addresses and ii) respectively it provides similar services with a wider scope at the infrastructure side including data storage services for published NDOs.

Clients can either publish only locators by which an NDO can then be requested or the NDO itself could be published. In the former case, NRS is only responsible of resolving publication's locator(s) and some entity needs to maintain the NDO's reachability. For instance, the NDO could be available in the mobile node that was the publisher of locator(s) or then it could be hosted by some third party service. In the latter case, a client just publish and push the NDO into a network where NiProxy stores the NDO and then the client can for instance disconnect itself without sacrificing the NDO's availability.

With publish, clients can also include a meta-data that describe what the publishing is about. For instance, the meta-data can include a content type and type specific info such as video codec type. Further it can contain keywords categorizing the data to be published. The meta-data and keywords can then be used by search functionality to search published data. Search functionality takes one to many strings as search keys and then it tries to find all publications for which all provided keys are matching. Ni names of all matching publications are then returned back to the requested client.

Compared to the NetInf protocol specification, the support of unpublishing was added due to need of it for EwLC scenarios. Its implementation follows closely how other NetInf protocol messages are defined. Basically unpublishing is like <PUBLISH, PUBLISH-RESP> message exchange with fewer options in the message. In order to do unpublishing, we only need to identify the published data to be unpublished and other things needed for publishing like meta-data can be omitted. For NRS, this simply means that it needs to update its internal mappings and additionally remove NDO from its local storage, if it exists. However, it should be noted that if the published data has been already cached autonomously (without NRS's involvement) in other places like in NNRP, then the content may still be served from those, typically on-path, caches after unpublishing it.

3.1.3 Android Device Prototype for the Event with Large Crowd Scenario

As shown in Figure 3.8, the Android device prototype is based on five main elements:

- The application layer;

[1]http://commons.apache.org/

- The control layer;
- The internal cache layer;
- External Mobile Data Storage (MDS) units of nearby devices;
- Different external servers.

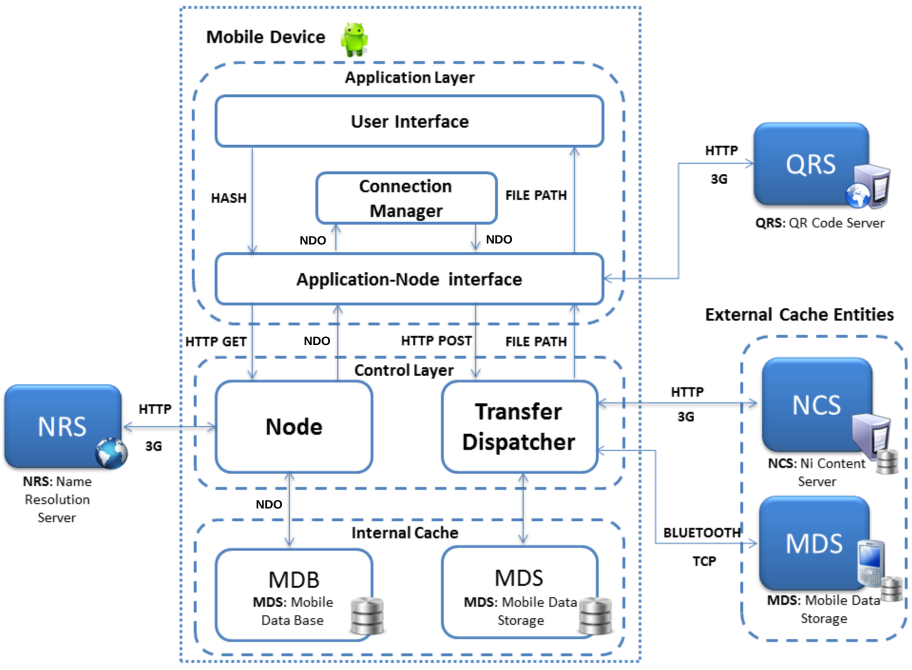


Figure 3.8: Android device prototype for the Event with Large Crowd

The application, control and internal cache layers are located within the mobile device and will be running in every NetInf-enabled device. The fourth component, the MDS of nearby devices, will be running in adjacent NetInf-enabled devices running the same architecture. Finally, the external servers are not attached to any particular location; they just need to have IP connectivity to the Internet. The Android device prototype interacts with an NRS server prototype based on the NiProxy described in Section 3.1.2.

The application layer interacts with the user and with the interface to the control layer as shown in the figure. Also, it uses the connection manager to participate in the discovery process of Bluetooth and WiFi Direct. All the communication between the application and the control layer is done over HTTP. Finally, this layer will also communicate with the QR-code Server (QRS) in order to allow for an easy way to obtain the NDO names.

The Control Layer is based on the OpenNetInf architecture components [38]. In the Control layer, the two main components are the Node and the Transfer Dispatcher. The Node is the component responsible for operations related to NDOs. It allows components running in the application layer to get, put, delete, and cache NDOs. To provide these functions, the Node will communicate with two components: one that resides within the device (the Mobile Database (MDB)) and one that resides in the external network (the Name Resolution Server). The MDB resolves NDOs locally and it will contain a list of all the NDOs that have been put or cached by the user. The Name Resolution Server is an external entity that allows NetInf-enabled devices to register NDOs and get locators associated with registered NDOs.

The Transfer Dispatcher is the component responsible for handling of NDO content (transferring, serving, storing, etc). Given a target NDO, the Transfer Dispatcher is capable of retrieving the bytes associated with it from a remote peer MDS or from a NetInf Content Server (NCS), and store it in the local MDS. This component is also responsible for listening for other adjacent peers requests for file transfer. Therefore, the Transfer Dispatcher is in charge of two servers: the Bluetooth server and the TCP server. Finally, the internal cache layer is responsible for managing two different types of storage: the MDB which is a structured data storage unit based on a SQLite database and the MDS which uses the file data storage capability provided by Android.

The Android prototype is used in the EwLC. The evaluation of this scenario is described in Section 4.3.2.

3.1.4 NetInf via Delay-Tolerant Networking

The DTN architecture as described in RFC 4838 [39] envisages that DTN nodes will manage a store of messages in transmission. In contrast to packet networks, such as those based on the IP, these messages are expected to be complete units of content. Clearly, this is similar in concept to the carriage of NDOs and the ad-hoc in-network caching that are at the heart of the NetInf architecture. We have therefore developed a mapping from the NetInf architecture onto a realization DTN; this is still a work in progress and, as with the other CLs presented here some changes are likely in the future.

The realization of DTN championed by the Internet Research Task Force (IRTF) Delay-Tolerant Networking Research Group (DTNRG)[2] is based on the DTN Bundle Protocol (BP) [40] that handles the communication of "bundles" across a DTN-enabled network. The content of an NDO can be carried intact as the payload of a single bundle. However, whilst the choice of Protocol Data Unit (PDU) semantics and the key role of in-network caching mean that using the BP as a CL for NetInf is relatively straightforward, there are several challenges:

- The naming of bundles is a private matter, relevant only to the BP, and designed to ensure that duplicate bundles can be discarded, whereas a key characteristic of NetInf is the name of the NDO that is the public identifier of the content and is used to match NetInf requests with cached content.
- Routing and forwarding of bundles in the BP is controlled by DTN destination addresses known as Endpoint Identifiers (EIDs) that are selected by the originator of the bundle, whereas in NetInf routing and forwarding of NDO related messages is driven by the name and the originator has essentially no control over the location where a message is "satisfied".
- NetInf messages may be forwarded to many nodes either to locate an NDO to satisfy a GET, to carry out SEARCHes of multiple caches, or to PUBLISH an NDO into multiple caches, whereas although late address binding in the BP may mean that a bundle travels on multiple routes, conventionally bundles are directed to specific nodes pre-determined by the message originator.

To adapt the BP and the DTN substrate as a useful infrastructure for a NetInf CL, we have added one extra feature to the BP and used some existing features to meet the requirements of the NetInf architecture and protocol.

3.1.4.1 The Bundle Protocol Query (BPQ) Block

To carry the name of an NDO or search strings needed to identify content in NetInf messages, an extra BP block type, the BPQ, has been defined [41]. Unlike the bundle identifiers used within the

[2]http://www.dtnrg.org

BP, the value carried in the BPQ is defined by the creator of a bundle and maybe used, but not modified, by the nodes that process a bundle.

As is usual for DTN networks, bundles, including those containing a BPQ, are cached in nodes through which they pass. According to [41], the Bundle Protocol Agent (BPA) in BPQ-enabled nodes carries out extra processing on bundles containing a BPQ block.

In addition to the BPQ value the block contains a number of other fields that are used to control the extra processing. Those that are important for the NetInf CL are:

BPQ-kind Indicates whether the bundle containing the BPQ block represents a query (with no payload) or a response (generally with metadata and, usually but not always, with a content payload).

Matching-Rule Used to determine how queries and responses should be associated by the BPA.

Original Source Identification May be used as a locator to indicate a potential source for the content in a response in addition to the source of the response message.

In the cache of a BPQ-enabled node, bundles with a BPQ with a BPQ-Kind of "response" ("response bundles") are treated specially:

- such bundles are separately indexed by the BPQ-Value in the bundle, and
- such bundles are retained until their lifetime expires, subject to resource availability constraints, even if they would otherwise be candidates for early deletion.

When a bundle containing a BPQ block with the BPQ-Kind of "query" arrives in a BPQ-enabled node, the BPA compares the BPQ-Value in the incoming bundle with the BPQ-Value in each of the "response bundles" in its cache that have the same Matching-Rule as the incoming bundle. The value of Matching-Rule specifies how this comparison is to be carried out. The current BPQ specification only defines an exact (string) match but other rules could be provided.

Any "response bundles" that match are copied, readdressed using the source of the query as destination and scheduled for transmission towards that destination.

3.1.4.2 Using the BP with BPQs as a NetInf CL

The messages of the NetInf protocol [10] are mapped onto the BP using bundles with a BPQ block and, if required, a metadata block [42] to carry any affiliated data as a JSON object encoded as a string. The mapping is as follows:

GET BPQ-Kind of "query", with the Matching-Rule set to "exact match" and the BPQ-Value set to the ni scheme URI of the NDO sought. The msg-id and any ext values specified are carried in the metadata block. The payload is empty.

GET-RESP BPQ-Kind of "response", with same BPQ-Value and Matching-Rule as the GET message. The msg-id and other affiliated data as documented in Section 2.3.2.1 are carried in the metadata block. If the NDO content is available, it is carried in the bundle payload.

PUBLISH BPQ-Kind of "response", with the Matching-Rule set to "exact match" and the BPQ-Value set to the ni scheme URI of the NDO carried. The msg-id, any locators, the"full-ndo-flag", indicating if the content is included in this publication, and ext values specified are carried in the metadata block. If the content is being published (full-ndo-flag is set), the content is carried as the bundle payload.

PUBLISH-RESP The PUBLISH-RESP message does not require a BPQ block since it is purely needed to inform the originator that the content has been published in one or more nodes (several responses may be generated from any BPQ-enabled nodes that the PUBLISH message passes through). If required, confirmation that the content has been published can be obtained by requesting status messages for bundle reception[3].

SEARCH BPQ-Kind of “query”, with the Matching-Rule currently set to “exact match” but it is envisaged that other rules could be defined that would allow more detailed specification of the type of search to be carried out. The BPQ-Value contains the search query string that controls the search. The msg-id and any ext values specified are carried in the metadata block. The payload is empty.

SEARCH-RESP BPQ-kind of “response”, with same BPQ-Value and Matching-Rule as the GET message. The msg-id and other affiliated data including the set of search results (ni scheme URIs) as documented in Section 2.3.2.3 are carried in the metadata block. The payload is empty.

When a bundle carrying a NetInf GET or SEARCH message passes through a BPQ-enabled node, the BPA will be triggered by the BPQ-Kind of “query” to carry out a search of cached “response bundles” as described in Section 3.1.4.1 and, if there is one or more matches, generate “response bundles” directed back to the source of the GET or SEARCH bundle that are copies of the cached bundle except for a modified BPQ block with as described in Section 3.1.4.1 and carrying the msg-id from the query.

When a bundle carrying a NetInf PUBLISH, GET-RESP or SEARCH-RESP passes through a BPQ-enabled node, the BPA will be triggered by the BPQ-Kind of “response” to enter the bundle into the index of “response bundles” so that it can be matched by subsequent queries.

In order to allow a bundle to be readily passed to any node implementing the NetInf, the DTN EID destination for NetInf GET, PUBLISH and SEARCH messages will be set to a specialized multi-node EID that is ’expressed’ or advertised by every node that offers the relevant NetInf facilities.

Applications located on BPQ enabled nodes can register to receive bundles addressed to the destination EID for NetInf messages. One usage for such an application would be to provide a gateway between a network domain using the DTN BP CL and a well-connected domain using the HTTP and/or UDP CLs.

3.1.4.3 Implementation of the NetInf DTN CL

A prototype implementation of the NetInf DTN CL has been produced using the DTNRG-championed open-source Reference implementation of the DTN BP (DTN2) reference implementation[4] of the BP and the BPA. DTN2 is released subject to the Apache v.2 license[5]. The BPQ and the extra processing for bundles with BPQ blocks as specified in [41] is incorporated in release 2.9.0 of DTN2 available from Sourceforge[6].

In addition to the implementation of BPQ in the DTN2 BPA, the download contains a number of BPQ related applications to send GET or SEARCH messages using BPQ query blocks and receive the resulting responses, as well as sending out PUBLISH messages using BPQ response messages.

[3]In future an additional status message could be generated to indicate that a bundle had been recorded in the BPQ cache index.
[4]`http://www.dtnrg.org/wiki/Code`
[5]`http://www.apache.org/licenses/LICENSE-2.0`
[6]`https://sourceforge.net/projects/dtn/files/`. Note that version 2.9.0 of DTN2 requires version 1.6.0 of Oasys.

Additional software being incorporated in the "NetInf Device"[7] allows an application to apply the Apache Lucene indexing and searching software[8] to the bundles in the cache of a node running DTN2 and send a report of any files flagged as matching the search query.

Further work is in progress to provide a gateway between the DTN CL and the HTTP CL and test the implementation of the NetInf DTN CL.

3.2 NetInf Software Modules on Sourceforge

The SAIL project has released an open-source (subject to the Apache v.2 license[9]) set of tools for NetInf[10]. These implement various aspects of the NetInf protocol in different languages. At the time of writing, there are C, Python, PHP: Hypertext Preprocessor - a recursive acronym (PHP), Ruby, Clojure and Java implementations with the python, ruby and PHP code having seen the most development so far. Others are at various stages of maturity, for example, the C library has basic "ni" name processing only at present. We have made three "releases" to date, with the most recent being on October 15th 2012. On-going releases are planned for the remaining life of the project and thereafter.

The PHP (Section 3.2.1), Python (Section 3.2.2) and Ruby (Section 3.2.3) implementations are outlined below — for details, see the protocol specifications and code.

3.2.1 PHP HTTP NetInf Server

This section describes the NetInf PHP server prototype implementation that has been developed to implement the NetInf protocol[10] HTTP CL.

The PHP implementation was developed mainly in order to provide a platform against which other language binding implementations of the NetInf HTTP CL could be tested, and has been used in this mode by SAIL partners over the last 6 months. To this end, a "live" service [11] has been maintained to allow partners to do interoperability testing and for use during code-sprints. The PHP scripts that provide this service are part of the NetInf open-source release (see Section 3.2).

The PHP module also contains scripts that allow a web site operator running Apache to generate "ni" names and .well-known URLs for all the files on their site. This is intended to allow one to quickly take existing static content and make it available via NetInf. The release also contains Apache configuration fragments required to offer the PHP scripts on a web site.

The scripts themselves implement the GET, PUBLISH and SEARCH operations. GET requests will only succeed if the requested URI is present on the web site, either as a static file that is linked below the .well-known/ni subdirectory of the DocumentRoot, or in the PUBLISHed or SEARCHed object caches. There is no routing of any NetInf messages in this implementation.

The PUBLISH implementation checks for name-data integrity if a full object is supplied. If not, then only meta-data about the PUBLISH is stored (which would be returned to a GET for the same name). If the full object was supplied and name-data integrity checking passes, then the object and meta-data are both stored and will be available for retrieval. In order to avoid consuming storage on the test service, and to allow others to exercise failure-mode code paths, full objects that are PUBLISHed are deleted at the start of the next hour after which only meta-data will be returned in response to a GET.

[7]It will however not be incorporated in the Event with Large Crowd scenario.
[8]`http://lucene.apache.org/`
[9]`http://www.apache.org/licenses/LICENSE-2.0`
[10]`http://sourceforge.net/projects/netinf/`
[11]`http://village.n4c.eu/getputform.html`

For PUBLISHed objects, meta-data is accumulated for logging purposes but only a merged version of this is returned in response to GET requests. For example, if one PUBLISHes the same name twice, with the same locators, only one instance of that locator will be returned in the GET-RESP.

When a SEARCH is received, the PHP implementation turns that into a HTTP request to Wikipedia for the first 10 items (exactly) matching the search terms[12]. For each search-hit from Wikipedia, the PHP implementation fetches and caches the matching web page, generates an "ni" name (and .well-known URL) for each and then returns those in the NetInf format.

Subsequent GET requests for those "ni" names will then succeed. Objects downloaded from Wikipedia are generally small and so are not deleted regularly by the service.

The PHP implementation has been very useful whilst doing rapid prototyping exercises with other SAIL partners — the availability of a "live" service and easily modified server code has been hugely beneficial.

3.2.2 Python HTTP NetInf Clients and Server

This section describes the NetInf Python server prototype implementation and client applications that have been developed to implement the NetInf protocol[10] HTTP CL.

The server be configured either

- as a standalone "lightweight" HTTP server that is dedicated to serving HTTP requests related to the NetInf protocol only, or
- a plugin module for the Apache web server.

In either case, the server manages a cache of NDO content and metadata files and offers essentially the same set of services as the PHP server described in Section 3.2.1.

The server (`pyniserver`) can be configured to run on any machine that has an appropriate set of Python 2.x[13] modules installed.

For the standalone server, the address and port used by the server together with a number of other items (such as the location of the root of the cache directory tree) are configurable either via command line options or a configuration file.

The Apache version uses the mod-wsgi[14] module which implements the Python Web Server Gateway Interface (WSGI) interface[15].

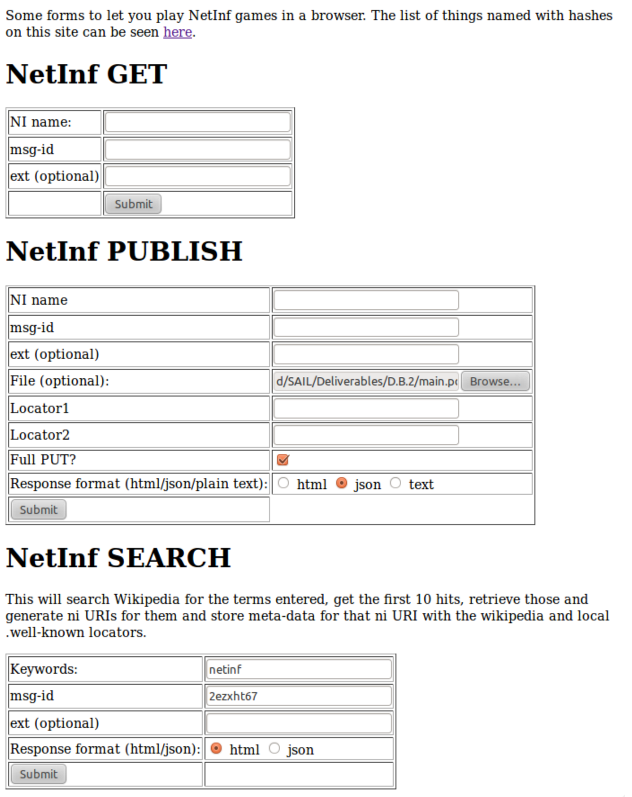
The source code is structured as a pure Python package named "nilib" and is installable using the standard "setuputils" mechanism. Full Doxygen documentation has been provided for the "nilib" package.

The server NetInf capabilities can be accessed either via a browser, using the form at `http://<server_netloc>/getputform.html` (see Figure 3.9) or via a set of command line utilities (`pyniget`, `pynipub` and `pynisearch`). All the programs have comprehensive help on the command line options available using the `--help` or `-h` command line option.

GET requests will currently only succeed if the NDO named by the URI in the request exists in the server's cache when the request is received. The server does not route requests of any kind to other places. The result of a successful GET operation via the browser form is shown in figures 3.10 and 3.11.

Unlike the PHP server, the Python server implements a single NDO cache and does not have facilities for accessing "static" files as it has no concept of a "Document Root". The cache can be populated either by PUBLISH operations or by SEARCH operations which access Wikipedia in the same way as the PHP server.

[12]Again, this was done to limit the load on Wikipedia and our test service.
[13]x equal to or greater than 6.
[14]http://code.google.com/p/modwsgi
[15]http://www.python.org/dev/peps/pep-0333/



Some forms to let you play NetInf games in a browser. The list of things named with hashes on this site can be seen here.

NetInf GET

NI name:	
msg-id	
ext (optional)	
	Submit

NetInf PUBLISH

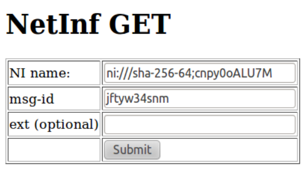
NI name	
msg-id	
ext (optional)	
File (optional):	d/SAIL/Deliverables/D.B.2/main.p(Browse...
Locator1	
Locator2	
Full PUT?	✓
Response format (html/json/plain text):	html [✓ json] text
Submit	

NetInf SEARCH

This will search Wikipedia for the terms entered, get the first 10 hits, retrieve those and generate ni URIs for them and store meta-data for that ni URI with the wikipedia and local .well-known locators.

Keywords:	netinf
msg-id	2ezxht67
ext (optional)	
Response format (html/json):	[✓ html] json
Submit	

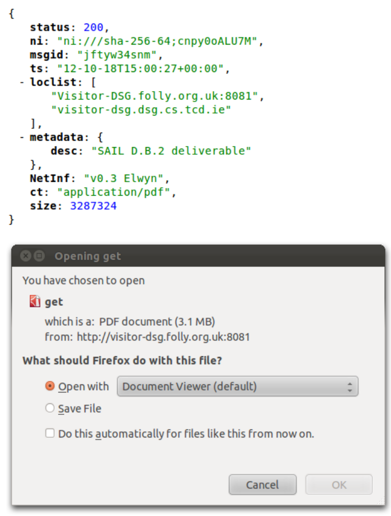
Figure 3.9: Python HTTP convergence layer — getputform.html



NetInf GET

NI name:	ni:///sha-256-64;cnpy0oALU7M
msg-id	jftyw34snm
ext (optional)	
	Submit

Figure 3.10: Getting an NDO



```
{
   status: 200,
   ni: "ni:///sha-256-64;cnpy0oALU7M",
   msgid: "jftyw34snm",
   ts: "12-10-18T15:00:27+00:00",
 - loclist: [
      "Visitor-DSG.folly.org.uk:8081",
      "visitor-dsg.dsg.cs.tcd.ie"
   ],
 - metadata: {
      desc: "SAIL D.B.2 deliverable"
   },
   NetInf: "v0.3 Elwyn",
   ct: "application/pdf",
   size: 3287324
}
```

Opening get

You have chosen to open

get

which is a: PDF document (3.1 MB)
from: http://visitor-dsg.folly.org.uk:8081

What should Firefox do with this file?

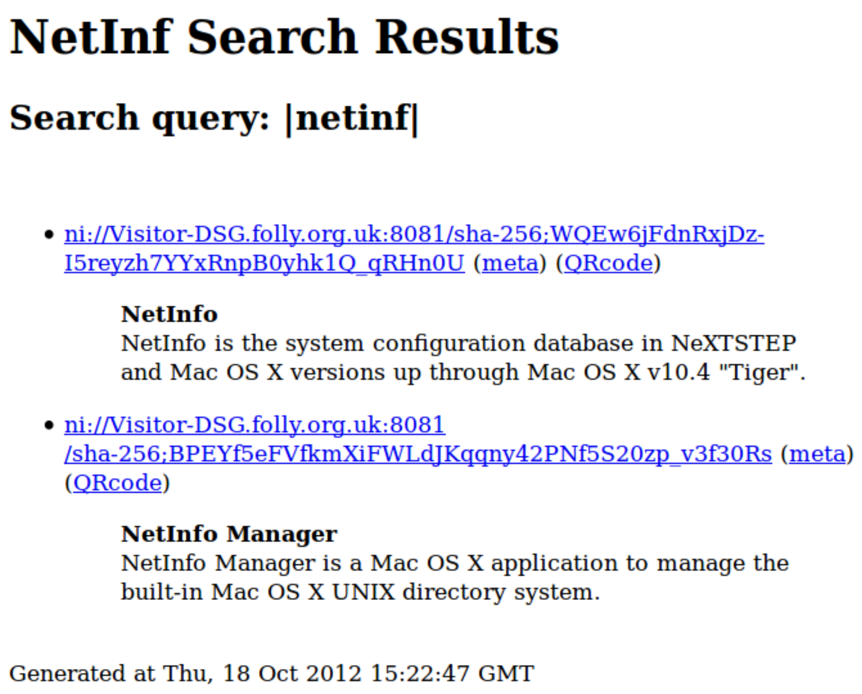
[✓ Open with] Document Viewer (default)
Save File
☐ Do this automatically for files like this from now on.

Cancel OK

Figure 3.11: The results of the GET

The PUBLISH operation checks name-data integrity if the content is received with the request and stores the content in its cache if the check succeeds. If the content is not received, the operation stores only affiliated data (metadata) for the NDO. If further PUBLISH operations are received for the same object, the metadata including any locators supplied are accumulated for logging purposes but the metadata will be summarized when returned with a GET operation or as part of the response to a PUBLISH operation. The summary will contain a list of all the different locators stored, a list of all the search query strings that resulted in the NDO being flagged plus the most recent values any other metadata items recorded.

The SEARCH operation sends a search query string to the Wikipedia OpenSearch interface and then generates ni URIs for the content of the first ten (or less) suggestions returned by the search request using the sha-256 digest algorithm. The resulting NDOs are stored in the server cache together with metadata recording how the NDO was generated, including the search query string and the query server used. The list of ni URIs generated is returned with the SEARCH-RESP message. An example results page from a SEARCH operation is shown in Figure 3.12.



NetInf Search Results

Search query: |netinf|

- ni://Visitor-DSG.folly.org.uk:8081/sha-256;WQEw6jFdnRxjDz-I5reyzh7YYxRnpB0yhk1Q_qRHn0U (meta) (QRcode)

  NetInfo
  NetInfo is the system configuration database in NeXTSTEP and Mac OS X versions up through Mac OS X v10.4 "Tiger".

- ni://Visitor-DSG.folly.org.uk:8081/sha-256;BPEYf5eFVfkmXiFWLdJKqqny42PNf5S20zp_v3f30Rs (meta) (QRcode)

  NetInfo Manager
  NetInfo Manager is a Mac OS X application to manage the built-in Mac OS X UNIX directory system.

Generated at Thu, 18 Oct 2012 15:22:47 GMT

Figure 3.12: Python HTTP convergence layer — Results page from a SEARCH operation

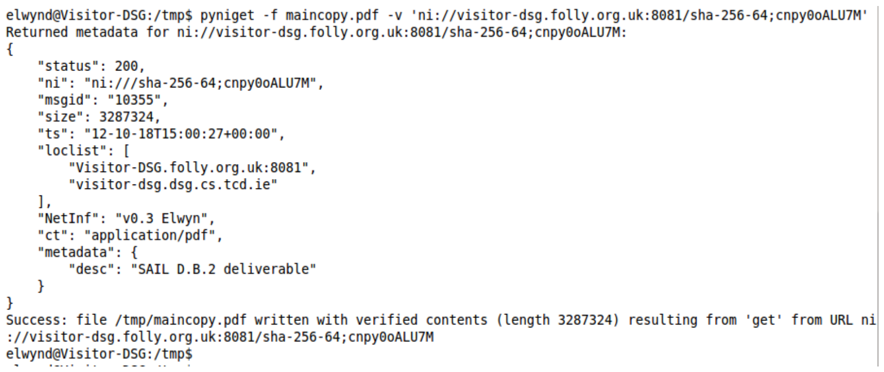
The server also implements a list operation via `http://<server-netloc>/netinfproto/list` which returns an HTML document listing the contents of the server cache separated by digest algorithm used. For each item in the cache, the list displays the ni scheme URI of the item. This is a clickable link if the NDO's content in the cache with the link URI being the HTTP .well-known name by which the NDO can be retrieved. The list also displays clickable links labelled *meta* and *QRcode* that retrieve the complete affiliated data of the NDO as a JSON encoded string and a QRcode image that encodes the ni URI for the NDO respectively.

The command line utility `pyniget` verifies the name-data integrity of the returned content if any against the supplied ni scheme URI, stores the verified content in a file and, on request, displays the affiliated JSON encoded data returned with the content. An example run of `pyniget` is shown in Figure 3.13.

The command line utility `pynipub` either generates the ni scheme URI from a supplied template (without the digest) and an arbitrarily named file of content, or verifies that a supplied complete ni URI matches the content file, before sending the PUBLISH request to the server.

The package also contains a command line utility `pynicl` that can generate an ni or nih scheme URI for a content file or verify the name-data integrity of a ni/nih and content file pair.

Work is still in progress on this package. During the remainder of the project we intend to integrate the server and client parts into a component that would provide a complete NetInf infrastructure for a node. We also intend to integrate the HTTP CL infrastructure with the DTN CL described in Section 3.1.4.



```
elwynd@Visitor-DSG:/tmp$ pyniget -f maincopy.pdf -v 'ni://visitor-dsg.folly.org.uk:8081/sha-256-64;cnpy0oALU7M'
Returned metadata for ni://visitor-dsg.folly.org.uk:8081/sha-256-64;cnpy0oALU7M:
{
    "status": 200,
    "ni": "ni:///sha-256-64;cnpy0oALU7M",
    "msgid": "10355",
    "size": 3287324,
    "ts": "12-10-18T15:00:27+00:00",
    "loclist": [
        "Visitor-DSG.folly.org.uk:8081",
        "visitor-dsg.dsg.cs.tcd.ie"
    ],
    "NetInf": "v0.3 Elwyn",
    "ct": "application/pdf",
    "metadata": {
        "desc": "SAIL D.B.2 deliverable"
    }
}
Success: file /tmp/maincopy.pdf written with verified contents (length 3287324) resulting from 'get' from URL ni
://visitor-dsg.folly.org.uk:8081/sha-256-64;cnpy0oALU7M
elwynd@Visitor-DSG:/tmp$
```

Figure 3.13: Python HTTP convergence layer — Using pyniget

3.2.3 Ruby Router and NetInf Tools

The Ruby implementation of the NetInf protocol consists of a standalone router and several command line tools, e.g., for creating names, requesting and publishing objects etc. The Ruby router implements both the HTTP and the UDP Convergence Layer of the NetInf protocol.

Ruby[16] is a dynamic, open source programming language that is extremely flexible (essential parts of Ruby can redefined and extended by users) and that provides a comprehensive, high-quality standard library, which makes it an ideal platform for rapid prototyping of NetInf functions.

The following Ruby programs are provided:

nid: The NetInf router that provides request forwarding, caching, routing, and search functionality;

mkni: a tool for generating ni names (for web objects or local files);

chkni: validates an ni URI against a file or web object;

ni2nih: converts an ni URI to the nih (human readable) format;

nih2ni: converts an nih URI to an ni URI;

ni2bin: converts an ni URI to the binary format;

ni2qr: creates QR code for an ni URI;

getni: GETs a named data object (using UDP and HTTP convergence layers);

pubni: PUBLISHes a named data object (using HTTP convergence layer);

regni: REGISTERs a named data object (using HTTP convergence layer);

searchni: sends a SEARCH request to a another NetInf node.

The router and the tools are based on the following foundations:

Ni URI class The ni URI class implements ni URI handling, i.e., creating, parsing, validating etc. It is derived from Ruby's standard library's URI class so that the Ruby URI parser is extended to support the `ni` URI scheme.

[16]http://www.ruby-lang.org/

NetInf protocol implementation The NetInf protocol implementation consists of a `NetInf` base class that defines the interface with NetInf protocol functions (GET etc.), and that is derived by classes implementing a specific Convergence Layer (`NetInfHTTP` and `NetInfUDP`).

Object mapping and routing table configuration The `ni` router can be statically configured using domain-specific languages (DSL). For example, the following example defines a mapping for some objects (individual objects or a collection in file system directory) to ni names and makes the corresponding Named Data Objects available in the router:

```
#mapNi will map the specified resource to an ni name and make this
#available. You have to specify the resource identifier (HTTP URI or
#file URI), the authority (can be empty string), and you can
#optionally specify options (see examples).

#mapNi maps exactly one resource to an ni name.

#map <URI> <auth> <options>
#URI: <file-URI> | <http-URI>

mapNi "http://www.ietf.org/images/ietflogotrans.gif",  "www.ietf.org"
map "file:///tmp/file", "example.com", {:hash-algo => "sha-256"}
map "file:///var/music", "example.com", {:recurse => true}
mapNi localFile("dirk.jpg"), "example.com"
mapNi localFile("README"), "neclab.eu"
```

The following examples show how mappings can be installed in the router that make the router transform names of requested NDOs into locators (file locator or HTTP URIs):

```
#mountNi will install a mapping that is transforming the ni URI to
#another URI (HTTP URI or file URI). The transformation specification
#will be executed in the context of the ni object, i.e., you can use
#ni member functions to access URI elements.


# the .well-known mapping
mountNi '"http://#{@host}/.well-known/ni/#{hashAlgo.to_s}/#{hashAsBase64 + queryPart}"'

# mapping to a local file system, using the content type parameter
mountNi '"file:///var/ni/media/#{contentType}/#{algo}/#{hashAsBase64}"'

# mapping to resource specified in loc URI query parameter:
mountNi '#{loc}'
```

Finally, `nid` supports regular expression based routing configuration. This allows the specification of next-hop node addresses (in conjunction with the Convergence Layer to use):

```
# Examples for routing table entry specification using the DSL
# specified in 'routedsl.rb'

# syntax:
# route add <regex>, <destination-uri>, <options>
# route add <string>, <destination-uri>, <options>
# route del <regex>, <destination-uri>

route add /^ni\:\/\/village\.n4c\.eu\/.*/, 'nihttp://village.n4c.eu', {:prio=>10}

route add 'ni://village.n4c.eu/sha-256;pbJzzNm2CZyRD5NhgXgiFPkT3G_O4NYOg5f1IFMg1Ag',
          'nihttp://village.n4c.eu', {:prio=>1, :redirect=>true}


route add /^ni\:\/\/ietf\.org\/.*/, 'nihttp://village.n4c.eu', {:prio=>1, :nocache=>true, :redirect=>true}

route add /^ni\:\/\/ietf.*/, 'nihttp://village.n4c.com', {:prio=>255, :nocache=>true}

route del 'ni://ietf.org/.*', 'nihttp://village.n4c.eu'
```

3.3 EwLC Integration and Demonstration

The Event with Large Crowd (EwLC) scenario has been chosen as a suitable scenario for demonstrating the benefits of NetInf over previous networking architectures. This section will describe how different partner prototypes fit together and are integrated to create a consistent NetInf system for the EwLC scenario, and then outline the plans for a final demo of this scenario at the end of the project.

The EwLC scenario targets situations when large crowds come together for a limited duration of time at some location due to a popular event occurring such as a sports event or outdoor festival. 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. When large crowds gather in a relatively small area during a relatively short period of time (on the order of tens of minutes to hours), this creates a very high load on the cellular network.

Common for all these scenarios is that they occur during events that gathers a large crowd interested in accessing data from the network. This 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 there for the same event, they can be expected to have similar interests that drive their data access patterns (e.g., at a football match, it is likely that most of the crowd want to view a replay of a goal). Thus, there are great potential for using NetInf in this scenario as NDOs can be cached close to users, 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.

3.3.1 Integration of Components in the EwLC Scenario

In this section, we will outline how some of the prototypes described earlier in this chapter can be integrated to create a system that can realize the EwLC scenario.

Figure 3.14 shows a conceptual overview of the EwLC scenario. The name resolution service is important as that keeps track of the NDOs that exist in the system, and can be consulted by clients in order to locate or get those objects. The NiProxy prototype described in Section 3.1.2 implements this functionality and using the NetInf protocol can communicate with other partner prototypes. The caching functionality in Figure 3.14 can be provided by NiProxy, but this functionality can also be provided through the NNRP prototype described in Section 3.1.1.

Mobile nodes (MNs) are the main end user terminals and the required functionality to create, publish, search, and retrieve content can be provided by the Android NetInf client described in Section 3.1.3 and the DTN based device described in Section 3.1.4. Due to differences in technology used for node-to-node communication, it is unlikely that the different mobile platforms will be able to do direct exchange of NDOs between each other, but they should be able to do so through intermediate caches.

As a connection between the two bottom "clouds" shown in the figure, and as a complement to the Android NetInf client, a content directory application being currently developed will also be included in the system to make it easier for users to visualise what content is available in the system and from where and when it originates.

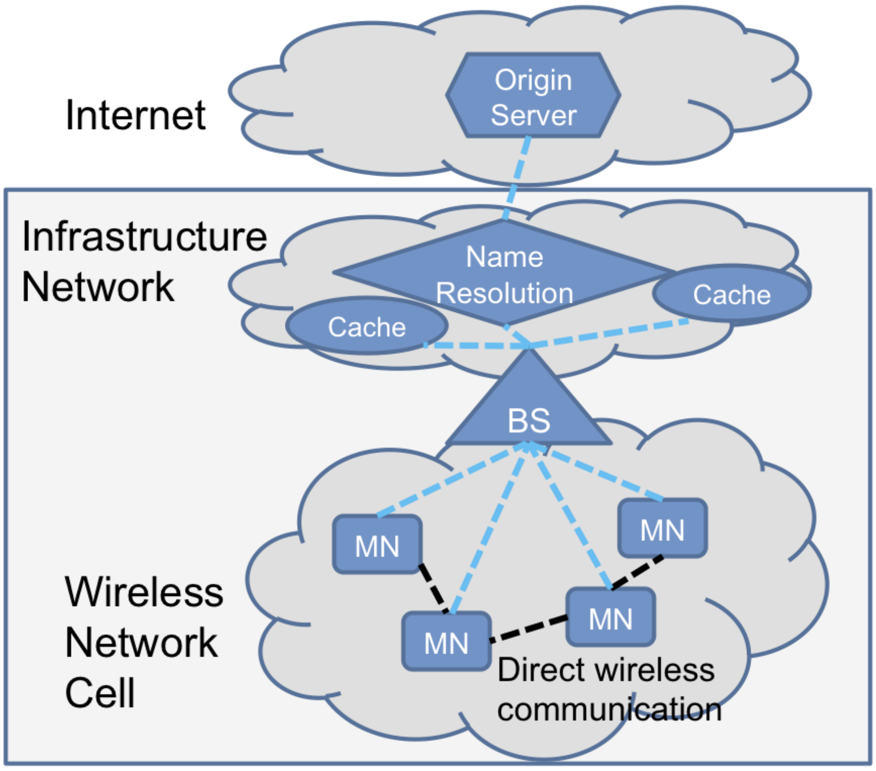


Figure 3.14: Interconnection of components and prototypes for the EwLC scenario

3.3.2 Final EwLC Demonstration Plans

The final EwLC demo will consist of two main demonstrator setups. To demonstrate the end user applications and show how the NetInf network would be used, a smaller network setup will be shown with partner prototypes running on physical nodes, including client software running on mobile phones. In order to further show the benefits of NetInf in a large-scale setting and to show the performance gains of NetInf over traditional Internet technologies in terms of offloading of the cellular network and latency gains for the user, the demo will also utilize virtualized nodes connected through a simulated underlying network. This allows greater flexibility and more possibilities for a thorough performance evaluation.

3.3.2.1 Physical node demo

The demo will show the EwLC scenario as outlined in Section 3.3.1 and visualized in Figure 3.14. The demo setup will consist of real physical devices running partner prototype code. Due to limitations in availability of devices and the difficulty in coordinating larger tests with a multitude of mobile nodes, this setup will be done on a limited scale and be a proof-of-concept demonstration.

Name resolution will be provided by the NiProxy prototype that will keep track of the NDOs published in the system. Mobile nodes that create a NDO can either only register it with the NiProxy so that NiProxy keeps track of the locator of that NDO, or it can publish and upload the NDO to the NiProxy so that the NDO is also cached there. Further caching can be provided by nodes running the NNRP prototype.

The main mobile nodes in the demo will be Android phones running the NetInf implementation described in Section 3.1.3. We intend to make this software publically available before the demonstration so that audience members that are interested can install the software on their own Android devices and use them to participate in the demo and use the NetInf network during the demo event to share content. The software on these nodes allow users to take photos and record video of interesting events and share with other users by publishing their NDOs at the NiProxy. When a user wants to get an NDO previously created, a request is sent to the NiProxy, which returns a list of locators where the NDO is available. The node checks its local wireless neighbourhood, and if any of the locators is reachable through WiFi Direct or Bluetooth, that wireless channel is used to retrieve the NDO locally. If not, it is retrieved using the 3G network from an infrastructure based cache or content server. To further demonstrate the benefits of this approach, we will show that even if central content server is not reachable, nodes will still be able to get the desired content.

For mobile users to be able to know what content to request, we will provide two possible methods to find available content in the network. The first methods consists of a web page that contains a listing of all available objects registered with the NiProxy, and the corresponding QR code (for easy access through the Android user application). In addition, we provide a content directory running on the Android devices that provide a map-based visualization of the available content over time and space in the intended stadium scenario. Using this application (preliminary version of the GUI shown in Figure 3.15), a user can scroll through the past timeline and visually see what nodes in certain locations (in different seat positions in the arena) have published NDOs at different points in time through a intuitive colour scheme. The user can then also get more information about the NDO and directly retrieve and display it from within the directory app.

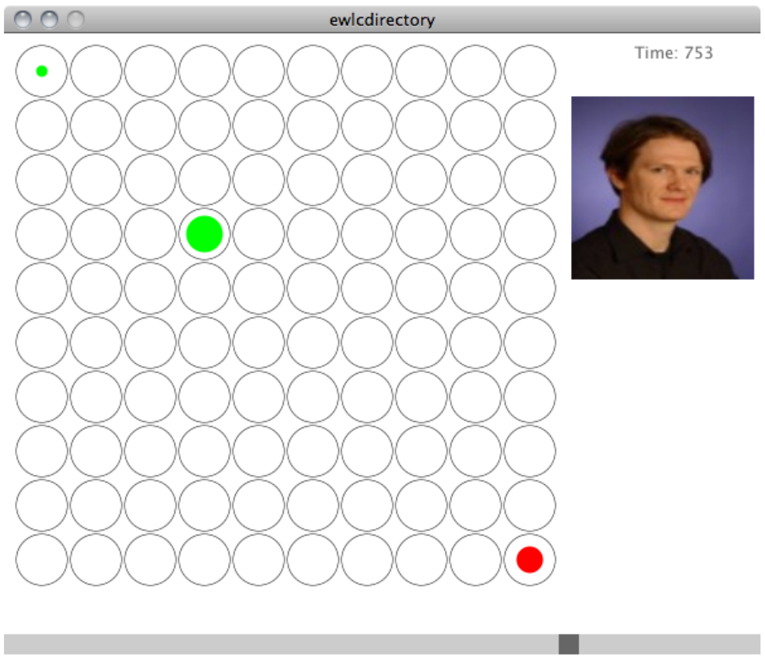


Figure 3.15: Map-based content directory for EwLC (preliminary GUI)

The required CLs for the DTN based prototype described in Section 3.1.4 will also be implemented in order to be able to use that as a second mobile user client device in this scenario for greater node diversity. Because of different device platforms used for the prototypes, the different types of mobile nodes will not be able perform local direct peer-to-peer exchange of NDOs, but will be able to communicate through intermediate routers and caches.

3.3.2.2 Emulator framework

In order to perform tests and evaluations that provide reliable and realistic results, it is desirable to perform those tests in the real target environment or something that is as close to that environment as possible. As described in the previous section, we will demonstrate the system on with real client nodes running in a deployed network. However, such demonstrations are limited in their scale due to the difficulty and cost involved in deploying hundreds of nodes and having users control them in real time. In addition, such deployments are also not very flexible as it is difficult to vary the number of nodes, network topology and conditions, and user behaviour is unpredictable. Therefore, we will also demonstrate the use of NetInf in a larger-scale EwLC setting through an ns-3 [43] based emulation framework. This utilizes functionality in the commonly used ns-3 network simulator that allows lightweight virtual machines (LXC Linux containers) to be connected to the simulated nodes. Thus it is possible to use the simulator to simulate the network topology, node mobility, wireless channel between nodes, and user behaviour, while still running the same software in the nodes as is used in the demo with physical node hardware. This ensures that we are showing a real system with working software, but also enables us to create networks that are larger than what would be possible with real hardware. The use of the emulator framework also enables us to run the same scenarios in a system where NetInf functionality is not present, thus giving us a baseline against which we can determine the benefits of a NetInf deployment to support an EwLC scenario.

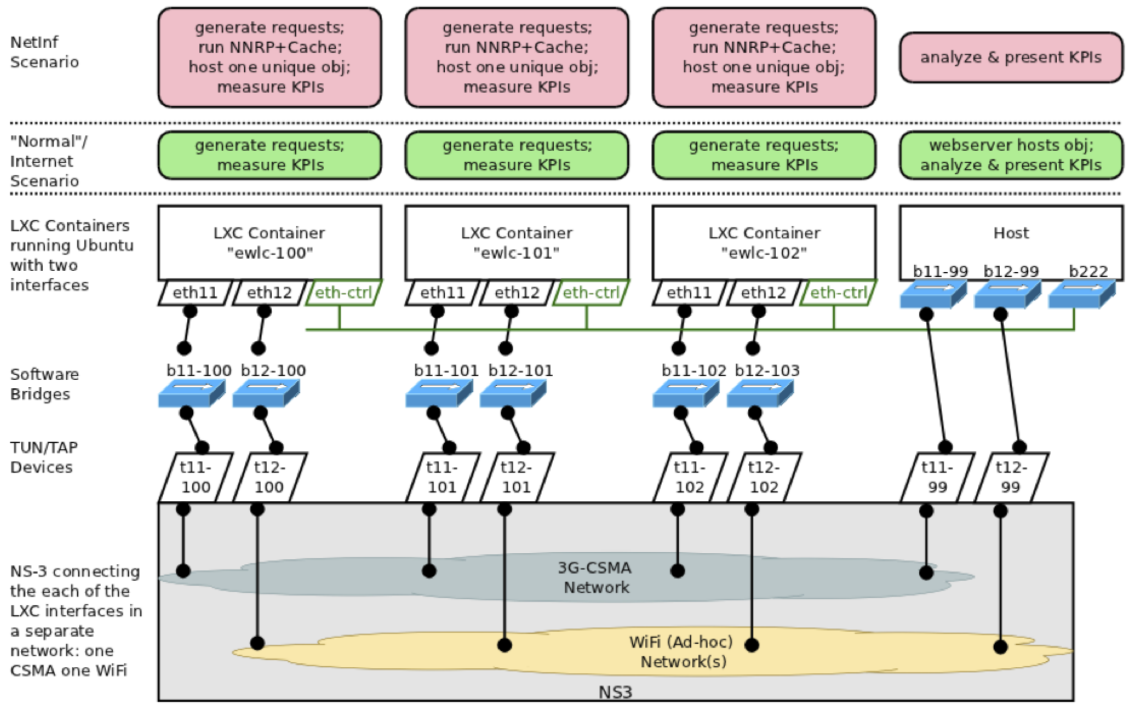


Figure 3.16: Basic network setup within ns-3 emulation framework

Figure 3.16 shows a basic EwLC network setup within the emulation framework. At the bottom of the figure an instance of ns-3 is shown running, providing the virtual nodes with a 3G network connection and an IEEE 802.11 channel that can be used for ad-hoc communication between nodes. This simulator instance is then connected to the LXC containers through the use of TUN/TAP devices on the host machine over which a network bridge is setup, which is in turn connected to virtual network interfaces in the LXC containers. This provides two network interfaces for each virtual node running an LXC container that the node can use to access the two simulated networks and communicate with each other. Within each LXC container, an instance of the relevant NetInf software is run. The required measurement processes to collect relevant Key Performance Indicators (KPIs) are also run in order to evaluate performance.

Evaluation scenario Our initial evaluation scenario based on the emulation framework is intended to evaluate the performance gain that the NetInf-based communication model can incur — compared to existing web-based communication models.

For that, we run the system with two different profiles: a "legacy" profile and a "NetInf" profile. In both cases the clients (nodes in the crowd) try to fetch random objects from a predefined list.

In the "legacy" profile, the objects are located on a centralised server and all clients request the objects over the 3G connection. This is a model of the current internet.

In the "NetInf" profile, each client holds one of the objects, and tries to fetch objects from neighbours through the Ad-Hoc WiFi link. If that fails, the object is fetched through the 3G connection. The object is in any case cached, and served to any neighbour who might request it. Figure 3.16 shows the demo setup. KPIs are collected on the nodes and sent through the control interface to a monitoring server where they are processed. The processed and aggregated values are then accessible through a web interface (see Figure 3.17).

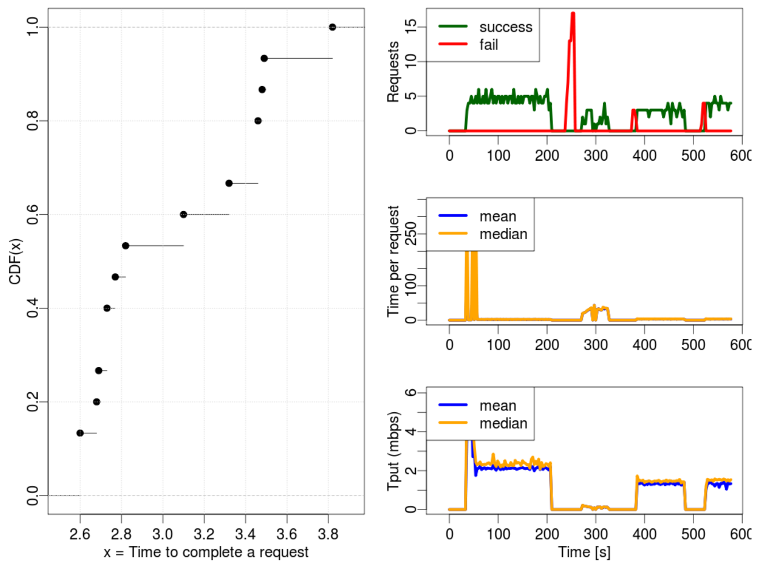


Figure 3.17: Visualisation of KPIs in the demo setup

In the continued evaluations, the NiProxy NRS system will also be integrated into the system. A set of scenarios with different node density, bandwidth on wireless and access networks, and user mobility and data access patterns will be run to evaluate the impact of the bandwidth offloading capabilities of NetInf from the cellular network and what this means for operators in terms of reduced resource usage and for user performance in terms of reduced latency. We will also demonstrate scenarios where the infrastructure is not functioning properly and has outages and show the potential for NetInf in enabling communication to continue through the local exchange of cached and locally published NDOs.

3.3.2.3 Potential connection of physical and emulated setups

The emulation framework provides the possibility to not only connect virtualized nodes to the simulated network, but to also provide a connection between the emulated and the physical world through the connection of external devices to the framework. This, for example, allows the use of NetInf user clients running on physical Android phones to interact with an emulated large-scale NetInf network, with NDOs being passed between the two environments. This would enable us to further investigate user experience in larger-scale networks. We will consider such integrations, but a thorough execution of the two separate demo settings will be given priority before considering a connection of the two.

3.4 GIN Prototype

A demo prototype of Global Information Network (GIN), an ICN proposal for the Network of Information described in [36], has been implemented during the SAIL project. GIN is a hybrid architecture able to support both dissemination and conversational communication models. It uses a stateless packet-based forwarding protocol, called Global Information Network Protocol (GINP), without employing any Convergence Layer. It aims to interconnect NDOs over heterogeneous L3/L2 sublayers, 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 initial GET requests to one or more destinations through a chain of Dictionary nodes integrated in the network infrastructure and arranged according to some hierarchical scheme embedding topological properties (e.g. content locality, locality of resolutions and routing paths), such as Multilevel DHT (MDHT) ([44]) or Hierarchical SkipNet (HSkip) (Section 2.4.1). Each object request initiates a direct communication session between the requesting entity and the object source(s). Data packets in the communication sessions are routed on the shortest path with fast lookups in the node Next Hop Tables (NHTs).

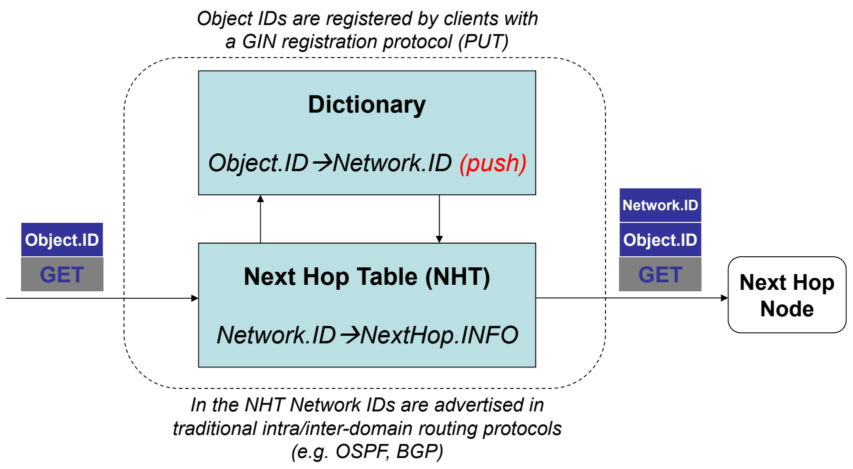

Object IDs are registered by clients with a GIN registration protocol (PUT)
Dictionary
Object.ID→Network.ID (push)
Object.ID
GET
Next Hop Table (NHT)
Network.ID→NextHop.INFO
Network.ID
Object.ID
GET
Next Hop Node
In the NHT Network IDs are advertised in traditional intra/inter-domain routing protocols (e.g. OSPF, BGP)

Figure 3.18: GIN name-based forwarding

Figure 3.18 illustrates the basics of GIN forwarding. When a GINP packet reaches a GIN node, the ID on top of the Destination ID Stack (DIS) is looked up in the NHT. The NHT is populated, by means of traditional intra and inter-domain routing protocols, with entries mapping identifiers of network entities (nodes and network domains) into next hop information. If a match is found, the packet is immediately forwarded to the next hop, according to the encapsulation of the existing underlying protocol. Such sublayers may include both L3 (e.g. IPv4, IPv6) and L2 (e.g. Ethernet) protocols. If, on the contrary, a match is not found in the NHT, the packet is sent to the node Dictionary subsystem for a resolution step. The node Dictionary is part of the network distributed Dictionary, which, in this prototype, is built according to the MDHT strategy. The Dictionary database is then populated by means of a registration protocol (PUT). The node Dictionary is searched to find one or more bindings mapping the top ID to a new network ID. If no bindings are found in the Dictionary, the MDHT table provides the network ID for the next Dictionary node in the resolution path. The new ID is pushed on top of DIS and the packet is passed again to the forwarding subsystem for a new lookup in the NHT.

The current GIN demo prototype provides the following features and services:

- name based routing and forwarding over heterogeneous networks (IPv4, IPv6, Ethernet);
- integrated name resolution by means of a distributed MDHT dictionary;
- in-network registration and storage of named objects;
- multicast ping of named objects;
- end-to-end receiver-based multipath retrieval of named objects.

Besides, a simple search engine provides support for search of data objects by keywords and names.

The GIN node prototype, see Figure 3.19, has been developed on a FreeBSD device and consists of two subsystems: the GIN Switch and the GIN Dictionary.

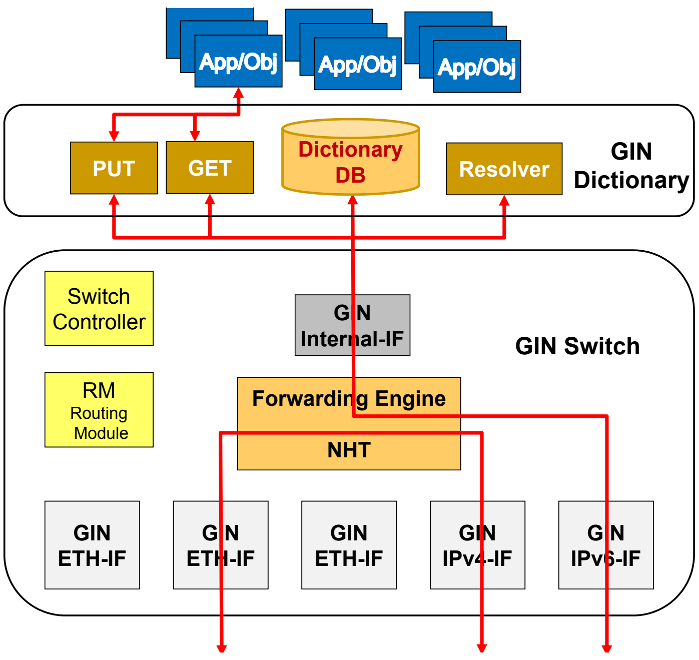


Figure 3.19: GIN prototype

The GIN Switch is implemented with a multithreaded program in C language. All modules shown in Figure 3.19 have been implemented, with the only exception of a dynamic Routing Module. In the current software release, a set of line commands is provided to manage and configure a GIN node. In particular, it is possible:

- to start/stop a GIN switch subsystem;
- to enable GINP over Ethernet Interfaces (GINP over IPv4/IPv6 interfaces is enabled by default if IPv4 or IPv6 interfaces are operating on the node);
- to add, delete, print GINP static routes;
- to configure one or more GIN node IDs;
- to configure a shaper on GIN output interfaces;
- to print or reset GIN interface counters;

- to send GIN echo requests to target IDs and receive echo replies;
- to generate GINP traffic at desired packet rate.

The GIN switch also provides an internal Interface towards upper applications and Dictionary modules. Such an interface provides a level of mux/demux name-based socket services, which allows multiple upper application processes to send and receive GINP packets to/from the GIN forwarding system.

The second subsystem of a GIN node is the GIN Dictionary. It is implemented in PHP language and runs over an Apache HTTP server providing proxy services. The GIN Dictionary is composed of a Dictionary database, containing network bindings for registered object IDs, and three main modules:

- The "Resolver" module handles resolutions for GINP packets which cannot be directly forwarded in the GIN Switch NHT; besides, it handles GIN echo requests (called gpings) for NDOs and manages the duplication of data requests or gping packets towards multiple destinations, implementing a sort of multicast service.
- The "PUT" module implements the registration protocol (client and server side). The registration protocol populates the distributed Dictionary with bindings according to the MDHT strategy and allows the storage/caching of published objects on the GIN nodes.
- The "GET" module implements the retrieval protocol (client and server side). The current implementation provides a reliable end-to-end transport service for the retrieval of named objects from multiple sources in parallel. The GET module uses the GIN echo protocol (gpings) to learn about the existence of available sources for a desired object. The gping capability provides a sort of resolution service, but gping replies effectively come from connected sources available to provide the desired object. The GET module then asks parts of the object to multiple sources in parallel, until the completion of the task. In the current implementation, the transmission rate is controlled by means of an AIMD congestion control algorithm per peer. Gping replies also carry relevant metadata for the requested object, including necessary information to check the integrity of the object. Gping requests can be sent periodically, in order to keep the list of available peers updated during large downloads and avoid to stop the download session if one or more remote peers leave the network. The current version of the GIN transport protocol for data objects is still under design and optimization. It differs from the other NetInf proposal (Section 3.1.1.4 since it is packet based, it adopts multiple congestion control loops at the receiver (one for each data source), it does not require chunks or object parts to be registered in the NRS and does not rely on Convergence Layers.

A network testbed of virtual GIN prototype nodes has been setup on a VMware platform, which provides access for GIN services to legacy IP clients from Internet, by means of a web server configured over each GIN node. In the testbed, several GIN nodes are interconnected through different sublayers (IPv4, IPv6 and Ethernet) and no IP connectivity is provided end-to-end. GINP packets flow seamlessly over different sublayers and provide the common network communication level. Client data objects can be stored in GIN access nodes, and objects can be retrieved and *gpinged* from the GIN access nodes. Current prototype software and documentation will be publicly released. Future work comprises several activities, including: implementation of a GIN client, demonstration of mobility and real-time traffic support, implementation of dynamic name-based routing and MDHT, addition of on-path chunk caching in the forwarding subsystem, evaluation of a possible Software Defined Networking (SDN) approach (with the GIN Switch implemented in the data plane and the GIN Dictionary in the control plane).

4 Evaluation

This chapter reports on results from NetInf evaluation activities based on the prototyping, emulation, simulation, and theoretical investigations reported in the previous chapters. Each of the evaluation activities contribute to one or a couple of the following general evaluation classes:

Throughput, latency and other performance metrics – measuring and characterising the performance of NetInf mechanisms and the overall system, including throughput and latency. The results of this kind of evaluation are contributing to showing the advantages of NetInf for the end-user, and to showing the feasibility of particular mechanisms.

Section 4.1 below largely covers the results of the performance-related evaluation activities, including the evaluation of HSkip lookup latency, GIN latency and throughput, the throughput, latency and fairness of NetInf transport flow control, and using NetInf for local group collaboration.

Scalability – simulating and analysing the scalability properties of central NetInf mechanisms, in particular name resolution and routing, to establish the feasibility of the NetInf system for a global scale network.

Section 4.2 reports on the results from the scalability analysis of GIN, NetInf routing with hints, and content search.

Event with Large Crowds (EwLC) project-wide scenario – showing several aspects of the feasibility and benefit of NetInf in a complex, project-wide, scenario that is difficult to realise with current technology. The results of this type of evaluation naturally has many facets, including performance, feasibility and efficiency, overlapping with other evaluation activities.

Section 4.3 presents the results of EwLC evaluation activities using emulation, simulation and an Android prototype implementation.

Increased efficiency and reduced costs – showing and estimating the increased efficiency of NetInf in terms of reduced need for network capacity, other resources and the resulting lower cost. The results of this kind of evaluation are largely contributing to showing the advantages of NetInf for network operators.

Sections 4.4 and 4.5 describe management and operational issues relating to network operators, and efficiency and cost gains based on analysis of real traffic data. Some of the results in Section 4.1, in particular from the local collaboration experiment, also contribute to demonstrating the efficiency gains possible with NetInf.

4.1 Performance Evaluation

This section reports on NetInf performance evaluation results as follows. Evaluation results based on analysis and simulation of the hierarchical name lookup system SkipNet are described below in Section 4.1.1. The GIN prototype has been evaluated with regard to response latency, see Section 4.1.2. Evaluation results for the receiver-based flow control protocol are described in Section 4.1.3. An experimental network for local collaboration with Subversion over NetInf has been built and evaluated as described in Section 4.1.4.

4.1.1 Evaluation Hierarchical SkipNet (HSkip)

This section describes some evaluation results for the hierarchical SkipNet (HSkip) name resolution system, described in detail in Section 2.4.1. We concentrate here on results for lookup latency and how it is influenced by the probability of being able to resolve a name at a given level (*level probability LP*), assuming we are looking at that particular level already. LP is a simple way of capturing locality of usage patterns; it is thus an external parameter, not a factor that a protocol can control. Other results (e.g., load balancing) can be found in a forthcoming paper.

4.1.1.1 Assumptions and notation

The resolution latency is dominated by the network latency. If resolution at level i fails, it is performed at the next higher level until a hit is found or the top level is reached. No network latency occurs at the AN level ($i = m + 1$) as it only consists of separate ANs. Hops *between* levels typically happen on the same physical node because each node typically participates in all levels. In some rare cases (but at *most* once per level), inter-level hops can happen between two separate nodes. However, these nodes are by definition close to each other with a latency of ≤ 0.5 ms based on our latency model. Hence, in both cases, inter-level hops do not add significant latency and can be neglected for real-world systems with a limited number of levels.
We define:

- m: Number of hierarchy levels (lowest level, nearest to end user, has number m)
- k: Nodes in the NRS form a k-ary tree of in total $n = k^m$ nodes
- L: Random variable, overall resolution latency
- L_1: Random variable, latency to find the answer
- L_2: Random variable, latency to return the answer
- Y_i: Event, object found exactly at level i
- $p_i = \mathrm{P}(Y_i)$
- $\overline{h}_i$: Avg. number of overlay hops at level i
- $\overline{l}_i$: Avg. overlay hop latency at level i
- $\overline{x}_i$: Avg. latency when object is found at level i

4.1.1.2 Analytic evaluation

Assuming independence of events Y_i between levels, we can derive p_i easily:

$$p_i = \mathrm{P}(Y_i) = \begin{cases} LP \cdot (1 - LP)^{m+1-i} & \text{for } 2 \leq i \leq m+1 \\ 1 - \sum_{j=2}^{m+1} p_j & \text{for } i = 1 \end{cases} \tag{4.1}$$

For HSkip, the average number of overlay hops at level i:

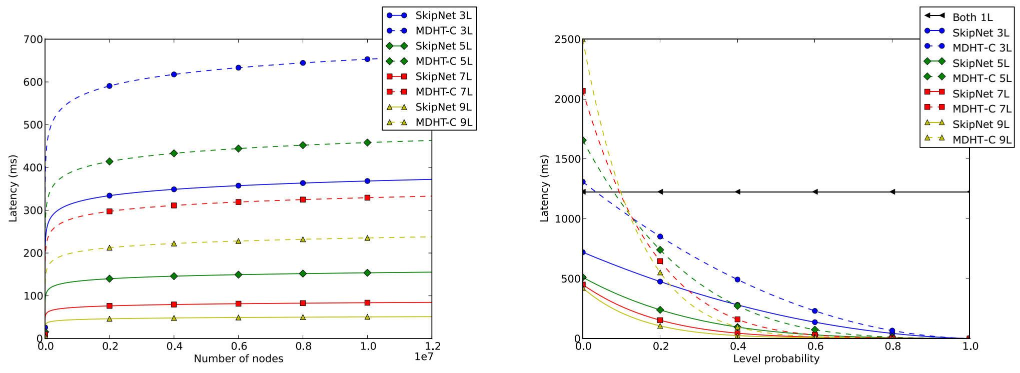
$$\overline{h}_{i_{\text{HSkip}}} = \log_2 k = \log_2 \left(n^{\frac{1}{m}}\right) = \frac{1}{m} \cdot \log_2 n \tag{4.2}$$

In the worst case, HSkip has to perform the numeric forwarding at each HSkip level. In this case, the expected number of overall required hops $\overline{h}_{W_{\text{HSkip}}}$ is given as:

$$\overline{h}_{W_{\text{HSkip}}} = \sum_{i=1}^{m} \frac{1}{m} \cdot \log_2 n = \log_2 n \in O(\log n) \tag{4.3}$$

Figure 4.1a illustrates the average latency for Level Probability (LP) = 0.3, for both MDHT and HSkip. Both MDHT and HSkip have lower latencies than a flat DHT system (not shown here for a better y-axis scale, ≈ 1200 ms with 12 million nodes), thanks to the effects of the level probability.

With 12 million nodes, 9 levels, and LP=0.3, MDHT has an average latency of ≈ 240 ms and HSkip has an average latency of ≈ 50 ms.



(a) Analysis: Latency of MDHT ($O(\log n)$ DHTs) and HSkip; 3–9 levels (L); LP=0.3; dashed=MDHT, solid=HSkip

(b) Analysis: Latency MDHT ($O(\log n)$ DHTs), HSkip; 1–9 levels (L); 12 mio. nodes; dashed=MDHT, solid=HSkip

Figure 4.1: Analytic comparison of MDHT and HSkip

Figure 4.1b also illustrates the strong influence of the Level Probability on the overall latency. Both MDHT's and HSkip's latency is significantly reduced with an increasing LP.

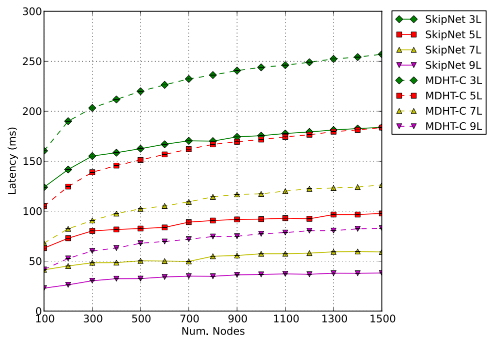
4.1.1.3 Simulation-based evaluation: Latency

In addition to the numeric illustrations of the analytic equations derived above, we also implemented a simulation of HSkip and MDHT, based on OMNeT++ and Oversim. The setup uses up to 1500 nodes NRS nodes and 4500 client nodes; each NRS node registers on average 300 objects; objects are updated with exponentially distributed inter-update dates according to web models [45]; cached entries have an exponentially distributed lifetime as well, akin to P2P networks [46]; object popularity follows a Zipf distribution; network latencies at the various levels are computed according to Equation (4.4), which gives a good match for latency in overlay networks based on P2P network analyses [47]; maximum latency at the highest node is set to 500 ms.

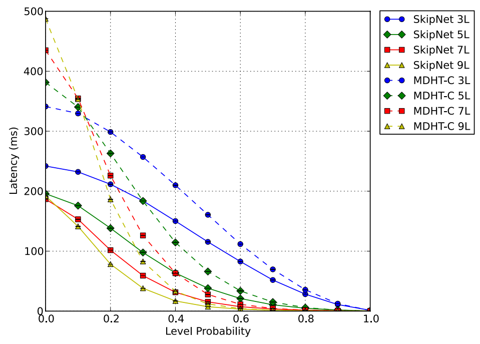
$$\bar{l}_i = \sqrt[m]{\bar{l}_{\max}{}^{m+1-i}} \tag{4.4}$$

For brevity, we concentrate here only on latency results, contrasting HSkip with our earlier MDHT system. Figure 4.2a illustrates the influence of the number of nodes and the number of levels on the average MDHT and HSkip latency with LP=0.3. Unsurprisingly, the latency increases with an increasing number of NRS nodes as more NRS nodes result in more required hops in both MDHT and HSkip. The increase is sub-linear thanks to the $O(\log n)$ characteristics of both Chord (which is underlying the MDHT system) and Skipnet (which is underlying the HSkip system). More importantly, for both MDHT and HSkip, the latency is *decreasing* with an increasing total number of levels, confirming the previous analysis results. The HSkip latency is generally lower than the *independent* MDHT latency for the same number of levels and nodes. With 9 levels, 1500 nodes, and LP=0.3, the independent MDHT and HSkip have a latency of approximately 85 ms and 38 ms.

Figure 4.2b confirms the strong influence of the Level Probability on latency. The latency decreases significantly for both MDHT and HSkip with increasing LP. In our DNS measurements [13], we have found the Level Probability to be in the range of 0.4–0.7, which would, e.g., result in average latencies of 10–65 ms for the independent MDHT and 10–30 ms for HSkip for a system with 7 levels and 1500 NRS nodes.



(a) Simulation: Latency of MDHT ($O(\log n)$ DHTs) and HSkip; 3–9 levels (L); LP=0.3; dashed=MDHT, solid=HSkip.



(b) Simulation: Latency MDHT ($O(\log n)$), HSkip: 1500 nodes; 3–9 levels (L); dashed=MDHT, solid=HSkip.

Figure 4.2: Simulation-based comparison of MDHT and HSkip

4.1.2 GIN Latency and Throughput Analysis

The performance of GIN/MDHT have been analyzed in detail in the TI technical report describing the GIN approach [36]. Here, main results on GIN latency and throughput are summarized from that report.

A GIN network has been modeled with the transition diagram illustrated in Figure 4.3. Formulas for throughput and latency have been derived from it. The diagram models a scenario in which the first request packet of a GIN data flow goes through a resolution path (in the MDHT hierarchy), while the following data packets are fast forwarded on the shortest paths (with only NHT lookups). Throughput and latency strongly depend on the locality characteristics of traffic and on the usage and architecture of the cache system. In particular, the main parameter is the *network locality* Λ, defined as the probability of an object request to be resolved and satisfied internally to a given provider network. The network locality Λ depends on the probabilities α_i that a data request can be resolved from a local source/cache, taking into account that each MDHT level i can have its locality characteristics and its own cache system.

For the evaluation of GIN, we have considered the global network modeled as a MDHT/REX[1] intra-domain network composed of three internal levels (Access Node, Point of Presence (POP), AS) plus a fourth external level for a remote AS, representing the rest of Internet. We have also assumed the AS level to be REX-enabled, that is, with a copy of all REX entries cached in the DHT at the top level (see also Section 4.2.2.2). Another hypothesis has been that all intra-domain DHTs have full finger tables, so that DHTs only need one hop at all levels. Such assumption is realistic in the context of a GIN provider network, with a limited number of infrastructure nodes (N.B. MDHT does not run on GIN client devices).

	Average	Comments
Packets/flow	18	
Bytes/Packet	600	Including 100B (20%) GIN header overhead
Flows/sec per Mbit	13.9	
Client Throughput	1.2 Mbit/s	10 times traffic per user today in peak hour
Bytes/Object	8460	This is a rough approximation; may change considerably in future ICN

Table 4.1: GIN traffic statistics (hypothesis)

[1]Resolution EXchange (REX)

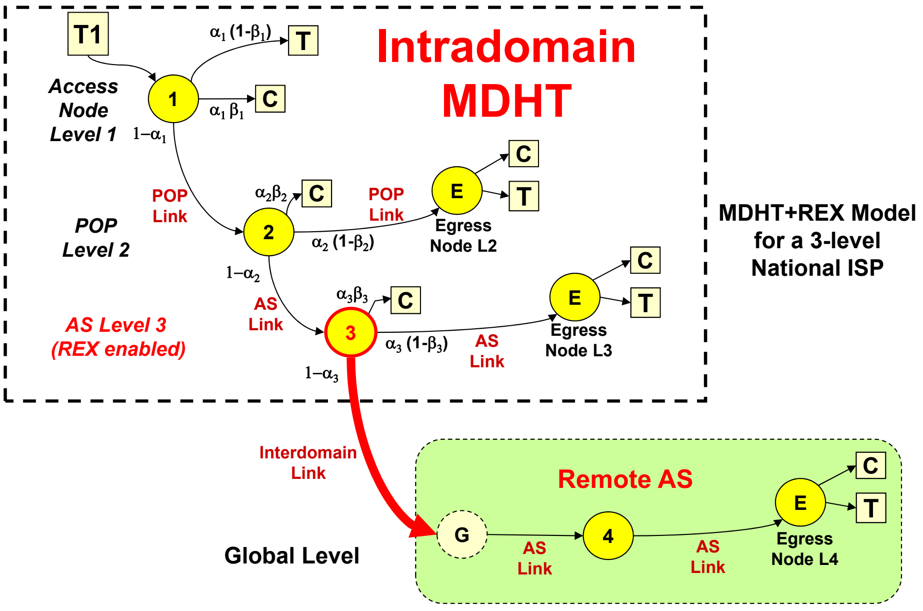


Figure 4.3: Transition graph of a global MDHT/REX network

In the analysis, we have also assumed GIN traffic to be analogous to IP traffic. There is no evidence for that, but GIN is designed to support not only data distribution according to the ICN model, but also any other common and legacy point-to-point communications (e.g. voice, interactive data, etc.). Of course, it is likely that ICN will introduce some variations in the way applications communicate and distribute data, but, for the moment, the only reasonable hypothesis is to assume the same traffic profiles of IP. Therefore, recent (mid 2011) IP traffic statistics have been considered, collected by CAIDA at the Equinix datacenter in Chicago from a backbone 10 Gigabit Ethernet link of a Tier1 ISP[2]. From those IP statistics, we have derived a set of statistics for the future GIN traffic, reported in Table 4.1.

In the following, the main results of the latency and throughput analysis in [36] are reported briefly:

- As shown in Figure 4.4, the rate of resolution requests (i.e. object requests) in a GIN/MDHT node with 100K clients and 10 times the traffic per user with respect to today (i.e. 1.2 Mbit/s in peak hour per user for the future ICN instead of current $\approx$ 120 Kbit/s) is less than 3.5 Mpps for any value of network locality Λ (i.e. amount of on-net traffic), and falls below 2 Mpps for $\Lambda \geq 0.8$. In this scenario, a node with $\Lambda = 0.8$ and 100K clients should be able to forward up to 125 Gbit/sec. Such numbers are compatible with the technologies discussed in Section 4.2.2 for the implementation of both the NHT and the Dictionary.

- In GIN, the MDHT/REX system is used to route control packets, mainly resolution requests, to the final destination(s) where the requested objects can be found. The MDHT routing causes however the routing paths to stretch with respect to the shortest data paths. In order to evaluate the effect of such routing stretch over the resolution traffic, we have calculated a *MDHT Stretch Factor (MSF)*, which is the average number of resolution nodes crossed by a resolution request before reaching the target destination. The MSF is a function of the network locality Λ and the number of MDHT levels. As shown in Figure 4.5, for a certain network locality, a higher number of MDHT levels determines a larger MSF, that is, a higher

[2]http://www.caida.org/data/monitors/passive-equinix-chicago.xml

rate of resolution requests per MDHT node. So the number of MDHT levels must be kept as low as possible.

- GIN adopts asymmetric shortest data paths like in current Internet. Only resolution requests (first packet of a communication flow) are sent over the resolution path, according to the MDHT/REX strategy. In theory, MDHT could be used for all traffic, providing a routing system with symmetric data paths. Unfortunately, adopting such routing with symmetric paths would show an unacceptable overhead (271% with $\Lambda = 0.8$ in a 3-level MDHT network, see Figure 4.5) with respect to current IP networks, due to the high MSF. In addition, the latency over MDHT symmetric data paths would be $\approx$ 30% more than that of GIN over fast data path. On the contrary, with GIN data shortcuts over shortest fast paths, the extra overhead becomes acceptable (close to only $\approx$ 5%) because the MDHT routing stretch applies only to the first packet of a data flow. The results suggest then that symmetric paths can be acceptable only if data are sent over routes with stretch 1 or very close to 1! But no name-independent routing with such routing stretch is known [48]. The best known approach [49] works only with static networks and shows an average routing stretch around 1.4 on Internet topologies. The choice of GIN to route data over shortest paths with the only resolution packets routed over the stretched MDHT paths seems to be a good option.

- Latency of data and resolution packets decreases almost linearly with the network locality Λ (Figure 4.6). With inter-domain links of 10000 km on average, latency is below 25 msec for all types of GIN packets (resolution and data) with a network locality $\Lambda \geq 0.8$, and below 60 msec with $\Lambda \geq 0.5$. Latency of resolution traffic (i.e. first GET packet of each bidirectional flow) sent through the MDHT/REX infrastructure is only 15% more than latency of data packets routed on shortest Internet paths.

- Latency analysis shows that, in most cases, 2-3 intra-domain MDHT levels may be enough for small to large provider networks. Besides, REX is more efficient than any global DHT for the resolution traffic latency (Figure 4.7). Even in the ideal case in which a Global DHT is able to perform routing in only one hop, resolution traffic latency of a global network with REX is about 75% of latency with a global *one-hop* DHT.This is due to the fact that, in GIN, REX entries are announced and cached in each AS with global reach. Therefore, it is possible to do global resolutions directly inside the AS where the request is issued. So, there is no need for any resolution step into the global network, as required by even an optimal Global DHT: the request can be sent directly to the destination AS from the requester's AS (or from its upstream provider).

In conclusion, the performance analysis shows that, although the GIN end-to-end latency between two arbitrary points is equivalent to that in the current Internet, the increase in the network locality due to the ICN caching is associated to a proportional reduction of the average latency. Besides, results analyzed in Section 4.2.2 confirm that the amount of signaling (resolution and registration requests) and data traffic per node can be easily handled by current switching and storage technologies, even when GIN nodes have to manage traffic from a high number of clients (100K clients or more). The only drawback is the increased overhead of the GIN system with respect to the Internet Protocol. In particular,the larger GIN packet header produces an overhead between 15-20% with respect to that of IP, which is $\approx$ 8% with the current Internet traffic profiles. The extra overhead of $\approx$ 5% caused by the MDHT routing stretch is also to be taken into account, in place of the current DNS overhead. Finally, we need to point out that the increased ICN overhead should be compensated by the increased availability and locality of data. In that regard, the analysis of the MDHT-based collaborative caching system illustrated in [5] shows that the network locality can significantly increase with ICN. More detailed results on throughput, latency and caching performance of GIN/MDHT can be found in [36].

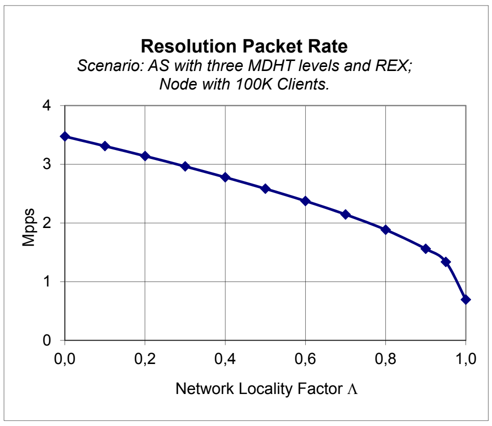


Figure 4.4: GIN/MDHT resolution packet rate

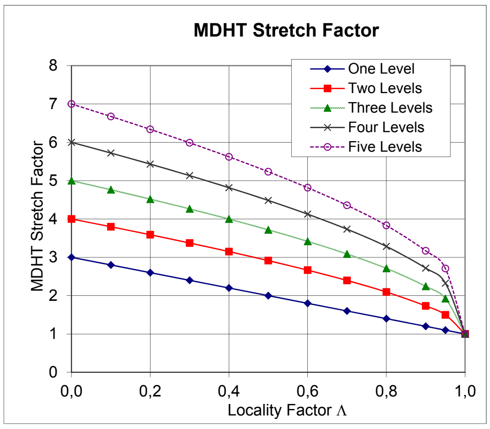


Figure 4.5: MDHT stretch factor

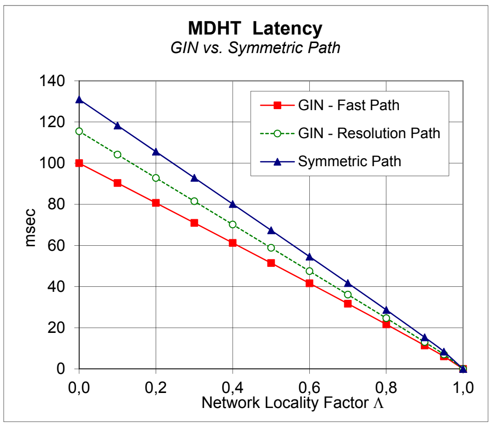


Figure 4.6: GIN/MDHT packet latency

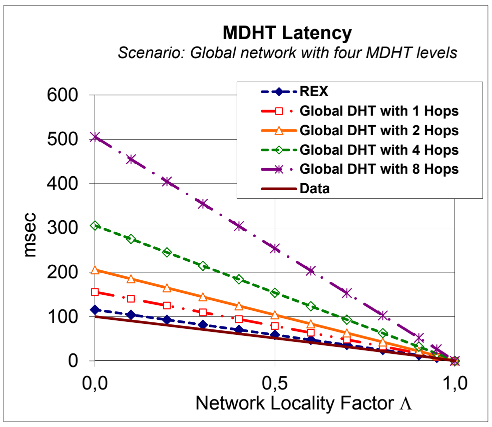


Figure 4.7: REX vs. global DHT

4.1.3 NetInf Flow Control: A Throughput, Latency and Fairness Evaluation

We here provide an evaluation of the NetInf receiver-based flow-control protocol that is described in Section 3.1.1.4. We first focus on the performance in terms of throughput and latency of the module, using a small testbed and the NNRP prototype. This experimentation is performed both in presence of a single source and of several sources.

Using Omnet++ simulations, we further investigate the fair sharing behaviour of the protocol in different network conditions sharing the resources with simultaneous TCP flows.

Testbed setup The evaluation was conducted on a testbed consisting of 5 laptops, connected by an 100 Mbps hub as shown in Figure 4.8. Each laptop was running NNRP. The client (on the left of the figure) was requesting through a regular web-browser a 30 MB image, divided into 500 chunks. This request was translated by the NNRP HTTP Client Interface module in a regular NetInf request. The servers (on the right) stored the chunks and the whole image and answered the request. As the evaluation was focused on the flow control module, routing was statically determined, and no NRS was used.

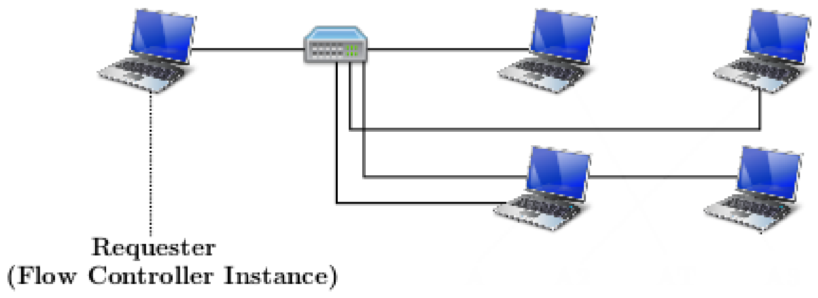


Figure 4.8: Testbed physical configuration

Single connection A brief comparison to TCP is made using one client and one server hosting instances of the NNRP application. For TCP, raw data transmission between two devices was observed. In order to reproduce more realistic networking scenario, we either added an artificial RTT delay between the client and the hub up to 200 μs, or decreased the capacity of the link between the server and the hub, down to 10 Mbps. Although we observe in Figure 4.9 a gap in performance from an artificially imposed delay of 100 milliseconds onwards, the module is well comparable to TCP in other cases.

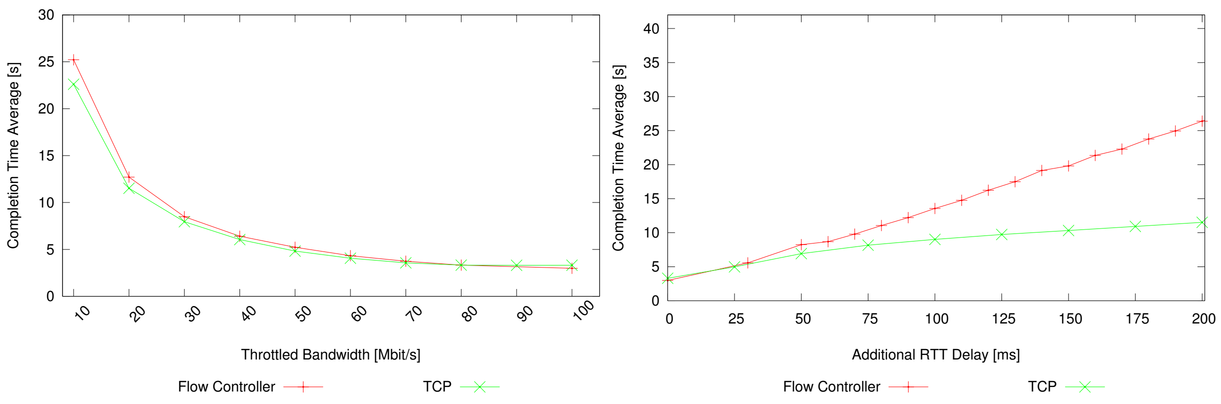


Figure 4.9: Transfer time for a flow as function of the link bandwidth (left) or latency (right)

Multi-source case The use of a receiver-based flow control protocol and the ICN security model allows to easily request different NDOs in parallel from several sources. To this aim, we implemented a basic module, which was forwarding the NetInf GET request to a set of servers, alternating in a round-robin manner. Note that this multi-source download is here native, and transparent for the application: it is controlled only by the routing of GET requests.

Using the same topology as the previous scenarios, two to three server nodes were used for the following tests. Our objective is to observe benefits of object partitioning: multiple source nodes contain fragments of a main object, from which recipients demand them in no particular order.

Recall that on one side, the link between the client and the hub is an Ethernet link which runs at 100 Mpbs. On the other side, the links between the hub and each server are artificially limited in bandwidth, down to a limitation to 10 Mpbs. They hence are the bottleneck of our setup. By using multiple sources, we hope to see an increase in performance as recipient nodes may use all the source nodes independently to assemble a single object.

These tests are based on a single object, demanded from multiple source nodes. When we take a look on Figure 4.10 at completion time values over both bandwidth and RTT variations, we may conclude that multiple sources also follow the same evolution as a single node. More importantly, the results are even more satisfying in two ways :

- Using the same total of bandwidth distributed between multiple source nodes, Flow Controller is able to catch up with TCP as we divide the available bandwidth to accommodate for new source nodes.
- Flow Controller scales out well by using additional source nodes: the most apparent example is the one with a 10 Mbps bandwidth. We begin with a single node capable of serving the content in approximately 25 seconds. A second node cuts it down to around 13 seconds, whilst a third node reduces it further to 8 to 9 seconds. This is almost a linear speedup over the number of nodes.

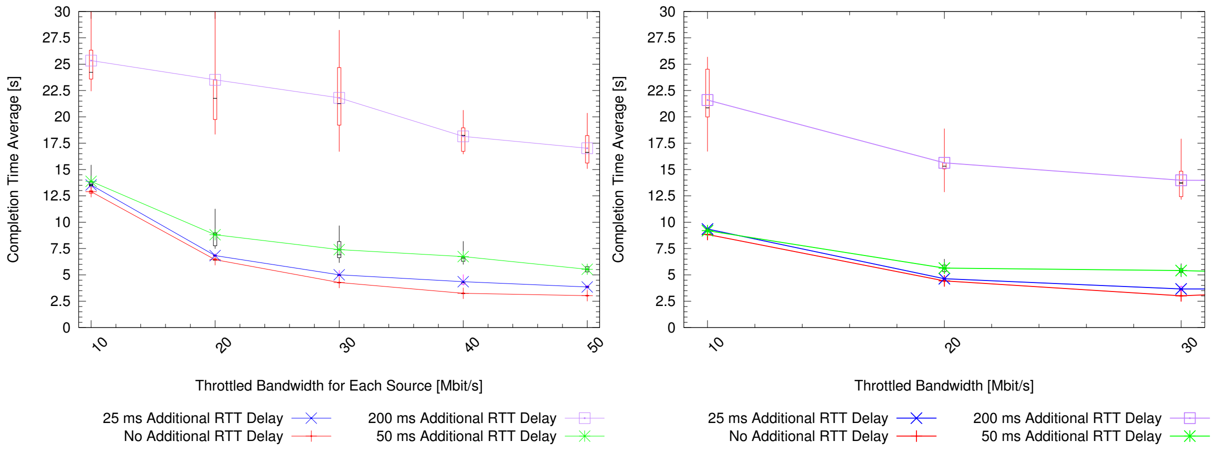


Figure 4.10: 2-sources (left) or 3-sources (right) transfer time

Fairness analysis We now investigate the fair sharing behaviour of the flow control protocol in different network conditions sharing the resources with simultaneous TCP flows. The point of this section is to show that the receiver driven transport protocol can coexist with existing transport protocols, which is predominately TCP on today's Internet. We analyse three different cases with the same topology based on the used TCP variants, which are TCP Reno, TCP Reno + SACK, TCP New Reno. The following network was used for the prototype testing. A bottleneck link of 100 Mb/s capacity a 2ms link delay a 32 frame buffer length, the source and receiver links are 1 Gb/s with small delay (0.4μs), the total data length is 30 MB. The following results are from the simulation with two TCP New Reno and two NetInf simultaneous flows, starting them at the same time.

Figure 4.14 shows that the two TCP flows still could only start slightly later, but mostly we can see the fair sharing of the 100 Mb/sec link. All flows are consuming around 25 Mb/sec overall, which is full resource utilizing for the four simultaneous flows. The NetInf flow control protocol still happens to be slightly more aggressive, which is due to the implemented retransmission strategy. In Figure 4.12 we can see that the two TCP flows start more aggressive, but later reduce their congestion window and all the flows are taking turns equally in terms of outstanding bytes. Figure 4.13 shows that the slopes of received bytes in time for each flows are parallel, which also represents

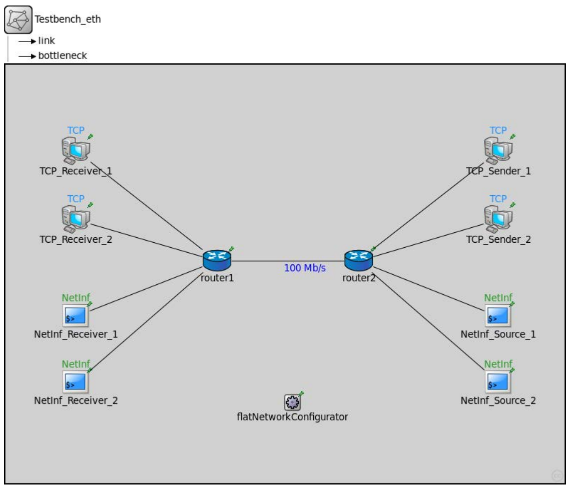


Figure 4.11: Test-bench topology

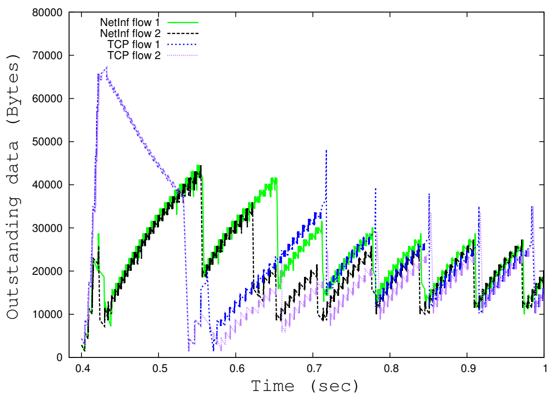


Figure 4.12: Outstanding bytes

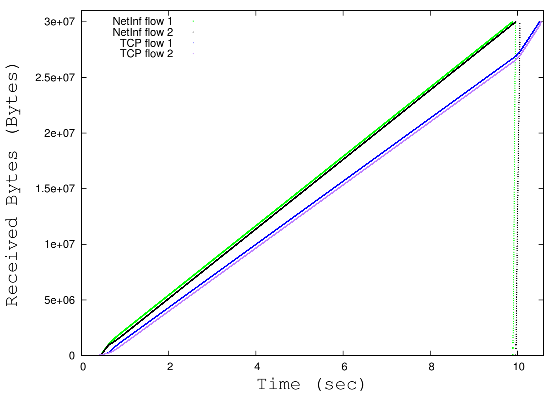


Figure 4.13: Received bytes

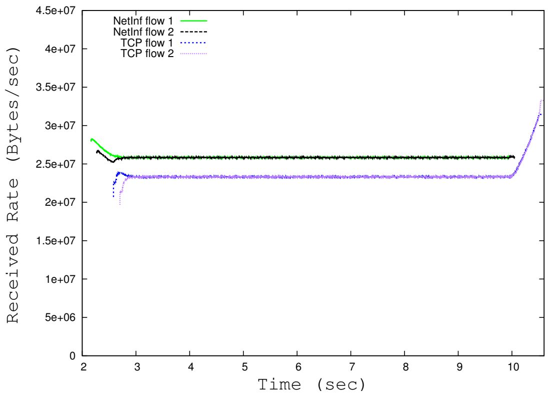


Figure 4.14: Received rate

the fair bandwidth sharing behaviour. The NetInf flow control retransmissions are taken place at the end, after that the two TCP flows speeds up.

The results in case of TCP Reno and Reno + SACK flows are showing less fair sharing behaviors, and the results are fairly sensitive for initializing parameters. Generally the TCP flows having difficulties to start up, and grab bandwidth efficiently compared to NetInf flows. In case of New Reno, the fair sharing results are much better.

Conclusion and future work This evaluation is still preliminary, but it shows that

- NNRP, using the NetInf flow control module compares in terms of performance to the current HTTP over TCP solution in some simple cases; this means that NNRP is able to fully utilize the bandwidth in these cases;
- NNRP and the flow control module still experience a performance gap when compared to TCP in presence of high RTTs;
- when the use of several sources allows to circumvent the bottleneck of the network, NetInf outperforms the current uni-source architecture, by being able to fully utilize the network capacity;
- the proposed flow control protocol shares the bandwidth between competing flows in a fair manner; it also is compatible with TCP, in the sense that both TCP and NetInf flows get a fair bandwidth when they share a link.

Our research agenda includes the two following items, in order to improve this evaluation:

- **Extension to multi-hop cases**, where the benefits of receiver-based flow control over a TCP layer without object fragmentation should be visible;
- **Analysis of the performance gap** between the current flow control implementation and TCP in presence of significant RTT. A possible reason is the fact that our module was ran in the NNRP prototype, hence at the application layer, whereas TCP tests were conducted using the classical kernel-level TCP stack. Another possibility is that we evaluate here an early implementation the protocol, which is compared to a version of TCP which has been optimized during years, and the difference might just come from code optimization. Finally, we do not rule out that allowing out-of-order delivery, hence removing the fast-recover and fast-retransmit phases, impacts the performance of the protocol in this settings.

4.1.4 Local Collaboration with Subversion over NetInf

In this section we summarise the experience and insights from an experiment showing the benefit of using an information-centric network service for local collaboration [50]. The purpose with this evaluation is to show that ICN is beneficial also for applications that are *not* disseminating content in a large scale, and in particular, for applications that can take advantage of local caching to lessen the need for good connectivity to a server.

The Subversion [51] version control system was adapted to using the service of OpenNetInf [38], and the resulting system was experimentally evaluated. The scenario was described in more detail in project deliverable *D-3.1 (D-B.1) The Network of Information: Architecture and Applications* [52].

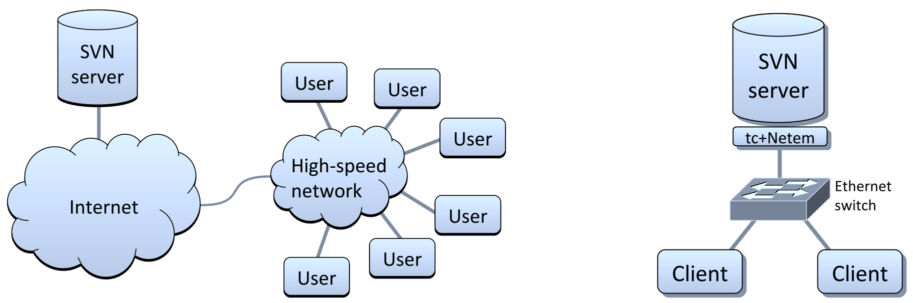


Figure 4.15: Local collaboration using Subversion: scenario (left), experimental setup (right)

4.1.4.1 Local collaboration scenario

The local collaboration scenario stems from our own project work where we at physical project meetings collaborate on documents and program code using a Subversion (svn) version control system. The users have good local network connectivity, but the network path to the svn server has limited capacity and can be located far away, as illustrated in the left half of Figure 4.15.

When a user makes a document or other file available to the others, the document is first uploaded to the server, and then downloaded multiple times, once per other user, back from the server. This is wasteful on the limited network capacity to the server and can often lead to performance problems, something that we have experienced in our own work. In an information-centric network, in contrast, the network service enables retrieval of a copy of the document from a neighbor client or from a nearby cache, circumventing the need for all clients to download from the server.

The right half of Figure 4.15 illustrates the experimental setup that we used to emulate the scenario. There are two clients connected via a switch to a server. The wide area path is emulated in Linux using the kernel traffic control (tc) mechanism together with the Netem queuing discipline.

4.1.4.2 Adapting Subversion to OpenNetInf

Subversion uses a centralised *repository* for sharing information. Clients can check out a copy of the repository, make changes to that copy, and then commit the changes back to the repository, creating a new version. Other clients can then update their copies to the new version, or to any other version in the past.

We adapted a Subversion implementation to use the network service of OpenNetInf in order to enable Subversion clients to checkout and update their local repository copies from any available source, whether that is the Subversion server, another client, or an in-network cache. This adaptation of the file transfer parts of Subversion was straightforward. It showed that the NetInf network service was a good match to the needs of this application.

4.1.4.3 Experimental results

In this section we present some of the results from the experiments with the modified Subversion using OpenNetInf for file transfer. The results show that local copies of data are retrieved automatically by the ICN network service without application involvement. The absolute performance numbers should however not be taken too seriously, as this is a far from optimised research prototype.

Figure 4.16 shows the CPU and network utilisation at the Subversion server when the first client checks out a repository using the OpenNetInf communication service. The repository contained six 10 MB files. During the first 20 seconds, the server makes the requested files available through the OpenNetInf service. This includes calculating cryptographic signatures and registration in the

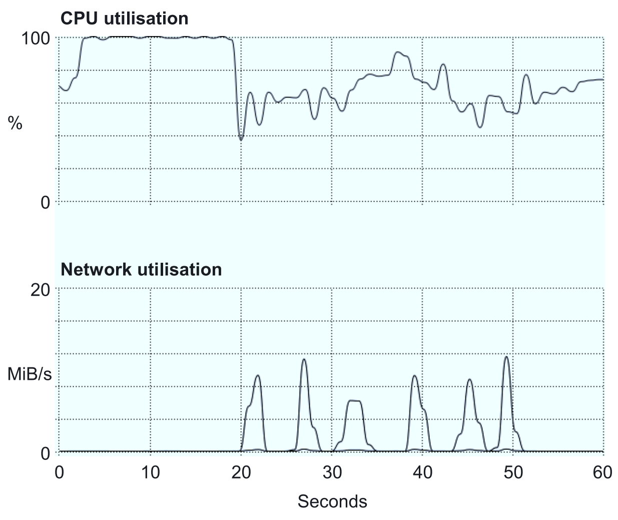


Figure 4.16: OpenNetInf CPU and network utilisation

	Legacy		**OpenNetInf**	
	mean	stdev	mean	stdev
100 Mbit/s, 5 ms delay				
1st client	19.8	0.53	49.6	0.81
2nd client	=	=	53.0	2.08
5 Mbit/s, 50 ms delay				
1st client	147	0.24	191	2.15
2nd client	=	=	53.8	1.11
1 Mbit/s, 100 ms delay				
1st client	726	0.45	770	2.61
2nd client	=	=	56.1	2.69

Table 4.2: Measured performance of checkout in seconds

OpenNetInf name resolution service for each file. During seconds 20-60 six files are transferred with good throughput to the client with short delays in between when the client verifies each signature.

Table 4.2 shows the measured performance of a checkout operation using Subversion over HTTP (legacy) and over OpenNetInf for three settings (capacity and delay) of the emulated wide area link. The first client fetches from the server, and for the OpenNetInf case, the second client fetches from the first client. The values are means for five runs. As expected, the second client is in practice unaffected by the setting of the wide area link.

Figure 4.17 shows the same data as the table for Subversion over OpenNetInf relative to the legacy Subversion over HTTP. The data for the third client and above are extrapolated as explained in the next section. Values below one (1) on the y-axis mean lower performance than legacy Subversion, and values above mean higher performance. As the 100 Mbit/s – 5 ms curves show, there is no benefit to run Subversion over OpenNetInf in an environment with unrestricted network capacity to the server as the overhead then is greater than the transmission savings.

4.1.4.4 Evaluation issues and conclusions

The experimental setup only included two clients. The results for three and more clients are extrapolated using the following formula for the time t it takes for n clients, where t_1 and t_2

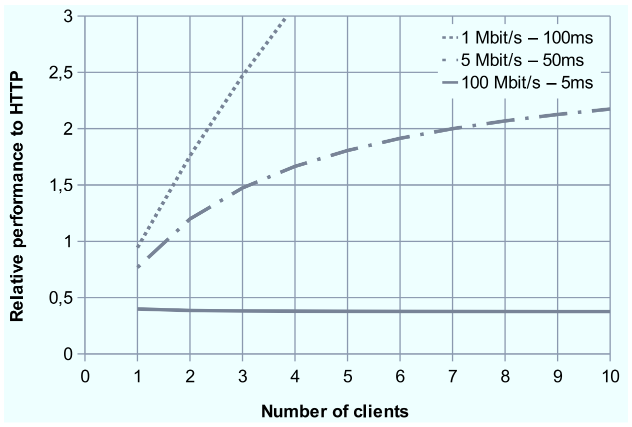


Figure 4.17: Subversion/OpenNetInf performance relative Subversion/HTTP

are the measured time for the first and second clients respectively:

$$t_n = t_1 + (n - 1) \times t_2 \tag{4.5}$$

This formula is a simplification. Parallelism is, for example, ignored. The clients are assumed to download the files serially after each other. For the legacy (non-ICN) case, there is limited amount of parallelism that can be exploited, since all clients download from the same server. For the ICN case, there is potentially much more parallelism, since the more clients, the more sources that can provide data. This formula is thus conservative, that is, is underestimating the expected performance of the ICN case.

It is clear that OpenNetInf has a higher base overhead than current network protocols, but the experiments revealed that the gain quickly offsets the costs. The measurements show a performance advantage for ICN already with two clients for realistic simulations of the WAN connection to the server, despite being a non-optimised research prototype.

While ICN can make communication more efficient for the same content distribution scenarios that motivate the use of CDNs, the local collaboration scenario in this experiment and similar scenarios that approach DTN provide additional motivation for ICN.

4.2 Scalability Evaluation

Scalability is the key issue of any novel Internet technology intended to be deployed on top of or replacing IP. The challenge of NetInf is to enable global access to NDOs published to any NetInf access router operated by a NetInf operator anywhere around the world. The global access should be available within a short period of time after publishing the NDO. The key dimension of scalability for a NetInf network are in terms of size and update rate. The size of a NetInf network is taken into account by the number of NetInf nodes (routers, caches, NRS nodes), the number of NDOs, the number of customers and related request rate, and the number of NetInf operators or domains. The update rate is taken into account in terms of adding/removing NetInf nodes and in terms of the publishing rate including publishing and unpublishing NDOs, modifying NDOs e.g. their meta-data, and adding/removing locators/routing hints. The contributions to scalability are structured as follows: In 4.2.1, a general view on scalability aspects of NetInf separated into the local and the global components are given. In Section 4.2.2 and Section 4.2.3 a detailed evaluation of the routing scalability of GIN and NetInf in general are given. Finally, the scalability of a content search mechanism is evaluated in Section 4.2.4.

4.2.1 NetInf Scalability

In a NetInf network we distinguish the scalability of a local (intra-domain) NetInf network under the control of a single operator and the global (inter-domain) NetInf network. The key scalability issues of the local network are in terms of the number of NetInf routers, caches, and local NDOs including the related update rates. The key scalability issues of the global network are related to the number of domains and required locators/routing hints also including the related update rates.

4.2.1.1 Local scalability

In the following the different aspects of scalability in a local single-operator NetInf deployment are analyzed:

- **Number of NetInf routers:** The number of NetInf routers scales the system horizontally. Critical aspects when increasing the number of nodes are stability of the system (success rate), robustness of the system (against node failures), system performance (content delivery delay, publishing delay), and coordination overhead (number and volume of signaling messages per node).
- **Number of NDOs:** The number of NDOs will primarily increase the number of routing table entries in the NetInf routers. The system scales with an increasing number of NDOs by increasing the number of NetInf routers and including a hierarchical component in the organized naming of locally published NDOs or a local NRS as proposed in GIN. However, this potentially introduces an increased effort for managing local name spaces that needs to scale with the number of NDOs. Critical system parameters are the system performance (content delivery delay) and in particular the node performance (forwarding delay). Another critical aspect is how the caching scales with the number of NDOs with respect to both the number of hit rates and caching efficiency as well as with the complexity of caching strategies. The criteria to observe for caching are in particular the cache hit rate and the caching efficiency.
- **Request rate for local NDOs:** The local request rate reflects the number of customers and the number and popularity of local NDOs. The NetInf routers and also caches need to scale with the number of requests. This can be achieved by increasing the capacity of NetInf components or by adding more NetInf components. Adding more NetInf routers leads to the scalability issues discussed above. Critical system parameters are the system performance (content delivery delay) and node performance (request forwarding delay). NetInf caches need to scale with the request rate in terms of lookup, delivery and replacement of cached content.
- **Request rate for global NDOs:** Requests for Global NDOs are resolved by the global NRS. The system scales by increasing the capacity or the number of local NRS nodes participating in the global NRS and sharing the load to identify routing hints or location global NDOs. The global NRS is organized in a way that scales with the number of elements so adding more local NRS nodes reduces the load per node in the global NRS. However, that does not necessarily mean that the load of a single load would then be equally shared among all local nodes. Instead, the load per local node with respect to global name resolution depends on the overall number of nodes building the global NRS. For a further discussion of the scalability of the global NRS, see also Section 4.2.1.2. Critical system parameters are the success rate and time for global NRS lookup.
- **Publish rate:** The publishing and also the modification of NDOs are the potentially largest challenge for NetInf. While a majority of published objects may be stable in the sense that they remain available for a long time and also keep their location, there may also be a significant

part of more dynamic NDOs that are available for a short time only or change their location frequently, e.g., if the source is moving. A NetInf network scales vertically with respect to the publishing rate by adding nodes with higher performance. Forwarding based NetInf does not scale with the publishing rate since adding more nodes does not reduce but rather increase the load of a single node. Concepts like a local NRS or hierarchical local naming schemes, however, may lead to an improved scalability. Critical system parameters to observe with an increased publishing rate are: stability of the system (local success rate, false delivery rate), performance of the system (publishing delay, content delivery delay), and coordination overhead (number and volume of signaling messages).

The scalability should be measured in a prototype with increasing number of nodes by observing the following KPIs (see Section 4.4): publishing delay, content delivery delay, local success rate, forwarding delay, false delivery rate. The impact of node failures or connectivity loss in particular with an increasing number of NetInf nodes needs to be considered as well. The EwLC scenario is deliberately chosen to investigate the most critical aspects of scalability, i.e., the impact of the publish rate and instable NetInf nodes.

4.2.1.2 Global scalability

In the following the different aspects of scalability in a global NetInf deployment are analyzed:

- **Number of Global NRS Nodes:** The number of nodes participating in the global NRS system is the means to balance the load of a single node. On the other hand, the overhead for coordinating and managing the global NRS needs to scale with the number of nodes. The global NRS needs to scale with the number of domains since the future role of a NetInf operator is difficult to foresee and consequently also the number of NetInf domains may be considerably higher that the number of autonomous systems in today's Internet.

- **Number of Domains:** The number of NetInf domains to be reachable via the global NRS defines the (minimum) number of entries in the global NRS.

- **Number of global requests:** Resolving of global requests is not achieved by a lookup in a single node but typically by a kind of DHT (mDHT, SkipNet) such that all nodes participating in the NRS share the load. Increasing the number of nodes naturally reduces the load for every single node though the load does not decrease linearly and a certain overhead for routing requests to the responsible node must be taken into account.

4.2.2 GIN Feasibility and Scalability

In this section, the feasibility of the GIN proposal [36] is analyzed in terms of scalability and performance of its main components, i.e. the Next Hop Table (NHT) and the Dictionary (see Figure 3.18). In particular, the analysis is focused on lookup times and memory sizes required by the forwarding function (Section 4.2.2.1) and by the resolution function (Section 4.2.2.2).

4.2.2.1 The Next Hop Table

The NHT contains forwarding entries for networkIDs (nids) of network domains, nodes and hosts. In the current proposal, nids are fixed length identifiers chosen independently from the topology, and are obtained by means of simple hashing from the object Master (authority or owner of the object) and user-level name. Then, the lookup operation is an exact match. Other choices for the GIN ID scheme, like the IP addressing scheme, may need different lookup rules, i.e. longest prefix match.

In each GIN domain, at least one intra-domain routing protocol should be run, in which infrastructure nodes are advertised by name (i.e. nid). Traditional intra-domain routing protocols, link state or distance vector, can be used. For example, routing protocols like RIP, EIGRP, OSPF, IS-IS could be modified to support such behavior. The NHT can then be filled with next hop entries for each nid advertised in the intra-domain network. A BGP-like protocol, or another enhanced inter-domain protocol, can be used to advertise nids of external ASes in the Internet. Such IDs could be derived from the current AS numbers or other topology independent names. The NHT may then contain one or more next hop entries towards any external Internet AS learnt via BGP. Finally, each GIN access node should keep in its NHT the next hop entries for the attached clients (hosts, devices and client networks). The nids of connected entities can be learnt by means of some local routing/discovery/registration protocol or by static configuration.

The size of the NHT can be estimated in the order of $10^5 - 10^6$ entries, as the sum of the number of intra-domain infrastructure nodes (usually in the order of 10^5 in the largest networks), the number of Internet ASes (currently in the order of 10^4 and in the future 10^5) and the number of connected devices, which depends on the node dimensioning and could be estimated in the order of 10^5, as for current Internet access nodes. Aggregation techniques can be studied to further reduce the NHT size, if required. Such dimensions are perfectly manageable and compatible with the available technologies for switching and routing tables in modern telco carrier-grade equipment. CAM, DRAM or SRAM technologies are obvious candidates for a hardware implementation of a switching/forwarding function at wire speed in current routers and switches. Therefore, such technologies are also plausible candidates for a hardware implementation of the GIN NHT.

In general, in a forwarding system a CAM returns the memory address of a Networking Dynamic Random Access Memory (DRAM) or Static Random Access Memory (SRAM) block where the associated forwarding entry (i.e. next hop) is stored. Therefore, a lookup in a CAM-based system is the sum of a CAM lookup plus a DRAM or SRAM access time. Table 4.3 provides state of the art performance limits, in terms of memory capacity, cost and access speed, in 2010 [53].

Technology	Single Chip Capacity	Cost per Chip	Access Speed
DRAM	128 MB	\$0.08-\$0.16	40-80ns
SRAM	9 MB	\$5.5-\$7.8	3-4ns
TCAM	4.5 MB	\$44.5-\$66.7	4-8ns

Table 4.3: Capacity, cost and access speed of networking memories (2010)

According to such figures, if we assume a NHT implementation based on CAM+DRAM, the NHT lookup time could be estimated conservatively around 50-100 nanosec, which means a single NHT forwarding engine can handle a packet rate of 10-20 Mpps. Current high-end IP routers can already manage more than 1 Million entries in their forwarding tables. With 128 bytes entries, a chipset with a single DRAM 128MB and 1-2 CAM chips can provide a NHT with 1 million entries. If needed, it is possible to combine more memory chips in parallel for larger lookup tables, with a very low degrade in rate performance. Therefore, the current router and switching technologies can easily cope with the expected size of a GIN NHT.

4.2.2.2 The Dictionary

The GIN Dictionary is the distributed database infrastructure which implements the Name Resolution System in the GIN network, according to the MDHT strategy [44]. Each GIN node may be equipped with a local instance of the GIN Dictionary. The Dictionary maps target IDs into next hop node IDs (or local paths for locally cached objects). The lookup operation in the Dictionary should be quite fast, in order to reduce the latency of a connection setup on the resolution path.

Besides, the size of a node Dictionary is also considerable, taking into account that it has to store all binding records of the information published by the attached clients.

In an intra-domain Dictionary based on MDHT, all NDOs can be registered at all levels, but the amount of Dictionary Records (DRECs) for each MDHT level is constant. In fact, thanks to its nested arrangement, MDHT guarantees that the ratio of DRECs per node is the same at all levels. Consider for example a network with three levels: Access Node, POP and AS, with N nodes per M POPs. If at the first level, an Access Node holds on average K DRECs, at the POP level there are $K * N$ DRECs distributed over N nodes, which still means K DRECs per node at POP level, on average. In the same way, at the AS level, $K * N * M$ DRECs will be distributed over $N * M$ nodes, with K DRECs per node at that level. As a consequence, since the MDHT infrastructure for an ISP network can be modeled as a hierarchical tree with L levels and N leaf nodes and all data objects can be registered at all levels, the total amount of DRECs per node is $L * K$, where L is in the order of the logarithm of the number of nodes and K is the average number of DRECs registered from clients at each access node.

Now it is necessary to dimension the memory of a Dictionary system in a GIN domain. Let us consider, as an example, a GIN domain with 3 DHT Levels and 10^9 DRECs of 1 KB each per node per DHT level, which means 3×10^9 DRECs and 3 TB per node. Each DREC is associated to an object ID and can contain related metadata and one or more bindings. In that case, assuming that each node handles a number of attached clients in the order of 10^5, the GIN system could directly address up to 10^{15} information objects and 10^{11} hosts in the Internet with only 10^6 GIN access nodes. As shown by the throughput performance analysis of GIN MDHT (see Section 4.1.2 and [36] for further details), a number of 10^5 clients per node is a realistic assumption.

In REX enabled networks, also REX entries should be cached in the Dictionary, at the AS level. The REX maps an authority name into the networkID of the Autonomous System where it is possible to find the authoritative NRS for that authority [36][44]. Tier-3 networks can store only a subset of REX, e.g. the authority names of peering and client networks. Transit providers should cache all bindings registered in the REX. Currently (October 2012), there are about 910M hosts advertised in DNS[3], 620M web sites[4] and 143M web domains in the 6 most popular gTLDs (.COM, .NET, .ORG, .INFO, .BIZ, and .US) [5]. Then, it is possible to assume, conservatively for the mid-term future, a number of REX entries in the order of 10^{10}. Since REX only maps authority GIN IDs (e.g. 8-12 bytes) to AS GIN IDs (e.g. 8-16 bytes)[6] which host their primary resolution server, REX entries can be quite short. Assuming, for example, 100 bytes on average per REX entry, 10^{10} REX entries can be cached in only 1 TB. Such an amount of storage can be further distributed on all GIN nodes of the same provider participating to the top MDHT level (AS level).

Available storage technologies for the Dictionary implementation are of two types: Solid State Disk (SSD) or Hard Disk Drive (HDD). Compared to SSD, HDD is a much slower technology due to its high access latency, usually larger than several milliseconds. Assuming a target lookup time for a GET request into a node Dictionary of less 1 msec, the candidate storage technologies for the Dictionary implementation are clearly Flash SSDs and DRAM SSDs, for their fast access speeds.

Flash SSD is a cheap technology, with access times below 200 microsec and read/write operations even greater than 4M IOPS[7]. Note that, from the throughput analysis reported in Section 4.1.2, an upper bound of 3.5 millions of resolution requests per second has been found for a GIN node with 100K users and 10 times the traffic load per user of today. Such an upper bound of the resolution request rate is found when the network locality factor is zero (i.e. all requests need to be resolved off-net), while decreases as the ratio of on-net traffic raises to one (Figure 4.4). Flash SSD is then

[3]http://ftp.isc.org/www/survey/reports/current/
[4]http://news.netcraft.com/archives/category/web-server-survey/
[5]http://www.whois.sc/internet-statistics/
[6]GIN ID formats are described in [36]
[7]e.g. http://www.nextio.com/show.php?page=products_vstor

a good choice if the lookup operation needs only one access to the storage in most cases. In that case, a Dictionary DB could be implemented with a hash table. The drawback of such a technique is that hash tables need a larger storage space to reduce the collision probability. A load factor of 0.75 is frequently used in those cases, which, in our scenario with 10^9 object DRECs (3 TB) and 10^{10} REX entries (1 TB or less if distributed) means an incremental storage of 1.3 TB per node (5.3 TB in total per node). Another option is to use DRAM SSD, with access latency between 3 and 15 microsec and read/write operations up to 12M IOPS[8]. Also the rack space required to host a storage server for a node Dictionary can be quite small[9].

4.2.3 Scalability of NetInf Global Routing with Routing Hints

NetInf global routing with routing hints, described above in Section 3.1.1.5, and in more detail in Deliverable D-3.2 *(D-B.2) NetInf Content Delivery and Operations* [5, Section 2.3], makes use of *aggregation* of routing information of two kinds in order to make the routing system scalable. First, the NDOs sharing the same name authority part or scope parameter are mapped to the same set of routing hints. Second, only the lowest priority routing hints need to be globally routable.

In the following, we analyse the scalability of the system by discussing involved namespaces, the number of items in each, and the relations between the namespaces. Recall Figure 3.7 on Page 37 illustrating the mapping from ni: name to routing hints.

Named data objects We estimate that the system needs to be able to handle at least 10^{15} NDOs. They are all named with the ni: naming scheme.

ni: name authority part and scope parameter The ni: name authority part and scope parameters are in essence DNS names. The number of domains in DNS is, as mentioned in the previous section, estimated to 143 million for some of the most popular gTLDs.

A publisher is expected to name its published NDOs with identical authority parts or use identical scope parameters for objects that form a natural group, or *aggregate*. Examples are pages on a web site, photographs in a collection, and parts of a movie. The number of such specific name authority parts/scopes is not expected to affect the number of domains allocated per organisation at the second (or sometimes third) level in the hierarchical DNS name space, since the former names are allocated as leafs under the latter domains. Adding more leaf names in DNS does not create a scalability problem, since the resolution for those names are highly distributed.

From a routing perspective, the assumption is that all NDOs in an aggregate can be routed the same, and thus share routing information. Furthermore, the cost of resolving the authority part/ scope to a set of routing hints can be taken once for the first NDO of the aggregate, and then cached and reused for all the other NDOs. This is quite similar to how DNS works for the web. The DNS name of the web server is resolved once into an IP address, which then is cached and reused for all further transactions with that web server.

Routing hints Routing hints look like IP addresses, and just like IP addresses they are used to make forwarding decisions in routers. The main difference is that a routing hint in a request does not name the destination for that request. It is only used by routers to look up the next hop to forward a request to.

Routing hints are matched exactly in the forwarding table — there is no longest prefix match. At first glance, this seems to result in a scalability problem. But the longest prefix matching (CIDR) of

[8]e.g. http://www.kove.com
[9]As an example, Texas Ramsan710 (http://www.ramsan.com/products/rackmount-flash-storage/ramsan-710) accommodates 5TB memory in only one rack unit.

Network	IP prefix
SICS local network	`193.10.64.0/22`
KTH network	`130.237.0.0/18`
SUNET aggregate network	`130.237.0.0/16`

Table 4.4: IP network prefixes for SICS' Internet access

IP is replaced with the explicit aggregation that comes from resolving an authority name part/scope into *multiple* routing hints with priorities. Only the hints with lowest priority need to be announced in the global routing system. These lowest priority hints are expected to represent a whole operator, one organisation, a site, or similar quite large network with many routers and hosts.

We argue that we will need a lower number of these lowest priority hints than we have IP prefixes in the current Internet's global routing table, currently about 500K. The number of lowest priority routing hints depends on the size and topology of the network, rather than on the number of NDOs. The introduction of NetInf is unlikely to radically change the size and link topology of the network as such, in addition to the natural growth and change, so there is good reason that the current set of IP prefixes can be replaced with the same number of routing hints. The conclusion is that BGP can be used to provide global routing for the routing hints equally well as it does for IP today. In addition, we only need to announce the lowest priority hints in BGP, resulting in a smaller global routing table compared to today. The routing for the higher priority hints is then provided locally within the larger networks that the lower priority hints represent.

As a concrete example, consider the current network topology and the IP prefixes relating to the Internet access for SICS listed in Table 4.4. The SICS local network is connected via the KTH network to SUNET, the Swedish university network. The latter peers with many other operators, and thus needs to announce the IP prefixes of all its customers to the rest of the world in BGP. That includes the first[10] and last of the prefixes in the table—the second is covered by the last using normal CIDR rules, and thus do not need to be announced by itself, nor need its own routing table entry. We assume that we will need routing hints corresponding to the three prefixes in the table for NDOs available from SICS, with the hint for the SICS local network having highest priority and the hint for the SUNET aggregate network having the lowest priority. It is enough to announce this lowest priority hint in the global BGP routing system for the hints, and only provide local routing for the other two. As it is safe to assume that there are many more similar edge network cases, we conclude that it is likely that the global routing table for the hints can be *smaller* than the current BGP table for IP addresses.

Summary and conclusion In the analysis we have described and discussed how the expected 10^{15} individual NDOs are aggregated using some hundred million DNS domain names that in turn, via DNS itself or external resolvers, map to less than 500K routing hints. Using BGP to provide routing for these hints are as feasible as for IP today.

4.2.4 Enabling a Metric Space for Content Search in NetInf

With the ubiquitous and pervasive presence of all types of content, one of the main challenges in ICN scalability is information usability: how to provide an efficient, scalable and accurate ability for users to locate information objects matching users' interest from multiple sources or caching nodes in the network. The objective is to put receivers (or content providers) in touch with senders

[10]This is in reality a simplification. The SICS prefix is part of a larger address block that with normal CIDR rules only needs one entry in the global routing table, so the announcement is shared with other prefixes. Despite this, the argument still holds, but the benefit is less.

(or users/consumers) on good terms, to distribute, index, search and retrieve information in a large scale and highly dynamic networking environment. We propose an extension of the Name Resolution Service to provide an efficient and scalable content search within the network layer, as a network built-in process and not as an overlay or external application. The complexity of search is decentralized and shared out over the Name Resolver Systems within the same processes for advertising and requesting content.

Basically, a search engine mines content available in the network and summarizes the semantic properties of any discovered content into descriptors. When a content search is requested, the descriptors extracted from the query meta-info are compared to the descriptors of the search engine's index using a similarity distance. The search function returns then a list of information objects which match the query object. Search engines should be sublinearly scalable with growth in network size and leading to a low query traffic volume and a high query success rate. For ICN, we use IO metadata as descriptors for disseminated content in the network, which then allows us to integrate content indexing and searching in the network layer by associating similar patterns between advertised metadata and search queries.

The basic design of our proposal for content search consists in considering information objects as a massive set of points in a d-dimensional Euclidean space with a distance to measure similarity of objects (d can be considered as the maximum number of metadata attributes in any IO). It is convenient to think of IOs as n points in $S = \mathbb{R}^d$ or as an $n \times d$ matrix, with n very large. This high-dimensional $n \times d$ point set when performing indexing or similarity searching for query objects is naturally challenged by scalability and computational complexity which often depends exponentially in the dimension. For comparison, Google has to maintain millions of crawlers for indexing the Web, but a large volume of resources are still beyond this exploration (only a small fraction of the pages available on Internet are actually indexed). To overcome these problems, we use a probabilistic embedding technique to reduce the dimensionality of the search space (which is in the order of $n \times d$) by following the lemma of Johnson-Lindenstrauss, which asserts that any set of points in a high-dimensional space can be embedded into a space of much lower dimension without suffering great distorsion, i.e., in such a way that all pairwise distances between the points are nearly maintained.

Lemma 1 (Johnson and Lindenstrauss) *Given $\varepsilon > 0$ and any set $\mathcal{P}$ of n points in $S = \mathbb{R}^d$, there exists a function $f : \mathbb{R}^d \to \mathbb{R}^k$ such that the point set can be projected in a subspace of dimension $k = O(\frac{\log n}{\varepsilon^2})$ that preserves the distances between all pairs of points within the small factor ε, i.e:*

$$\forall u, v \in \mathcal{P},\ (1-\varepsilon)d(u,v) \le d(f(u), f(v)) \le (1+\varepsilon)d(u,v)$$

The function f in the J.L lemma is usually a hash function, so that the lemma guarantees that the search space can be embedded into a lower dimensional space, which is logarithmic in the number of IOs available in the network and independent of the number of attributes used to describe those IOs. We propose to use distance-sensitive hashing based on p-stable distributions to construct this embedded space. Let us first recall the definition of stable distributions.

A distribution $\mathcal{D}$ over $\mathbb{R}$ is p-stable ($0 < p \le 2$) if for any n real numbers $v_1, \dots, v_n$ and i.i.d random variables $X_1, \dots, X_n$ with distribution $\mathcal{D}$, the variable $\sum_i v_i X_i$ has the same distribution as the variable $\|\overrightarrow{v}\|_p.X = (\sum_i |v_i|^p)^{1/p}.X$, where X is a variable with distribution $\mathcal{D}$. In particular, p-stable distribution for $p = 1$ is Cauchy, and for $p = 2$, it is Gaussian. [54] shows how we can generate stable random variables mainly from two independent variables distributed uniformly over $[0, 1]$.

Consider now $\overrightarrow{a} = (a_1, \dots, a_n)$ n i.i.d. random variables with a p-stable distribution. Given two information objects $u, v \in S = \mathbb{R}^d$ we can then assert by linearity that the difference between scalar products $\overrightarrow{a}.\overrightarrow{u} - \overrightarrow{a}.\overrightarrow{v}$ is distributed as $\|\overrightarrow{u} - \overrightarrow{v}\|_p.X$ where X is a p-stable distributed variable.

The gap between the two projections of $\overrightarrow{u}$ and $\overrightarrow{v}$ (onto the vector $\overrightarrow{a}$) depends then on the distance $\|u-v\|_p$: it is more than likely that these projections are close if $\|u-v\|_p$ is small, i.e. if the objects u and v are close (and vice versa). Therefore, the projection as a scalar product with $\overrightarrow{a}$ is then a good choice as a basis to define the distance-sensitive embedding function as mentioned in the Johnson and Lindenstrauss. Let us divide the real line into buckets of length w and we number these resulting buckets by $0, 1, 2, \ldots$. As dot products range over the real line, each projection of one object u onto the vector $\overrightarrow{a}$ will fall into one unique bucket. Two objects which are close (i.e. similar) will very likely be projected into the same bucket. The Figure 4.18 gives a geometric representation of the distance-sensitive projection onto the vector $\overrightarrow{a}$: the projection of the query object q onto the vector $\overrightarrow{a}$ (resulting in the value qa) has a high probability to be located into the same bucket as the projection of the object u_1. Let b be a real drawn uniformly in $]0, w[$ to assert the collision probability of hashing. The distance-sensitive hash function is then defined as:

$$h_{a,b}(u) = \lfloor \frac{\overrightarrow{a}.\overrightarrow{u} + b}{w} \rfloor$$

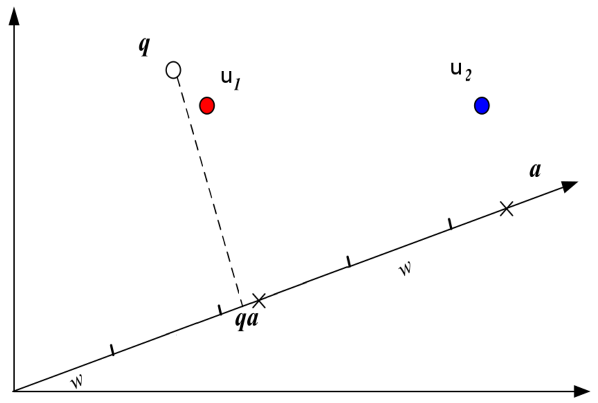


Figure 4.18: Geometric illustration of distance-sensitive projection

In order to minimize the distorsion variance of the embedding, we take a small collection of those hash functions h_{a_j,b_j}, corresponding to L different vectors $\overrightarrow{a}_j$'s $(1 \leq j \leq L)$) chosen independently, to construct the low-dimensional representation of all the IOs available in the network. Content search performed on this low-dimensional space is then a good approximation of searching in the original space, but with the cost of false positives and negatives. While false positives do not really have an impact on content search (they only produce results which are not related to the search query), false negatives (which induce missing results in the search output) can be troublesome, depending on the accuracy requirement of applications. In practice, if the bucket length is reasonably large, most proximate objects can be generally projected into the same bucket

Now, to implement content search over our embedded space within a DHT-based network, bucket identifiers are used for peers in the DHT. So for m-bit identifiers, we can order them in an identifier circle modulo 2^m for routing, similarly to the DHT Chord protocol. As we use L different hash functions $h_{a,b}$, we can order them into several rings located at different hierarchy levels (one ring for one hash function) to preserve content locality and path locality. On the Figure 4.19 we have summarized the comparison of simulations of our p-stable algorithm and Chord regarding the path length. The number of nodes have been set from 1E6 to 1E12, and the false positive probability to 0.2 (which depends mainly on memory size for the tables and number on content indexed). Both algorithms yield O(log(n)), p-stable having generally shorter paths but with more variance. The false negatives have not been taken into account for routing as it can be handled during the search by checking not only the bit corresponding to the projected bucket but also its neighbors (false negatives are due to the fuzzy borders on the projected buckets as hashing can project two close objects into two neighboring buckets). The variance is partly due to the random projections and partly due to the false positives, as instead of being able to get to a node with a shorter distance, the distance is then affected by a random value. Whereas for Chord, it only depends on the hash key value and how fast the dichotomy process will find the right node based on this value.

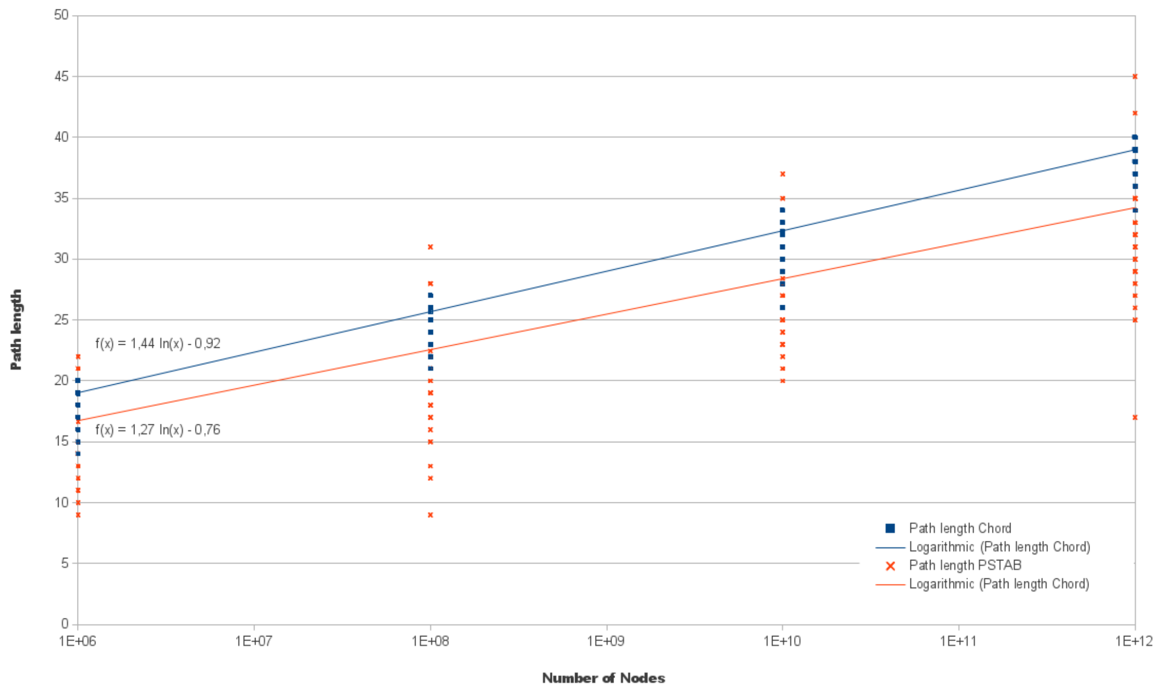


Figure 4.19: Comparison of our p-stable algorithm with Chord

4.3 Event with Large Crowd Evaluation

The EwLC scenario as described in Section 3.3 is about accessing named data objects in larger network crowd of people that generate and request such data objects — leveraging interest locality and the ability to communication locally, i.e., from device to device. We expect NetInf to provide several benefits in such scenarios.

The most important benefit is better user-perceived performance due to the fact that requests can be answered locally so that requests and responses do not have to be transmitted through a bottleneck infrastructure networks such as an LTE base station. This goes together with an efficiency improvement and cost reduction for network operators since requests and responses can be offloaded from the infrastructure network so that it is possible to serve a larger number of users.

The latter two benefits have been investigated in our evaluation work — leveraging real implementations, emulated as well we physical networks and simulation. Section 4.3.1 provides measurement results for a network of NNRP routers in an emulated EwLC scenario. Here, we have analyzed offloading potential and latency reductions. Section 4.3.2 provides result on NetInf bandwidth-saving potential (compared to current host-centric networking) based on measurements and an analysis of communication behavior in a large crowd scenario with short range radio technologies such as Bluetooth and WiFi Direct. Finally, Section 4.3.3 provides an analysis of ad-hoc routing and cache parameter tuning, based on simulation results.

4.3.1 Real-time Emulation of the Event with Large Crowd Scenario

We have developed a emulation framework based on ns3[11] for node mobility simulation and on Linux Containers (LXC[12]) as a virtualisation technology. This framework allows us to perform test with actual code, running in real-time with controlled and reproducible network conditions.

Using this tool, We have created an emulated EwLC scenario as described in Section 3.3.2. The objective of the measurements described here was to find out what performance savings and potential cost reductions NetInf could incur in an EwLC scenario.

The NetInf software that was used in these experiments was the NNRP (NEC NetInf Router Platform). We have created additional client tools that publish and request NDOs following specified patterns.

The network setup has been as follows:

- The network consists of 30 NNRP nodes and one NNRP publishing server (AP);
- Request generators contact NNRP nodes locally using NNRP's HTTP interface;
- NNRP nodes are configured to provide a WiFi (UDP CL) and a 3G (HTTP CL) interface;
- The WiFi UDP CL interface is used for broadcasting requests in the local network (so that peer nodes can receive and reply to those requests);
- The WiFi UDP CL interface is thus used as a local-network name resolution service: peer nodes that have matching entries for incoming requests would respond by sending back a locator (i.e., not the NDO itself). The requestor would then try to access the desired object in a second request (over the HTTP CL interface), using the obtained locator information;
- For each requested NDO requests are first sent out on the broadcast interface (with a 2 seconds timeout);
- If there is a response that provides a usable local network locator, the requestor tries to access the NDO using that locator;
- If there is no response (or the second request has failed), the requestor issues a request using the 3G interface (i.a., via the HTTP CL) with a timeout of 20 seconds;
- As soon as the object is available at the node it is cached and delivered to the application;
- At the beginning: all nodes have one (their own) object and publish it to the AP. Then each node continuously requests a random object from the total available list. All objects are equal size of 50593 bytes;
- Caches on nodes are limited to 131072 bytes, and can thus hold 2 objects. The cache on the 3GAP is bigger (16*1048576) and can hold all objects.

The emulation framework and its networks have been configured as follows:

[11]http://www.nsnam.org/
[12]http://lxc.sourceforge.net/

- All NNRP nodes run in individual LCX containers;
- The LXC containers are connected via linux software bridges, tun/tap devices and NS3;
- The WiFi network is emulated using NS3 as follows:
  - We are using the standard 802.11a model (WIFI_PHY_STANDARD_80211a) using the ns3::ConstantRateWifiManager with "OfdmRate54Mbps" DataMode;
  - We are using WiFi ad-hoc mode: NqosWifiMacHelper::Default and ns3::AdhocWifiMac;
  - The initial node locations are configured through: ns3::RandomDiscPositionAllocator" with Rho="Uniform:0:20"; X=20.0; Y=20.0;
  - Node mobility is configured through: ns3::RandomWalk2dMobilityModel with Mode="Time", Time=2s, Speed="Constant:1.0", and Bounds="0|40|0|40";
  - The LXC containers are connected via tun/tap devices and dedicated bridges per LCX container.
- The 3G network is emulated using traffic shaping on the software bridge as follows:
  - We use traffic shaping with tc[13] directly on the bridge;
  - The bandwidth limit is set via tc-tbf to 7.2 Mbit/s;
  - We have added a one-way delay added with netem[14]: 150ms (300ms RTT) ;
  - All LXC containers are connected to a single bridge.

Preliminary results The benefit of NetInf stems from the opportunity to obtain content from ad-hoc network connections and from the possibility to cache content. In a first run of our experiment we find that more than 50 % of the objects do not need to be transferred over the 3G network, thus providing a significant load reduction as can be seen from Figure 4.20).

The left-hand side part of Figure 4.21 depicts the distribution of times to successfully or unsuccessfully complete a request (Note this is a snapshot over a 10 seconds time frame only). Looking at the CDF of the request completion times, exhibits how many objects have been served from caches, from the local network or from the 3G infrastructure respectively:

1. Objects with x=0 are served from the cache;
2. Objects with x<2 are served from "WiFi", because after 2sec the request is forwarded over "3G" to the AP;
3. Objects with 2<x<20 are served from the "3G" network; and finally
4. Objects with x>20 failed, because the NNRP internal timeout (20sec) generated an error message back to the requestor.

Summary The tests performed with the emulation framework are useful as they allow us to conduct experiments with real prototypes (i.e., not only simulation) in networks of several nodes. The specific setup for emulating the EwLC scenario leverages broadcast in a local network to find requested NDOs there — before requests are made via the 3G bottleneck. Thanks to caching and a proliferation of popular objects in the crowd NetInf-based communication can help significantly to offload traffic from the infrastructure network, thus enabling important content distribution

[13]http://lartc.org/manpages/tc.txt
[14]http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

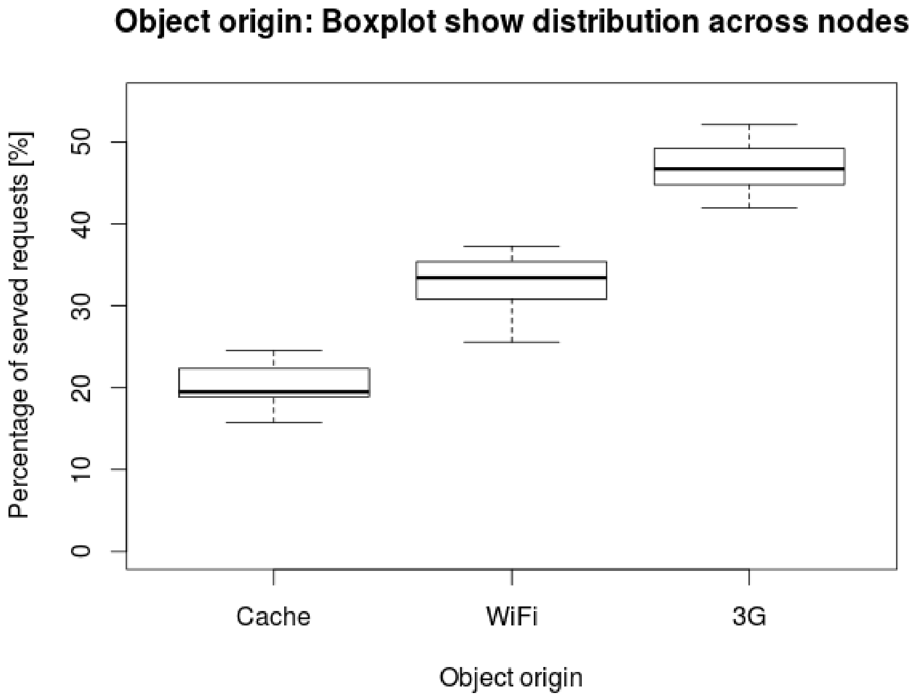


Figure 4.20: Object origin analysis

scalability enhancements. Detailed quantitative comparisons are obviously heavily based on the chosen settings (number of objects, nodes, cache sizes, timeouts etc.) — the conducted experiments demonstrate the potential in a basic setting. We are currently working on a larger setup that interconnects several of such networks and that is using different interest distribution patterns (Zipf-based distribution, leveraging the trace analysis that is described in Section 4.5.

4.3.2 Evaluation of Bandwidth Saving in the Event with Large Crowd Scenario

In this section, we will compare the host-centric and NetInf cases for content distribution in an event with large crowd scenario with short range radio technologies such as Bluetooth and WiFi Direct. The bandwidth saving potential of NetInf compared to the host-centric case is evaluated using the Android NetInf implementation described in Section 3.1.3.

4.3.2.1 Host-centric network case

In the host-centric case, the evaluation set-up shown in Figure 4.22 was used as described below.

- A 2.8 MB picture was stored in a web server in a fixed network. This is the content that will be requested by the mobile devices.
- To obtain the graphs that show the bandwidth usage (Figure 4.24 and Figure 4.25), the Wireshark IO Graph functionality was used.
- The picture stored in the web server was requested over a 3G network using the web browser in the mobile devices.

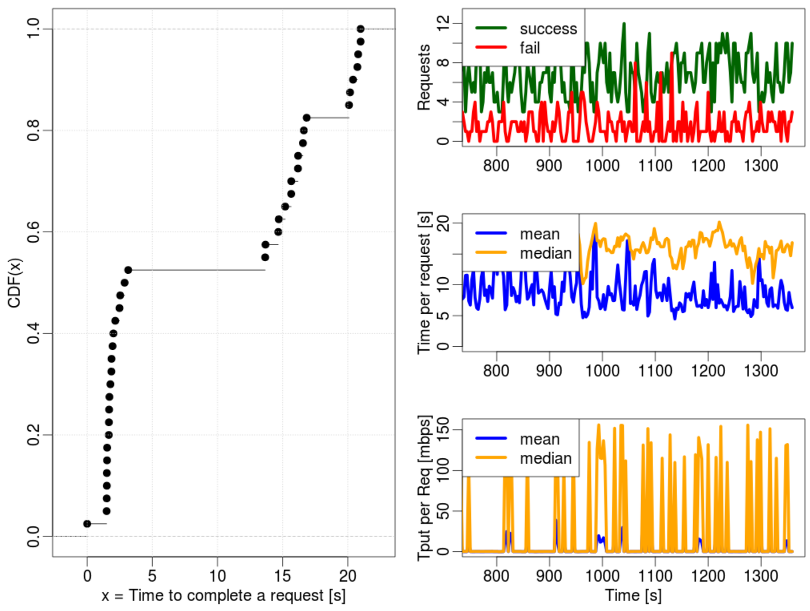


Figure 4.21: Performance analysis

- Four HTTP requests were made over 3G to the web server from the four different Android mobile devices available for the scenario test. The web server responded to each request with the content requested by the mobile devices (2,5 MB each).

For evaluation results and discussion, see Section 4.3.2.3.

4.3.2.2 Information-centric network case

In the information-centric case, the evaluation set-up shown in Figure 4.23 was used as described below.

- The Wireshark IO Graph functionality was used to obtain the graphs that show the bandwidth usage (Figure 4.24 and Figure 4.25).
- The starting point of this scenario is when the first mobile device publishes a 2.8 MB picture in the NCS. The content is cached in the local mobile cache, and it is also stored in the external NCS.
- For this scenario, QRcodes were used to get the identifiers of the objects. When the second mobile device reads the QRcode (e.g. from any screen available in the Stadium), an HTTP request (with the identifier=hashValue) is sent to the NRS to resolve that NDO. The response to the request is a list of locators containing the MAC addresses (e.g. Bluetooth, WiFi Direct) of the mobile device that took the picture and the IP address of the NCS. It was assumed that the first download was done from the NCS. The NCS replied with the content and the mobile device cached it. The last step was to register the MAC addresses (WIFI, Bluetooth or both if they are available in the device) in the NRS in order to make this interfaces available for future neighbors requests.
- Now, when the other three mobile devices want to retrieve the content (picture), they use the short range transport technologies (Bluetooth or WIFI Direct) instead of retrieving it from the server. The IO resolution was done using the NRS through 3G but the file transfer (2.8 MB)

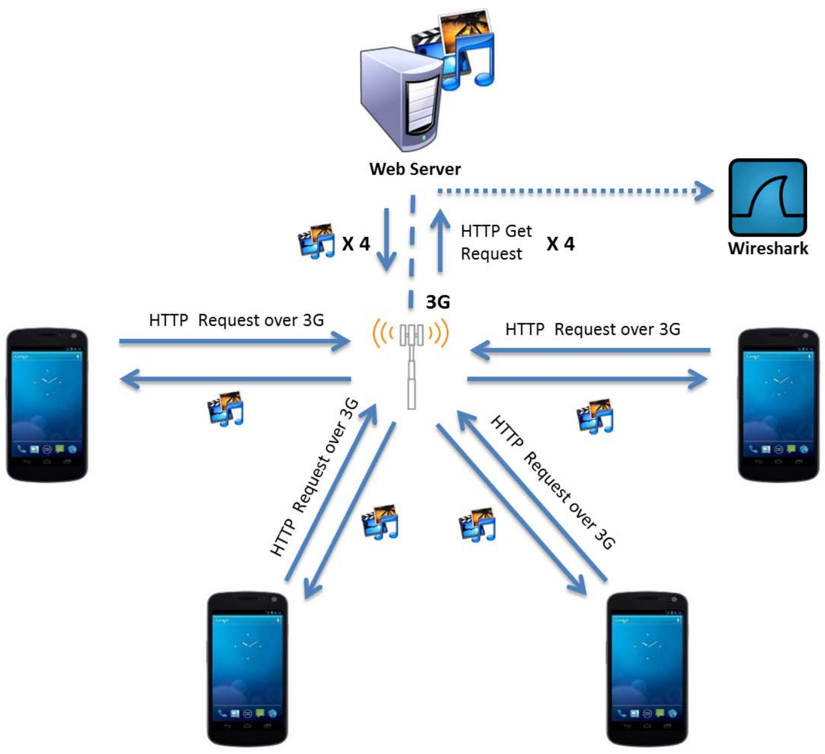


Figure 4.22: Host-centric network evaluation set-up

was done using the short range transport connection with the neighbor peers. All the devices cached the content making it even more available among the users in the stadiums.

- In total, one HTTP request was sent to the NCS (2.8 MB) and eight IO resolution requests (1 kB each) were sent to the NRS.

For evaluation results and discussion, see Section 4.3.2.3.

4.3.2.3 Evaluation results

This section describes the evaluation results for the prototype scenarios described above. First, the bandwidth usage in the web content server is analyzed for the host-centric network scenario. Second, the bandwidth usage in the NCS and NRS is analyzed for the information-centric network scenario. Finally, the bandwidth usage for the two cases is compared.

Host-centric network case
Figure 4.24 shows the IO graph captured with Wireshark for the host centric network case. The graph shows four curves that represent the bandwidth usage in the web content server when four content requests are sent from the mobile devices and the server responds. The scale used in the "Y" axis is logarithmic and represent Byte/second (bandwidth). The "X" axis represents time measured in seconds.

Information-centric network case
Figure 4.25 shows the IO graph captured with Wireshark for the information centric network case. The graph shows two different series, one representing the NRS and the one representing the NCS as indicated in the figure. Both series represent the bandwidth usage in the servers. For the

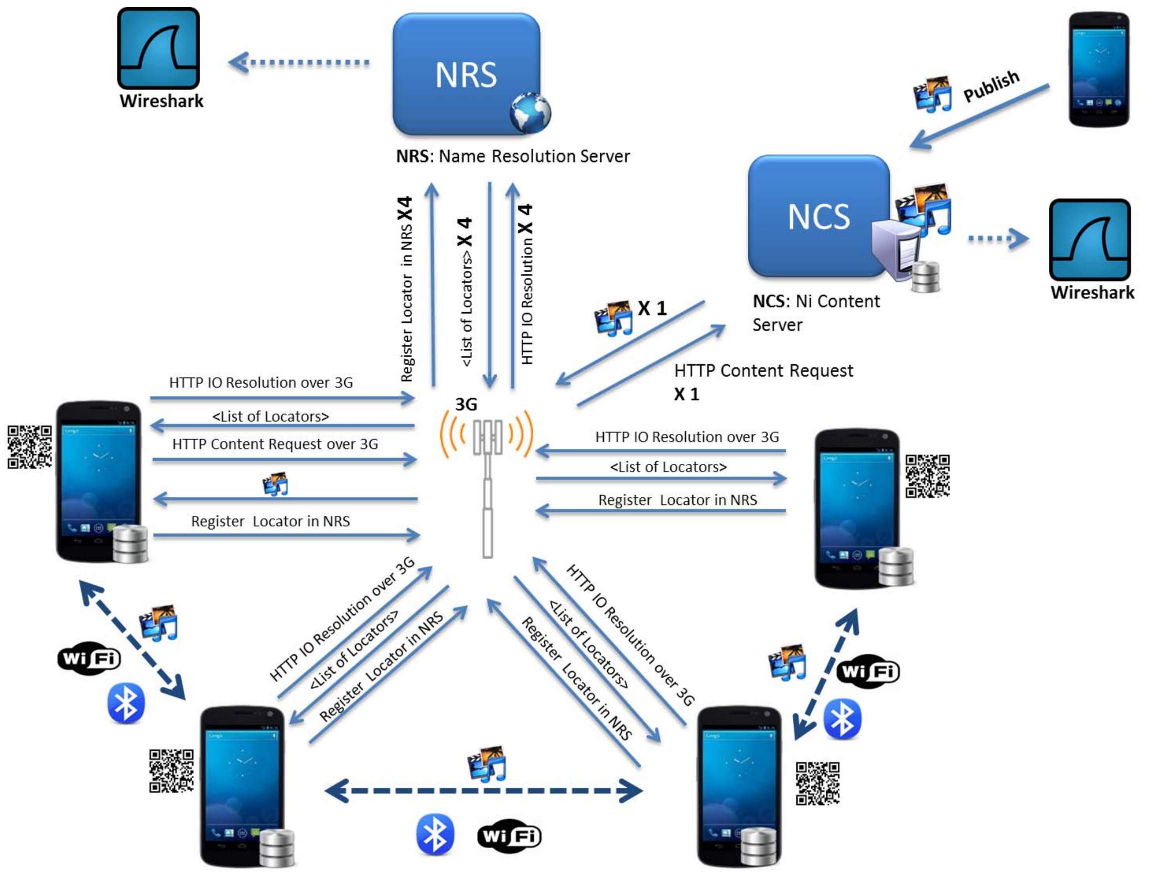


Figure 4.23: Information-centric network evaluation set-up

NCS, one wide curve represents the only content request that is sent to the NCS. Therefore, it is the only time in this scenario that a representative amount of data is sent over the 3G down-link. In the NRS series, it can be observed eight peaks that represent the different GET and REGISTER requests that are sent to the NRS to either resolve the IOs or register the locators. The scale used in the "Y" axis is logarithmic and represent the Byte/second (bandwidth). The "X" axis represents time measured in seconds.

Discussion and conclusions

In the host centric network scenario, the total data transmitted over the 3G down-link was 12 Mbyte. In this scenario, the mobile devices could get the picture separately using the 3G down-link, which is not desired in scenarios with large crowd where thousands of requests go through the same 3G down-link, thus generating link congestion. On the other hand, in the information centric network scenario the total data transmitted over the 3G down-link was 3 MByte, which included the first time when the mobile device downloaded the content form the NCS and the GET and REGISTER requests that were sent to the NRS (2.5 kByte per request). This implies 75% less 3G down-link bandwidth utilization for the information-centric scenario.

The results described in this section illustrate the benefit of using a NetInf architecture jointly with short range technologies for content distribution. The 3G down-link bandwidth usage was reduced by 75%, and the data requests handled by the content server was just one instead of four, i.e. less network resources consumption and less processing needed at the server side.

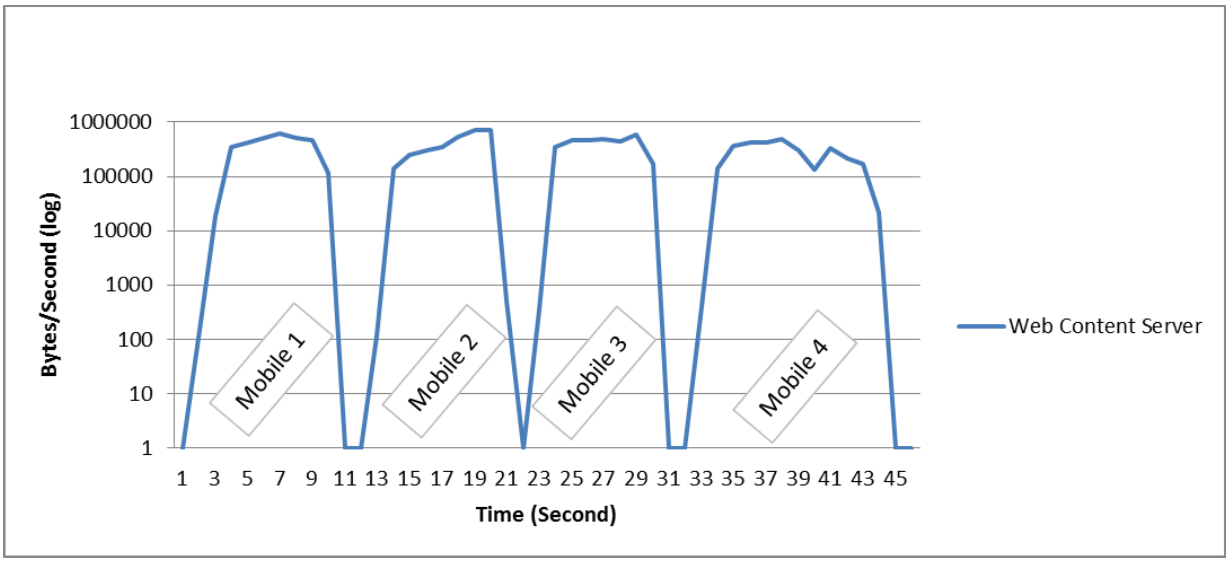


Figure 4.24: Host-centric network scenario showing the bandwidth usage in the Web Server

The purpose of these basic tests is to show how an ICN NetInf approach could be used in mobile devices (in this case Android phones) taking advantage of additional short range transmission capabilities, such as Bluetooth and WiFi Direct in order to overcome the 3G down-link congestion problem in large crowd events. Moreover, this scenario allowed us to demonstrate the interaction between several of the NetInf components (Node, NCS, NRS, etc.).

It is important to highlight that the results obtained in this scenario (75% of reduction of the bandwidth usage in the 3G down-link) were obtained using four devices in the same Bluetooth range. Therefore, after the first device caches the object from the NCS, all the other devices were able to find at least one nearby peer to retrieve the content from instead of going to the NCS again. In the real scenario were users are scattered all over the event, it can be expected that when new content is available, most of the first requests for such content will end up being answered by the NCS. The more popular the content gets, the higher is the chance that the requestor will find a nearby peer from which it can retrieve the content instead of going to the NCS. As soon as peers start finding adjacent devices that have the desired content, the down-link usage reduction will start. Of course it cannot be assured that it will always be such a considerably reduction as 75%, since this number depends on the scenario at hand.

Additional tests have to be done to simulate a more complete and complex scenario with more devices participating in the content distribution. Simulations could be suitable for further studies to show the network behaviour of hundred and even thousands of mobile devices interacting in the scenarios described above.

4.3.3 Simulation of Routing and Caching in the Event with Large Crowd Scenario

An ad-hoc routing and caching simulator tool was developed to evaluate different routing and caching algorithms in EwLC scenarios. The main purpose of the simulator is to answer questions like what ad-hoc routing and caching parameters are needed to improve performance in a NetInf EwLC with a limited wireless uplink capacity to a fixed network.

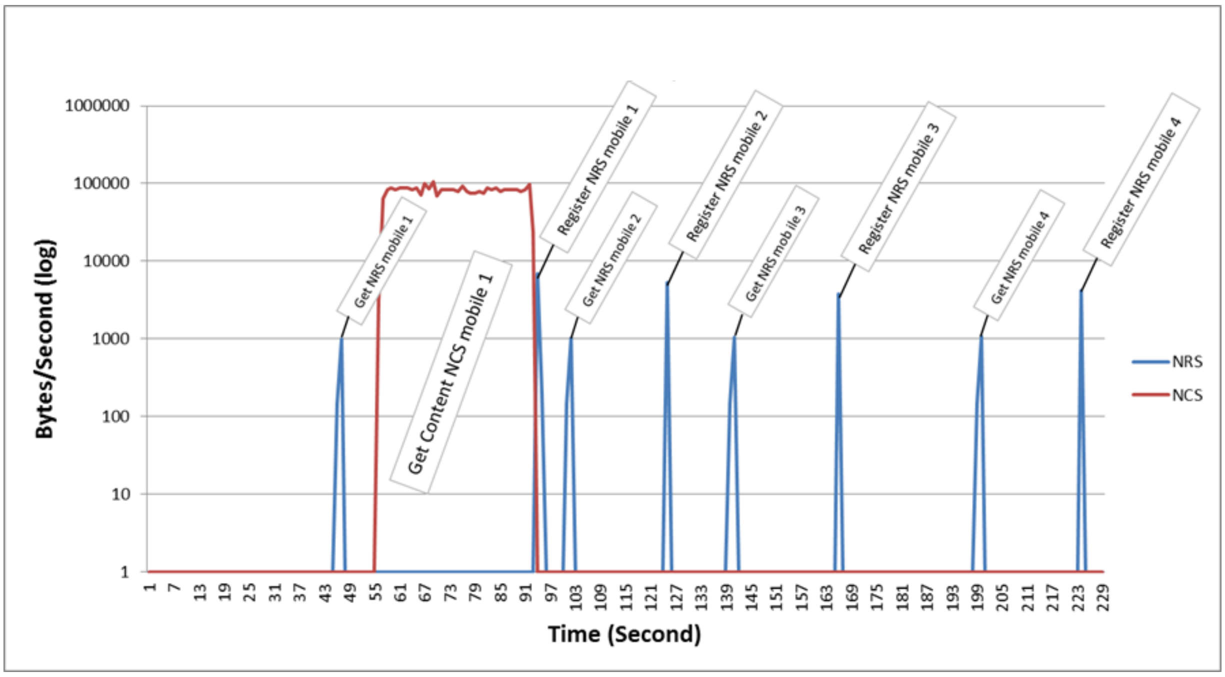


Figure 4.25: Information-centric network scenario showing the bandwidth usage in the NCS and NRS

4.3.3.1 Overview of the simulator

In its current version, the simulator does not consider limited traffic capacity in links between users or in the wireless uplink connection. The wireless uplink however is assumed to be limited in capacity. Due to this assumption users always try to retrieve objects locally before using the fallback option to download the object using the wireless uplink connection.

A typical simulation scenario is to start with a crowd of users, where the users have not downloaded any objects yet. Then users in the crowd start downloading objects. In the current version all requests are sent sequentially with a constant (configurable) delay between each request. The first users that try to download objects from neighbors will fail and after a timeout users will download content from the uplink. After a while more and more users will have the content in their caches and the probability of finding the object within the crowd increases.

An alternative scenario is that that there are a number of objects "preloaded" in the crowd. Then the possibility to download the content through the uplink can be turned off.

When running a simulation, a graphical view of the crowd is updated. If timing in the content request pattern is configured properly, it is possible to visually follow the routing of object requests, object transfers and caching of objects.

Figure 4.26 is a snapshot of an ongoing simulation. Each yellow square represents a user. The symmetrical location of clients in this figure could represent an audience in a stadium with all seats taken. Other scenarios with less dense and irregular user locations can also be defined. The small colored marks in the figure show:

1. Red square: This user has failed to download the object from a close local source. After the timeout it downloaded the object through the uplink.
2. Green square: This user succeeded in finding the object and could retrieve it from a neighbor (The user that earlier failed at 1.)

3. Blue square: The object is cached at this user when the object was forwarded through ad-hoc routing between two close nodes (1. and 2.).
4. Empty (yellow) square: Represents a user that has sent out a request to neighbors, but has not yet got a the object back (or reached timeout and downloaded the object through the uplink)
5. Small red circle: Represents an ongoing request sent from the user at 5. Will disappear after timeout.

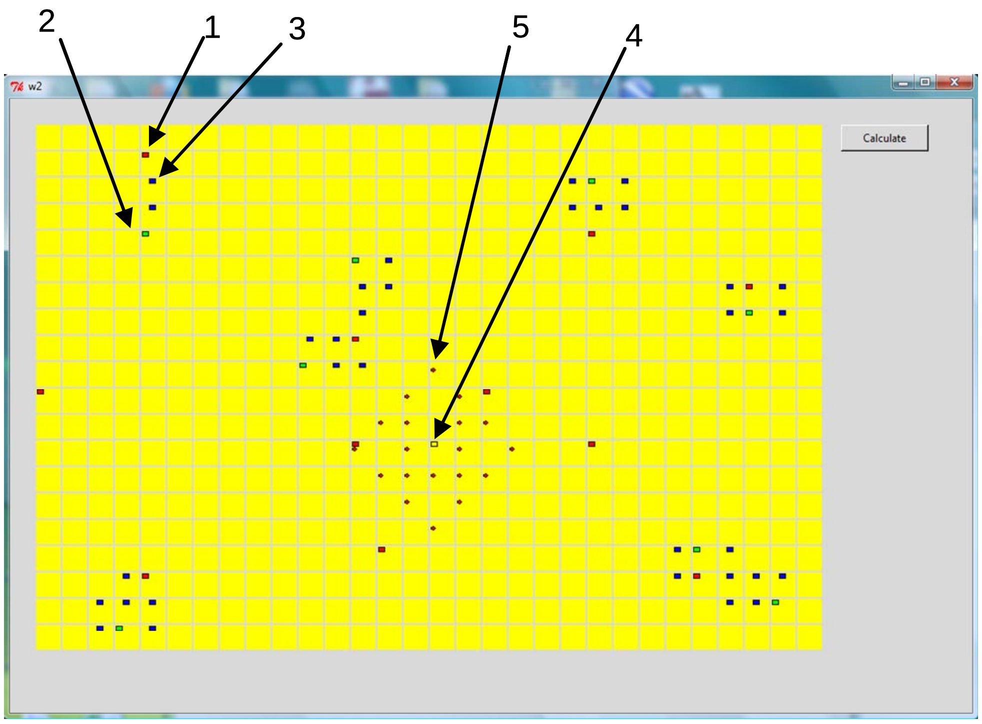


Figure 4.26: Snapshot of the simulator graphical display

When the simulation is complete a number of result parameters are presented.

Simulator input parameters:

- Crowd user locations defined by (x,y) coordinates;
- Neighbor reach ability;
- Pre-loaded content;
- Allow clients to download content through uplink connection when content is not found locally;
- Content request pattern, predefined or random;
- Ad- hoc routing Time To Live (TTL). Single TTL or gradually increasing TTL;
- TTL timeout;
- Routing algorithms, Broadcast to all reachable neighbors or to predefined or randomly selected neighbors;

- On route caching enabled or disabled.

Simulator output results:

- Number of objects that are retrieved from other clients in the crowd;
- Number of objects that are retrieved using uplink connection;
- Number of cached objects in the crowd;
- Total amount of object transfers in the crowd;
- Total number of object requests in the crowd.

If the transfer request transaction spans over several node-to-node hops, each hop is counted as one.

4.3.3.2 Results

After doing a number of simulation with different input parameters it is quite obvious (as could be expected considering the large number of input parameters) that it is impossible to find the "best" set of input parameters to be used in a "general" EwLC use case. The only way to find a set of optimal routing and caching parameters is to narrow down the use cases to a specific actual case such as a specific event on for example a stadium, but these parameters might not even be valid on the next event on the same stadium. It is important to keep this in mind when evaluating data from the simulations.

Another observation from the initial simulations is that the graphical display of the simulation gives a lot of "intuitive" information that is very useful when evaluating the result and tuning parameters. One example of this is that a random content request pattern can result in "chaotic behavior". The result of a simulation can vary depending on witch users make the initial requests.

Below we describe a few example results.

Example 1

A crowd of 600 users are evenly distributed in on a limited area (could be a stadium) Each user only has connectivity with the closest neighbors (short reach RF link). Ad-hoc routing is used and each user making a request sends it to all neighbors it can reach, and the neighbors receiving the request returns the object or sends the request to all their neighbors until TTL is reached. When the requested object is found and sent back it is cached at all nodes on the return path.

Figure 4.27 shows the local hit rate as a function of the number of requestors (percentage of all users) for different TTL values (TTL = number of request hops before request is terminated).

Figure 4.28 shows the corresponding object transfer traffic. The Y axis shows how many user-to-user object transfers that were made in average when one user requested and retrieved one object. This value can be much higher than one because the returned object will do more than one user-to-user hop and also because the object can be found at more than one user. All of the users that have the object cached and get a request will return the object.

There are some obvious conclusions that can be drawn from the two graphs:

- Increasing TTL above two seems to saturate the performance improvement while at the same time generating huge amounts of object transfer traffic. It seems like a TTL of 2 or maybe 3 is the best choice for this scenario.
- Objects that are requested by less then 5% of the users will be difficult to find through ad-hoc routing in this scenario.

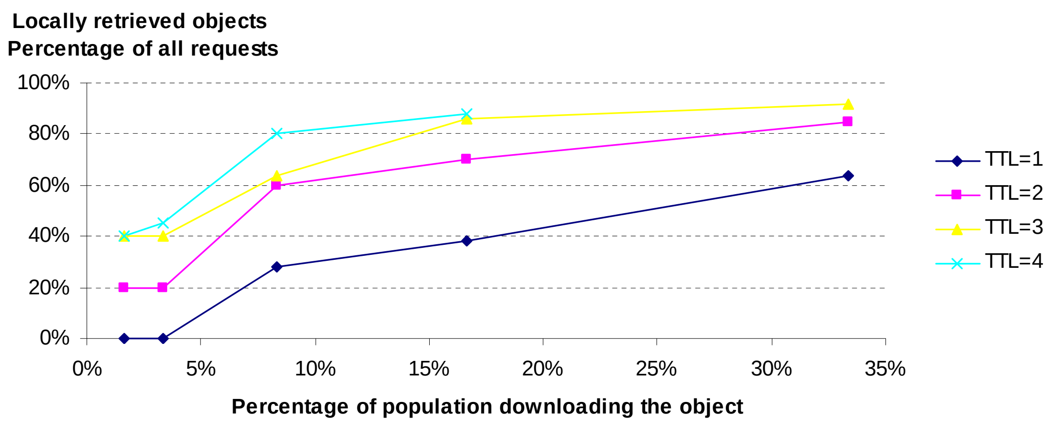


Figure 4.27: Local hit rate for different TTL and content popularity

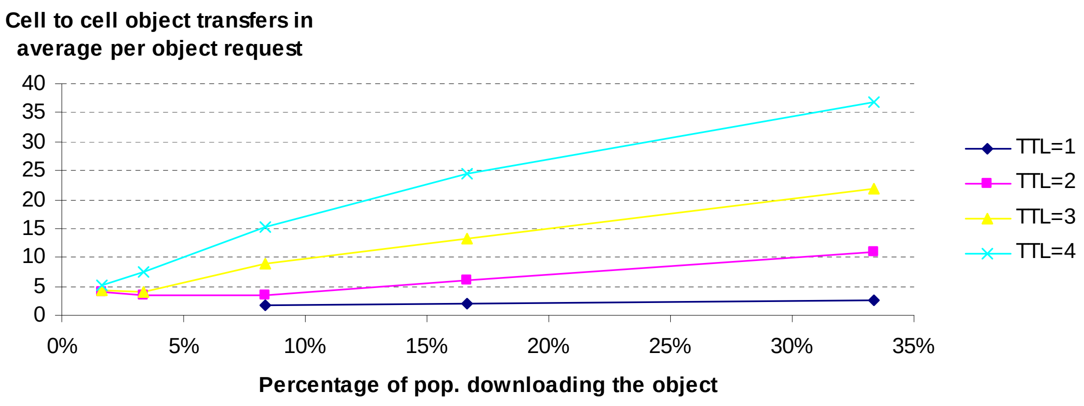


Figure 4.28: Object transfer traffic for different TTL and content popularity

Example 2

One intuitive assumption is that the more you cache the better the performance of ad-hoc routing in the EwLC. The simulator can be configured to cache objects on each node a returned object passes when being forwarded (on-path caching). The effect of this caching however turns out to have significant influence only at high TTL values and/or when a large fraction of the users at an event requests the same object. The on-path caching does however increase the total user-to-user traffic. The reason for this can be understood by looking at Figure 4.26 above. Let's say that a user in the figure having both user 1 and user 2 in its TTL range sends out a request. Then user 1 and user 2, and the both users in between (blue square = object cached) will return the object. This means that the traffic will be doubled compared to if no on-path caching is done.

4.4 Network Management

The objectives of network operations and management are to address specific requirements for a commercial, "carrier-grade" deployment of NetInf protocols. Thereby, NetInf elements and scenarios are analysed from a management perspective, considering topics such as resource provisioning, cache population, content- and load-balancing, longer-term dynamics of caching, and commercial deployment issues such as charging, and implementation of operator policies. In the following

NetInf is evaluated with respect to required and desired network management and operational functionalities.

4.4.1 Network Management

The goal is to make NetInf largely self-managing, through a combination of features provided by the NetInf Transport services and considering requirements from specific deployment scenarios. Self-management techniques are necessary to allow NetInf concepts be deployed in dynamic environments, where the change of parameters (mobility, depletion of resources, ...) require adaptive algorithms to reach the optimum performance to specific situations (cache placement, routing, ...). This goal can be met by adopting a self-management framework. Requirements for a self-managing NetInf deployment regarding the following key topics are discussed:

- self-configuration
- self-healing
- self-optimization
- key performance indicators

Self-configuration Self-configuration is an essential feature of any NetInf deployment. Essential items for a self-configuring NetInf network are:

- NetInf clients autonomously find and connect to the next NetInf access router.
- NetInf routers are plug-and-play, this includes autonomous topology detection and integration in local NetInf domain, setup of default route and connection to NRS. E.g. NNRPs are able to find each other in the same broadcast domain without specific name resolution, see also Section 3.1.1.
- NetInf domain routers must be able to join the local and global NRS. This is realized by organizing the NRS in a flexible data structure (DHT, SkipList) and having a bootstrapping mechanism as first entry point.

Self-healing Access to content through the NetInf infrastructure requires a resolution step. The reception of a data access request in a domain requires high availability of the resolution service. The resolution service can be centralised or distributed, but it still has the same requirements. This puts great reliability requirements on the NRS to be available for the very basic `PUT`, `GET` and `SEARCH` messages. Further, content distribution through caches provides some degree of reliability in terms of content replication. For example, if the transit link behind the caches fail they can still service content request, or if one cache becomes inoperative other caches still remain functional. But in order to exploit the improved reliability of replicated content in the caches, a NRS needs to be used to locate these replicas in off-path caches. This increases the reliability requirements of NRS which needs to be at least of the same or even better level as the replicated content.

Internet's NRS, namely DNS, is a hierarchical global NRS where reliability is achieved by using caching in DNS servers and clients and by making DNS service reliable, for instance, by having a set of DNS servers for providing the same resolution service. This is not adequate for the future needs, where the NRS system needs to support order of 10^{15} data objects replicated in multiple caches and even hosts. The implications of this is that a strict hierarchy of NRS similar to DNS is impractical because of the overhead of maintaining the configurations (that include records for the caches hosting content as well as records pointing to peering and parent NRS nodes that may be many). Therefore, a more distributed and self-configuring solution is needed. A solution that can work on both on flat and hierarchical names. More importantly a local failure of a name resolution system must not radiate across a global network but the failure and its implications should be contained within the domain where the failure happened.

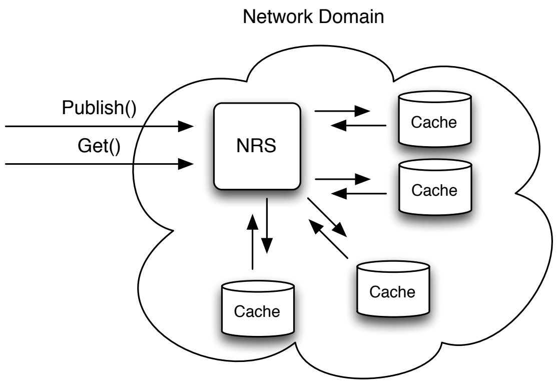


Figure 4.29: Example of NRS's operational environment

Self-healing techniques are necessary to ensure the availability and redundancy of content. NetInf is managing the migration of content in normal operation and in graceful degradations, where a NetInf entity is switched off or shutting down with previous notification (e.g., for maintenance), thus being able to migrate stored/cached content to another running NetInf cache. However, sudden failure of a NetInf entity will inevitably result in the loss of data stored on this entity. Thus, another entity must be responsible for making the data/content available once again. This could, e.g., imply ensuring a certain number of available copies of the content network, by dynamically creating new copies if the number of copies reaches a certain lower threshold. Or, if no other copies exist, data needs to be recovered from the original source or "owner" of the content. Also NetInf nodes (routers and clients) reorganize in terms of failure of a NetInf router, keep connectivity and update routing/forwarding tables.

In order to improve NRS's capability to revive itself after failures, self-healing techniques are required and NRS together with caches can implement self-healing feature. This is achieved by replicating key parts of the name resolution information in the distributed caching system, which then could be used to re-build resolution mapping tables when the NRS system is recovering from a failure. Figure 4.29 shows an example of NRS's operational environment, where NetInf protocol `Publish()` and `Get()` messages (Section 2.3) represent events in the network domain requiring resolution services. Similarly, there can be other types of messages also requiring resolution services. In the figure, NRS is represented as a single logical node, but the described approach is also valid for a distributed NRS implementation involving multiple nodes assuming that the identity of each NRS instance is stored in caches to ensure that each cache can identify the requesting NRS and find its data.

In the figure, the NRS can also take care of cache control functionalities like selecting what caches are used to store the published content. Alternatively, Network Management can take care of cache control duties and NRS can consult it, when needed. When the NRS crashes for instance, in the worst case, it will loose all its mappings, i.e., all information about locations of received and stored publications. In such cases, it is essential that the failure is not visible to the outside, thus the domain is autonomously able to restore the NRS's state to where it was before the failure.

In order to accomplish this, in some cases it is sufficient that a NRS rebooted and its internal states are retrieved from its persistent storage. But this does not always work, for instance, due to severe hardware breakdown. And these kind of cases, the only option to restore the resolution service is to instantiate a new NRS. Network management system has a central role here by governing how NRSs act during and after start-up phase including also management tasks done to get the network into a normal running phase. It is expected that the network management monitors the NRS through the same mechanisms as any other network equipment. These mechanisms are outside the scope of NRS's self-healing properties. Independently on which kind of failure case is in question, a failure

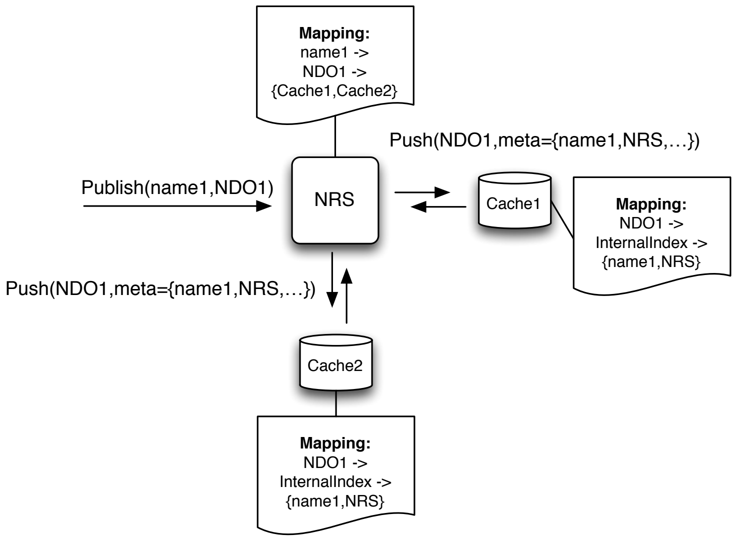
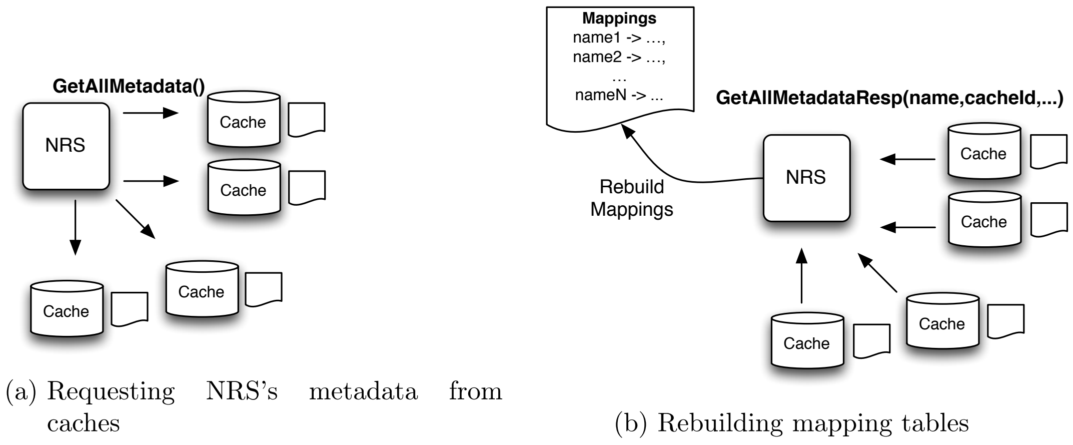


Figure 4.30: Example of publishing phase

of NRS can be recovered as following: each cache will store some additional information (metadata) for each of the cached NDO it has. This metadata consists of (at least) the following items; i) the ni name of the object (NDO) and ii) the identifiers (e.g., locators) of the NRS nodes that was/were used for the publishing the NDO into the domain. Figure 4.30 depicts an example where a client publishes the content “NDO1" named as “name1" to the NRS, which then selects the caches to store copies of the content. It should be noted that in practice, these names used for published content are ni names introduced in Section 2.2. The NRS then instructs the selected caches to store the NDO with the metadata expressing the NDO's name and NRS's identity that was processing the publish message.



(a) Requesting NRS's metadata from caches

(b) Rebuilding mapping tables

Figure 4.31: Self-healing phase

When either a replacing NRS becomes operative or an original NRS has rebooted, it will query (`GetAllMetadata()`) all the existing content copies and their objects' metadata based on the NRS identifier (see Figure 4.31a) to rebuild the resolution database representing the current state of all local caches (see Figure 4.31b).

Once the NRS has rebuilt its mappings, the domain has restored its full capability to provide resolution and caching services. Relating to Figure 4.30, there is a special case for publishing, where the content that is only stored in the host without a copy in the network as shown in Figure 4.32.

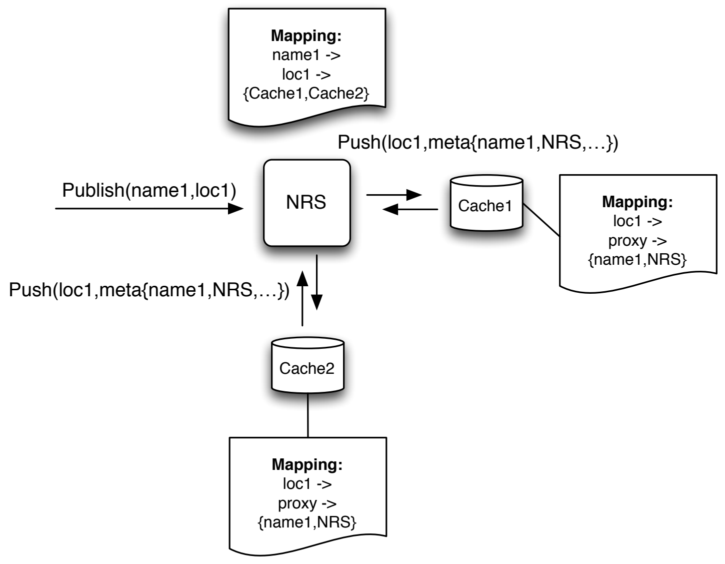


Figure 4.32: Example of registering phase

In this case, a NRS during the content publishing phase selects (at least) one of the caches to act as a proxy for the host that only maintains a replica of the host's locator and name-binding with empty content container. The content remains with the host. This proxy binding of name and the host containing the content itself is mandatory for recovering the NRS mapping database in cases of failures as explained above.

Self-optimization Self-optimization is essential for an efficient operation of a NetInf deployment. In particular, when deployed in a virtual network slice, e.g. as realized with CloNe, the NetInf network may adapt to the demands in terms of number of NDOs, total requests and requests per NDO, but also with respect to the churn of NDOs and their location/locators. Specifically, the following tasks are subject to self-optimization

- Content placement: Content placement includes cache replacement strategies including collaborative caching (see e.g. D.B.2 Section 3.3.6 [5]) and cache pre-population strategies. Collaborative caching strategies also require an exchange of information, e.g. for a popularity based caching the global popularity of an NDO needs to be monitored by aggregating local popularities.
- Cache placement and dimensioning: The NetInf network should be able to monitor the utility of a cache and its size. Via a resource description language it can communicate the resource demand for caches and network topology to CloNe and thus adapt the connectivity, bandwidth, and size of the deployed caches.
- NetInf router placement and activation: If NetInf is deployed as an overlay on top of IP services or in particular in a CloNe slice, NetInf routers can be deployed dynamically on network or CloNe nodes according to the request demand and number of NDOs in order to keep lookup tables and forwarding delays small.
- The NRS system should be organized in a way that delays for name resolution are stable, e.g., the underlying data structure (mDHT, SkipNet) must adapt to the number of entries by reorganizing or adding/deleting nodes.

Key Performance Indicators KPIs are monitored in order to evaluate the long- and medium term performance of the system that may trigger self-optimization or self-configuration processes, to detect imminent service degradation or even system malfunction and trigger self-healing processes. In general KPIs of NetInf nodes and services include typical network KPIs like CPU, bandwidth consumption, delays. NetInf specific KPIs are

- Number of NDO request outside the domain
- Number of NDO request from a peering domain
- Number of ni-validated and non-validated names stored within the domain
- Number of incorrect routing hint resolutions: number of request for that no NDO is found even if the hint was pointing to the domain
- Free storage capacity in a domain: capacity of a domain to store further NDOs
- Global Success Rate: Ratio of locally requested NDOs that the global NetInf network is able to return.
- Local Success Rate: Ratio of NDOs that are requested from and can be served from the local domain.
- False Delivery Rate: Rate of NDOs that are requested after being removed from the system (unpublished) and nevertheless delivered
- NRS LookUp Success Rate: Ratio of NDO requests that the global lookup service can resolve.
- NRS LookUp Rate: Ratio of locally requested NDOs that require a lookup.
- NRS LoopUp Delay: Average time until a locator/routing hint/next hop is returned from the global lookup service.
- Forwarding Delay: Average time for forwarding a request.
- Request Processing Delay: Average time for processing a request in a router.
- Publishing Delay: Average Time from publishing an NDO to the NDO being (globally) available.
- Content Delivery Delay: Average time from requesting an NDO to delivery of the NDO. Could be measured at every NetInf node including in particular the client.
- Cache Hit Rate: Ratio of requested NDOs that can be served from the local cache.
- Caching Gain: Average number of hops to next cache serving the same content.

4.4.2 Operational Considerations

The operational consideration reflect requirements for operating a NetInf deployment beyond node management and configuration. Here, important aspects are global domain management, content management, and operational models

Global domain management The global scalability and the inter-domain interoperability is based on the concept of domains that are encoded in the authority part of the NetInf naming scheme and a global NRS that is able to return locators or routing hints for domains. This global NRS is the key component for maintaining global inter-connectivity and fulfilling the key service of a NetInf network to make published NDOs globally accessible in the same way as the current Internet makes IP addresses globally accessible. Though the NRS should be mostly self-organizing as it is e.g. implemented as a hierarchical DHT still an authority is required to manage the NRS. The tasks of this global authority would be to authorize domain names (authority part in naming scheme), to grant publishing access to the NRS, to control whether domains actually serve their published NDOs, to provide a bootstrapping mechanism to enter the NRS, and finally to maintain the stability and integrity of the NRS.

Content access management The role of a NetInf operator goes beyond the role of an ISP as the NetInf operator gains increasing responsibility for content, in particular if NetInf becomes fully responsible for storing and not only caching content. As a consequence NetInf needs to implement a content access management which means in particular functionality to restrict the access to content to authenticated requests but also to control where content is cached. This also includes the possibility to delete or unpublish cached NDOs across NetInf operator domains. NNRP currently offers the feature to unpublish objects, however, this does not include deleting cached copies of an NDO.

Cache management When deploying objects in the network, we need to dimension the caches. These will initially be determined by the technological constraints, price per GB and access speeds. These were outlined in D.B.2, however some tuning will still be needed and that is to ascertain which objects are popular and will be cached in the space provided. This will depend on the popularity of the objects being published and requested. The Orange trace data gives us some indication of the object size for http requests, but other items will be requested, such as films and music files.

Summary We have discussed management in the context of NetInf networks. Given the large number of objects, some degree of self-management is needed which has been outlined above. Mechanisms have been described which use metric indicators for triggering actions automatically.

4.5 Traffic Analysis

This section reports on traffic analysis related to ICN in general, rather than specifically to NetInf. An analysis of the economical gains of caching is described in Section 4.5.1, and an analysis of Orange traces is described in Section 4.5.2.

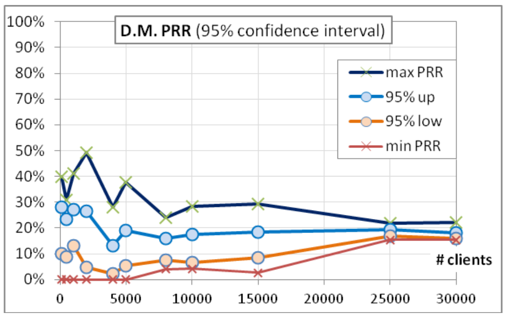
4.5.1 Caching Economical Gains

In this section, we analyze the potential global economical gains of developing caching in a network infrastructure. Caching is a key component of today architecture (e.g. CDNs) and is natively integrated in NetInf and most ICN architectures. While the current main driver for cache deployment is Quality of Service (QoS) improvement, its economical efficiency will also determine the scale of its development in future ICN architecture. By eliminating the need of duplicated transport of identical objects, caching allows an architecture to scale much more efficiently. It is hence critical to understand the global incentive to deploy caches in a network. Note that, as identified in Section 2.5, the interest of an individual operator to deploy caching might not follow these results: gains from transport cost reduction could be nullified by the decrease in peering revenue. We are here however speaking of the global network efficiency, and since ICN adoption will most likely trigger an update of business models, we prefer to analyze the global efficiency of caching, and assume that business models will provide correct incentives for each player to move towards the globally most efficient configuration.

The economical efficiency of caching obviously depends its technical efficiency, which depends on the nature of the traffic. Using traces from the Orange network, we here first compute typical technical efficiency of caches for two types of traffic: one considers HTTP traffic which is destined to the DailyMotion[15] website, and the second trace corresponds to a managed Video on Demand (VoD) service, with a much smaller catalog but higher quality of service. These two types of traffic show distinct characteristics, and represent most of the cachable traffic nowadays. We then apply an economical analysis to determine the incentive of deploying caches, as a function of the technical efficiency. Note that this second step is independent of the traces, and can be applied to any traffic, provided we know how efficiently it can be cached.

[15]DailyMotion is a Youtube-like User Generated Content (UGC) video website, which is popular in France.

Peak Rate Reduction Ignoring energy savings, most of the gain of caching comes from the load decrease at higher levels in the network, and therefore reducing the corresponding link dimensioning. We assume here that links are dimensioned according to the offered peak rate.[16] Within this context, Peak Rate Reduction (PRR) is the key technical metric for caching efficiency, since it determines the decrease in dimensioning needs.



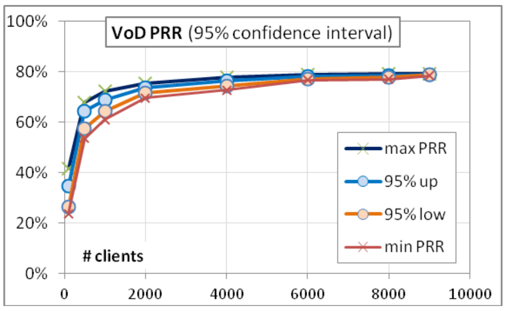


Figure 4.33: Trace-based PRR for DailyMotion (left) and VoD traffic (right), as a function of number of users

Figure 4.33 shows the PRR observed over sampled subsets of clients of various sizes (note that the 95% confidence interval defined with respect to the choice of active user, not with respect of time). Due to different popularity distributions, the PRR differs significantly between the VoD and DailyMotion cases. The results obtained for VoD traffic are very stable and allow deriving dimensioning rules given a size of population covered by a cache. For DailyMotion traffic, however, the PRR oscillates in a relatively large interval (say from 10% to 30%) with values that seems to converge around 20% when the size of the population increases. For dimensioning purposes, we will hence consider various PRR values and try to derive some impacts on the overall solutions.

Comparing saving of several configurations We first briefly describe the main assumptions of our models.

- **Network**: Our model is based on the French Orange Network, including regional networks. In the more detailed study, the backhaul level is also taken into account.
- **Service**: The models we use are generic in the sense that they can be applied for any type of service, once properly characterized by a catalog size, a demand level and delivery bit-rate. We focus once again on managed VoD service on one side, and OTT DailyMotion services on the other side. The growth in catalog size and delivery rate with years is taken into account.
- **Customers demand**: Several values were considered for the intensity of demands (percentage of users simultaneously connected — and active — at the peak hour) ranging from 1% to 30% of 6.5M clients. The value in 2012 is around 2%. Typical values are: 6% in 2016, 13% to 25% in 2020. We consider here the demand globally without distinctions between services.

[16]Note that the peak rate is defined here in a "fluid" way which ignores the effect of TCP rate evolution: to each request is associated a required rate (the size of the requested object divided by the session duration) and the time interval during which this request is satisfied. At any given time is hence associated a required rate, which is the sum of the request required rates for requests which are currently served. The peak rate is then defined as the maximum of these values. Our dimensioning rule therefore ignores any short term rate variation which would be due to the transport protocols, but take into account the (even short-term) variation of demand from the users.

- **Caching equipment characteristics** are based on realistic actual figures: streaming capacity is evaluated at 7 Gbps, whilst we consider two values (7 and 15 TB) for storage capacity. The purchase cost for a cache is evaluated at 20 k€.
- **Transport and Transmission costs** are evaluated by making a precise network inventory (real IP and WDM ports and chassis, content delivery servers) of all equipments required to accommodate the traffic. This leads to a typical cost for a link around 10 €/Mbps.

Three typical scenarios are considered by varying the potential location of cache servers:

- **Scenario SC1**: reference scenario without any distributed servers. The origin content server delivers all contents.
- **Scenario SC2**: scenario with one level of regional caches (named Intermediate Cache Server (ICS)) located at all regional POPs.
- **Scenario SC3**: scenario with one level of local caches (Leaf Cache Server (LCS)) located at all Network Edge (NE) nodes.

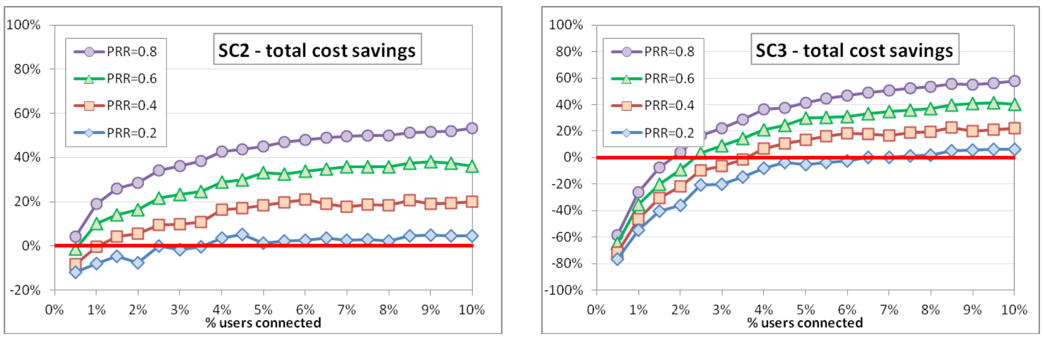


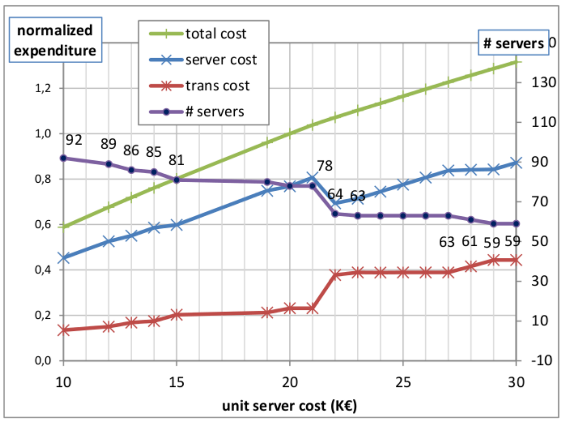
Figure 4.34: Relative savings from Caching for SC2 (left) and SC3 (right) deployment scenarios, as a function of number of connected users

Figure 4.34 displays the relative gain[17] obtained while deploying SC2 or SC3 (with respect to the reference scenario SC1) for different PRR values and different demand intensities (% of users simultaneously connected, on the x-axis). Values under the ref 0 axis mean that it is not beneficial to introduce caches whereas values above this red line indicate solutions where the caching servers induce a benefit. For instance, if we consider that caching techniques will allow a PRR of 80% for VoD, then SC2 is already a viable solution with 0.5% of users connected at once. With 2% (the connection rate measured in 2011), the benefit is already 30% of the total cost and the potential economic gain could grow up to about 45%, if 6% of all customers would require a video at the peak period. It takes a higher value of demand to cross the threshold in case of more disruptive scenario SC3, but above 2%, this scenario is also beneficial for high PRR services.

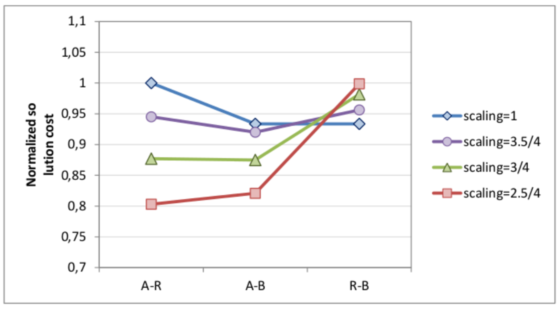
For services with a low expected PRR (for instance between 20 and 30% for OTT traffic such as Daily Motion), the benefit of caching is less obvious. The situation is worse with SC3 where the cost induced by the huge number of servers to deploy (252 locations instead of 28) makes the scenario profitable only for high values of demand (above 8%).

[17]Absolute values are not communicated, for business confidentiality reasons.

Sensitivity analysis A simplified model has been designed in order to investigate further the impact of some parameters. The topology is limited to the backbone network, and links are categorized in 3 groups only according to their depth in the network, and a mean price per Mbps is assumed for each group. This simplification allows to obtain optimal cost configurations where the caches can be deployed heterogeneously on the various node locations of the network, and hence to evaluate the sensitivity of the obtained solutions to each one of these individual parameters.



(a) as a function of cache cost



(b) for different configuration and scaling factors

Figure 4.35: Normalized optimal cost

Figure 4.35a displays the impact of the cost of each individual cache on the optimized solution for VoD services, with a focus around the actual cost of about 20 k€. Cost are normalized by the reference cost obtained with 20 k€ caches. We can see that the overall cost of the solution increases smoothly when the cache cost increases, although a significant gap can be observed between 21 and 22 k€, essentially due to the fact that this threshold value induces a massive migration of caches one level higher. This smooth increase means that our results are resilient to our assumption of cache costs. Similar smooth evolution are also obtained when studying the sensibility of the overall cost with regards to streaming or storage capacity of caches.

To investigate further the impact of the equipment characteristics on the cost of a caching solution, we have artificially created two additional caches with doubled (A) or halved (B) storage and streaming capacities compared to the referent (R) cache. Their cost is estimated according to several ratios that simulate the "economy of scale": a scaling factor of $\frac{x}{4}$ means that the cost is multiplied by x when the capacities are multiplied by 4.

Figure 4.35b shows the results obtained when using two different types of servers (A and R, A and B or R and B) to build a minimum cost caching architecture. The cost are normalized according to the cost obtained when using only servers of type R. We first see that for all scalings, we can find a configuration which is less expensive than the reference configuration. As expected, when there is no scaling (scaling = 1), there is no benefit to use larger caches and the solutions with minimum cost are obtained while using the smallest ones (because they can fit more closely to the traffic levels). When the scaling factor decreases, there is more and more benefit to use larger caches (because they provided higher capacities with lower unit costs), even if their capacities are not fully used with a scaling of $\frac{3.5}{4}$, the best solution is obtained while using large servers A and small servers B. With a scaling of $\frac{2.5}{4}$, it becomes more advantageous to use only the two largest types of servers (A and R), while for a scaling of 3/4, these last two combinations seems relatively close in terms of cost. With proper cost and scaling values and capacity characteristics, it is hence possible to evaluate the most cost effective combination of pieces of equipment.

Conclusion From our results, we conclude that, in addition to QoS improvement, the economical efficiency of the network can be a driver towards caches deployment for some types of traffic. These results are resilient to many assumptions, such as caches cost or scaling factors. The choice of NetInf to ease cache deployment through content naming is hence here justified.

The current decrease in memory costs and the increase in users connectivity and bandwidth requirements strengthen this conclusion and extend the types of traffic which can be efficiently cached. We recall however that this analysis focuses on the global scale, and assumes that well-adapted business models will provide each player enough incentives to move towards the global optimum.

4.5.2 Orange Trace Analysis

The evaluation of NetInf can be separated into three different aspects: (1) the performance and scalability of the local and global NetInf architecture for providing access to published objects, (2) the performance and stability of the transport protocol used for transporting a requested content potentially consisting of multiple NDOs from the source(s) to the requester, and (3) the efficiency of in-network caching solutions including the capability of NetInf to find the best possible source. The evaluation of these aspects should rely on scenarios that illustrate the benefits of NetInf, are challenging for NetInf, and are realistic with respect to traffic patterns observed in today's network. In order to get a basic understanding how to define these scenarios and potentially also to define exactly one specific scenario, the trace files provided by Orange are analysed.

These trace files contain a collection of all HTTP requests of the Orange network for one month including among others the request time, the MIME type and the volume of the requested object. Consequently, the traces contain information on the number of requested HTTP objects, the size of the HTTP objects that may be translated to a number of NDOs, the request rate, and the popularity of objects including even the temporal variation of object popularity. So, the traces are very useful to model the customer behavior for NetInf evaluation. What is not possible to deduct from the traces is of course the publishing activity for which a separate model will be required.

The current activity is focusing on the evaluation of caching (content replacement) and cache placement strategies. The essential data to be extracted from the traces are the number of objects, the size of objects, and the request rate for objects that also covers the popularity of objects varying over time. In the following, details on the available data is described and the current status of the evaluation which focuses on object popularity is given.

Available data set Facts and figures: The data set provided by Orange comprises 40 days (from 2011/12/16 to 2012/01/25), which are separated in 952 files of 1h (2-3 TB each).
The data was provided in the following format:

Type	Request Type (GET or POST for http) + RTSP
ANS	ANS from server, as present in the packet.
Hour	Time stamp for the answer (CET)
CustomerID	Anonymized Customer ID; unique over the period for the same customer
CustomerIP	Current (dynamic) IP address of the customer
ServerIP	IP address of the server
URL	URL as present in the http request
Host	Host as present in the http request
Referer	Referer as present in the http request
Content_Type	Mime-Type present in the answer packet (optional)
ContentLength	Length of the content part (optional)
LocInfo	Anonymized name of the base station.

Two sample records are:

```
GET  302  Wed Jan 25 16:59:59 2012  0fa6edaf1ceee123  10.230.43.152   46.33.76.67
          /FF/iPhone/getListe.php?token=5c6aefca7796cbb3  www.francefootball.fr
          text/html   316  308a0276b86dde32

GET  200  Wed Jan 25 16:59:59 2012  e146655aeb5d9a2c  10.224.234.248  192.168.10.100
          http://m.facebook.com/home.php?refid=7&_rdr     m.facebook.com
          text/html        7e26c6d825252418
```

We used different subsets to evaluate the data:

- Hour: a subset of 1 hour (from Jan 25 15:59:56 to 16:59:59 2012) with around 133.000 distinct base station/cell IDs, 1.34 mio distinct and active users, and 46 mio GET/POST requests;
- Day: a subset of 24 hours (from Jan 23 23:59:42 to Jan 24 23:59:59 2012) with around 178.000 distinct base station/cell IDs, 4.4 mio distinct and active users, and 833 mio GET/POST requests.

Reports The following reports could be generated from the dataset:

1. Popularity of certain domains/websites;
2. Content popularity, i.e. access statistics for certain content, e.g. YouTube videos ;
3. User activity (of all users or a particular user) during 1 day/week/month to find out peak hours, frequency and duration of idle/active periods, . . . ;
4. Activity seen at a particular base station to learn about the distribution over time, the load of base stations (number of users served), . . . ;
5. User mobility (How long is a user served by a certain base station? How often/frequent is he changing the cell?) in order to retrieve a mobility pattern of users ;
6. Rough idea of locality of base stations, i.e. if a user is connected to two base stations within a delta t, those base station are likely adjacent. This information can be used to retrieve location-aware reports of content popularity, user activity, ...

Unfortunately, quite late we were informed about a crucial constraint in the trace data. Due to "technical choices", information on clients and their localisation is only updated when they switch on their cellphone, or when they change from 2G to 3G or 3G to 2G. This has 2 meaningful consequences for the accuracy of our assessment:

1. There is no change in the base station ID as long as a user stays connected in the 3G network (or respectively the 2G network).
2. When a probe (re)starts, it has no client and base station information. Hence, all connections are allocated to a default client and default base station. This is slowly corrected, as users get correctly identified. The default user and base station have been identified and are omitted in all of our evaluations.

Popularity of domains Looking at the popularity, we evaluated the popularity of domains, as well as YouTube video content. For the popularity of domains, `www.google.com` ranks first with 2,681,153 requests within one hour (dataset "1 hour"). We observed 206,122 unique domain names, from which 157,089 (76%) had more than 2 hits. Looking at the frequency-rank-plot for domain names (Figure 4.36, there is an almost straight line in the log-log-plot, indicating a Zipf-distribution with heavy tail.

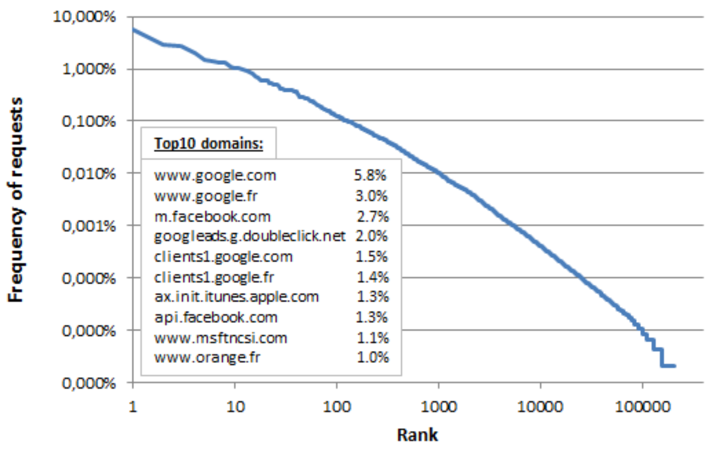


Figure 4.36: Rank frequency plot for domain names within "1 hour"

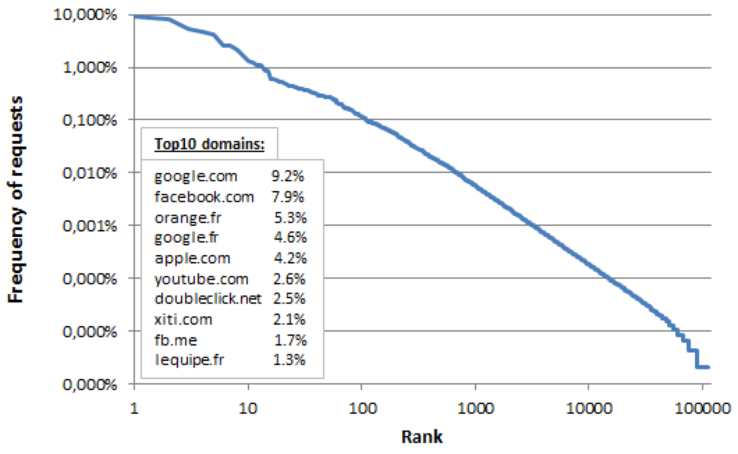


Figure 4.37: Rank frequency plot for domain names (top and second level only)

Performing an evaluation of second-level domain names only, the shape of the frequency-rank-plot is similar to full domain names. `Google.com` still ranks first with 4,250,697 requests. In total 112,098 unique second-level domain names were seen, with 90,136 (80%) having more than 1 hit within the observed period. The popularity distribution for second-level domains is shown in Figure 4.37.

Content popularity Looking at the content popularity of YouTube videos in the dataset "1 hour", we observed requests for 50,074 unique video IDs (out of 67.758 requests). Out of these videos, 1,347 (3%) videos were only requested once, and 50% of the videos had less than 5 requests. Looking at the popularity over the video rank, there is again an almost straight line in the log-log-plot (see Figure 4.38), which is a distinguished feature of a Pareto distribution and indicates a heavy-tailed distribution.

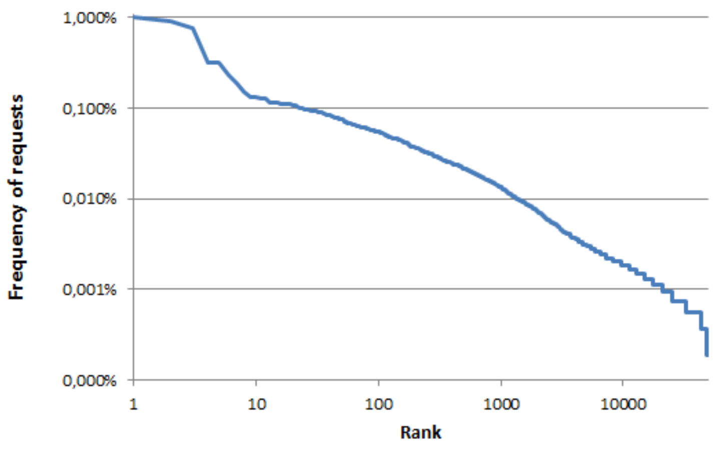


Figure 4.38: Rank frequency plot for YouTube videos within “1 hour”

The Pareto principle (also called 80-20 rule) states (translated into an YouTube context), that if k users have already watched a video, then the rate of other users watching the same video is proportional to k. This correlation results from several reasons and is also called the rich-get-richer principle. The more views a video has, the more people talk about it, and the more other people tend to watch the video. Automated recommendations, links in social networks and on websites, as well as searching for or sorting by the “most viewed” or “top-rated” videos amplify this phenomena.

Our observation coincides with other similar measurement in literature. The paper “I Tube, You Tube” [55], for example, presents trace-driven analysis of video popularity distributions. The main results from this study are: YouTube seems to be highly skewed towards popular files, i.e. little content is extremely popular, and the popularity distribution exhibits a power-law waist with a truncated tail. HP Laboratories in Palo Alto performed a study on the traffic characterization of YouTube traffic [56]. A similar curve was also published by two Google employees in 2012 [57]. They state, that 50% of the videos had less than 50 views over the period analysed.

As a consequence of this characteristic, by caching only 10% of the long-term popular videos, a cache can serve around 80% of requests [55]. In contrast to that, the remaining 90% of the videos account for very few requests. A similar behaviour can also be observed for other video data. An analysis of PowerInfo, a large VoD system in China, showed that about 40% of videos requested daily are different from the long-term popular videos [58].

Activity The overall activity in the network for one working day shows the following shape (see also Figure 4.39): The number of requests is low during night hours (1 am to 5 am). Then, it is increasing steadily until 11 am, with a small local peak around 10 am. Big peaks can be observed at noon (between 12 pm and 2 pm), around 5:30 pm, and in the late evening (from 20:30 pm to 23 pm).

The activity of different users is again Zipf-distributed with a few users being extremely active, but many other users showing much less activity during one day. A frequency-rank-plot showing the activity of users can be found in Figure 4.40.

Looking at the activity of selected users within one day, you can observe a quite different behaviour for different users. Figure 4.41 shows the activity of the Top3 most active users, as well as the 1000st and 5000st most active user. The most active user, for example, seems to be “nocturnal”, doing most requests during midnight and 7 am, which is in contrast to the overall network activity described above. Considering its behaviour and activity it might be a web crawler. Also, the second most active user has a high request rate during night, a peak around 8 am, and medium request rate during the day. User 3 is making requests only in the afternoon and evening, similar to user 5000

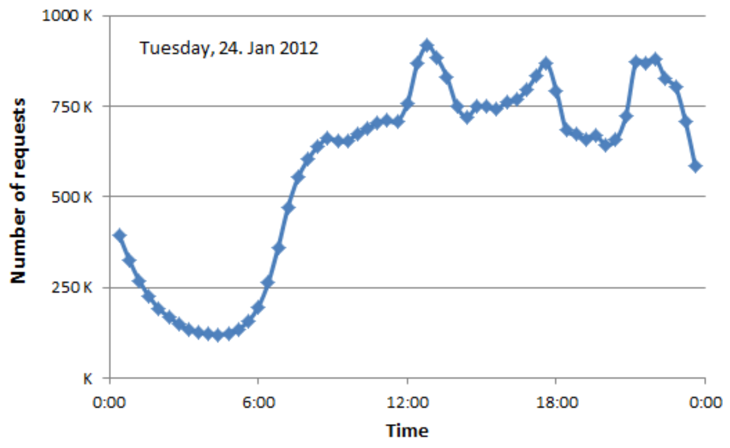


Figure 4.39: Overall activity on Tuesday, 24th January 2012

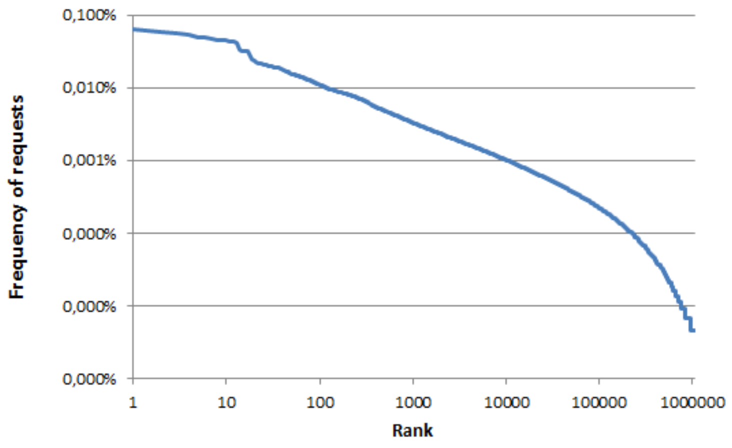


Figure 4.40: Rank frequency plot for user activity within "1 hour"

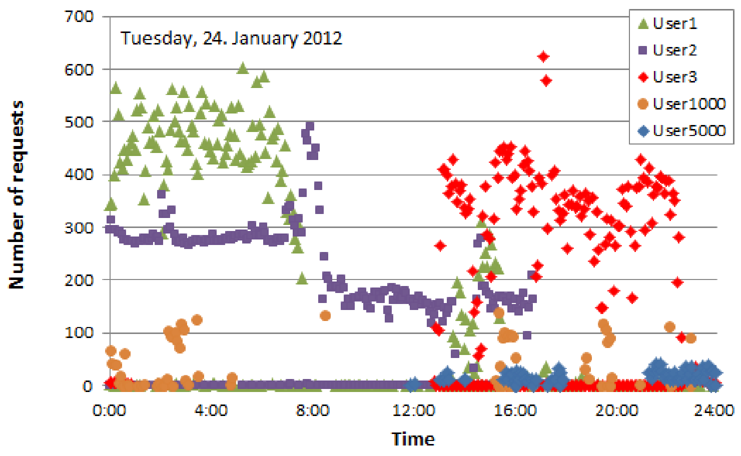


Figure 4.41: Activity of selected users on Tuesday, 24th January 2012

who is also active in the afternoon and night, yet at a much lower request rate. In contrast to that, User 1000 is showing a more peaky behaviour, with peaks around 3:30 pm, 7 pm, 8 pm, 0:30 am, and 3 am.

Locality information The locality of base stations is not disclosed in the provided dataset due to apparent data protection issues. However, for different reports, knowing about a rough distribution of base stations would be beneficial, e.g. to be able to assess regional content popularities. That is why we are working on partially re-engineering the adjacency of base stations. We are not interested in exactly locating base stations, or mapping the network to a 2D map of France, which even with high effort in all probability would not be possible. We are looking for clusters of base stations, to roughly learn about regions, in order to e.g. evaluate the benefit of regional caching.

In order to detect adjacent base stations, the following approach was taken: For each user in the dataset all records of the selected users are parsed in chronological order. If the user made two sequenced requests from two different base stations within a short period of time Δt (in the order of minutes), we can assume that these base stations are close to each other, and the user moved from the first base station to the second. Otherwise, if several minutes/hours are in between two sequenced requests are observed, the user could have moved large distances during that time and the base stations may be far away from each other. We then define a graph, with base stations being the nodes, and links between two base stations if at least one user made sequenced requests from two different base stations within a fixed time period Δt. In the following, we also refer to these links as "handovers".

In the database report, we considered different periods Δt (e.g. 1, 2, 4 minutes). Each link was weighted (i.e., defining its length) based on the shortest observed Δt. The number of observations, i.e. how frequent users moved from one base station to another base station, is defining the line width of the link in the plot. The weight of the node (i.e. its square footage) is defined by the degree of the node (i.e its number of links).

The resulting network graph is layouted with *neato*, a GraphViz tool. Plots can be found in Figures 4.42 and 4.43.

Unfortunately, due to an initial wrong understanding of the LocInfo mentioned earlier, the assessment of the locality information is not as accurate as expected and many actual handovers are missing as the LocInfo is not updated for intra-technology handovers. As a result, areas with good 3G coverage (or only 2G coverage) cannot be evaluated as users only switch from 3G to 3G (or 2G to 2G). Moreover, default LocInfo values had to be omitted from the dataset.

Orange trace visualisation Figures 4.44 and 4.45 are initial plots of visualisation of the Orange data files. The functionality is similar to Wordle by Google, however the internal data structures allow for more elaborate visualisations which will be shown during the final software day. The graphics were done with the Processing package.

Conclusion At the current stage, data on the popularity of domains, content popularity, and locality information has been extracted from the trace files. The popularity of domains is useful for evaluating the global NRS since this gives an indication on the potential clustering of requests to routing hints which is a key characteristic of locally requested global NDOs with significant impact on scalability and performance of the system. Content popularity, the resulting request rate per HTTP object and consequently also per related NDO is essential for evaluating caching and cache placement strategies but also for evaluating trade-offs between forwarding popular requests according to a routing table and referring to the NRS for less popular requests. Locality information that unfortunately could not be fully extracted from the available data is considered as extremely helpful for analyzing the differences in popularity of neighboring regions served by different caches.

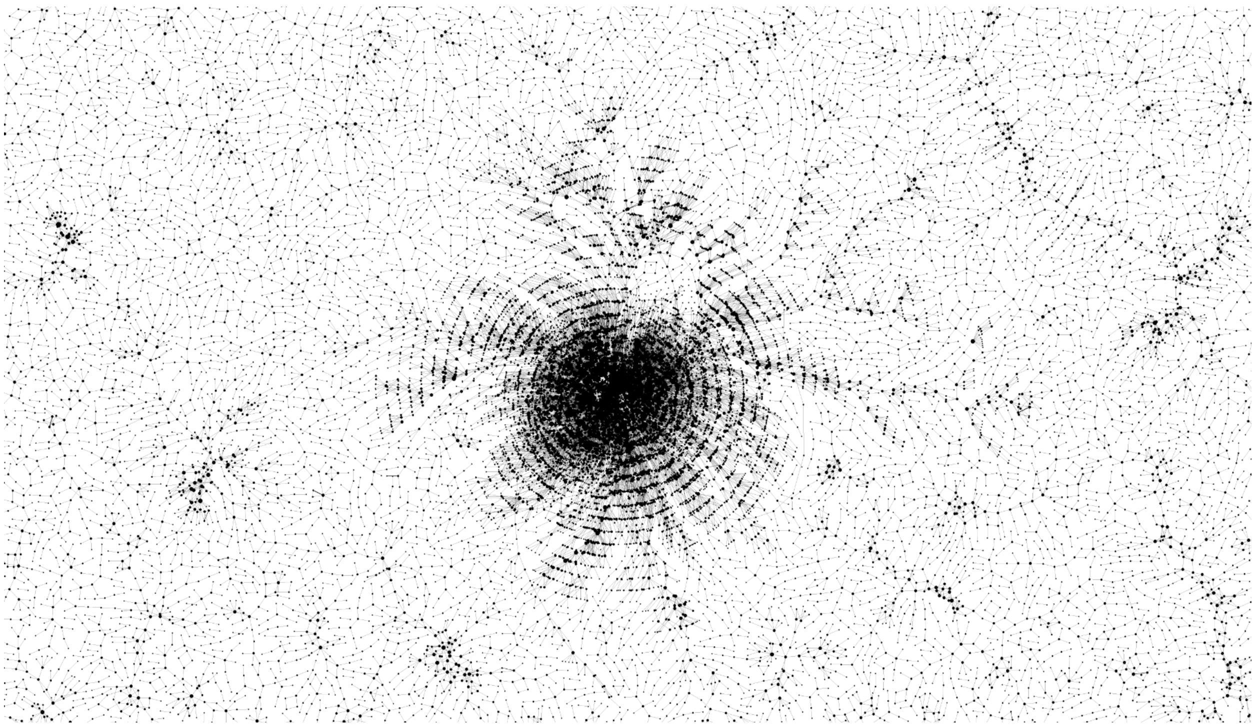


Figure 4.42: Network graph showing clusters of base stations

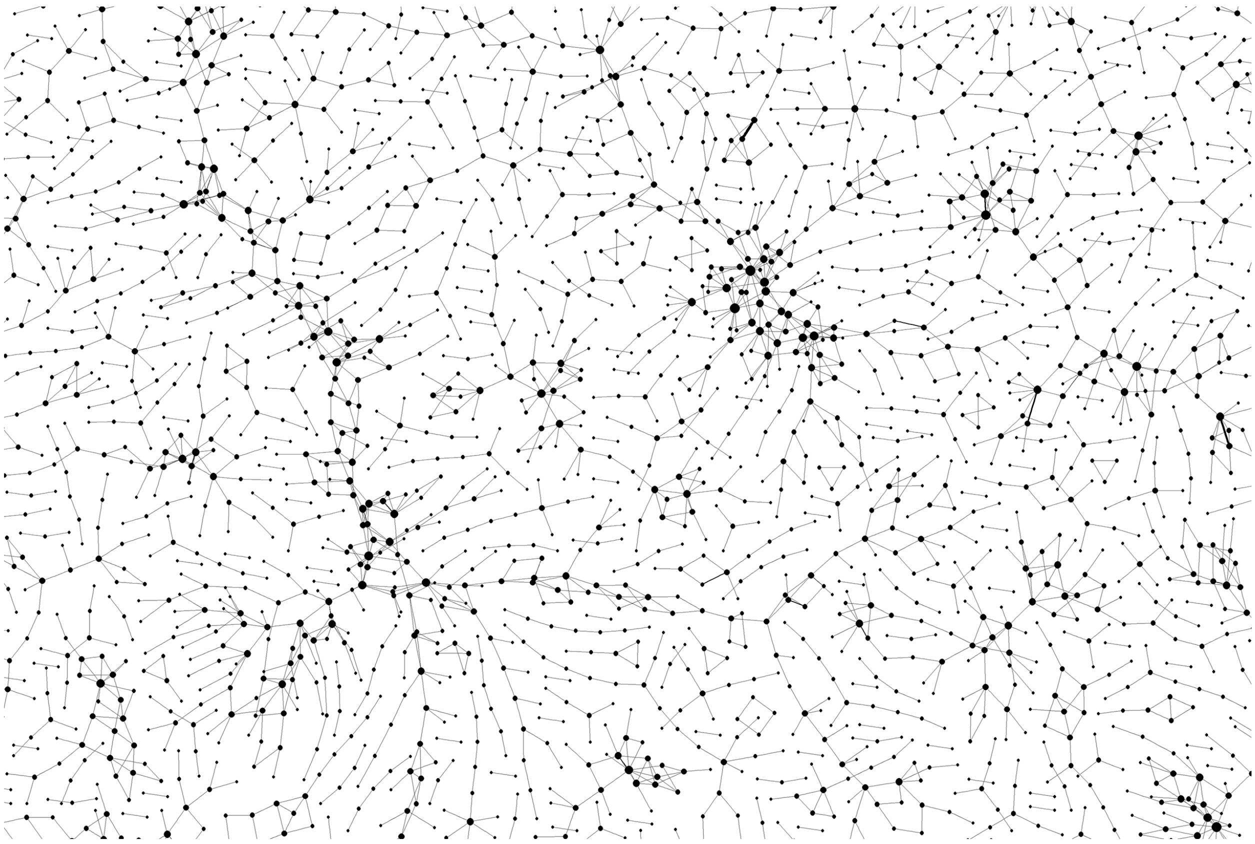
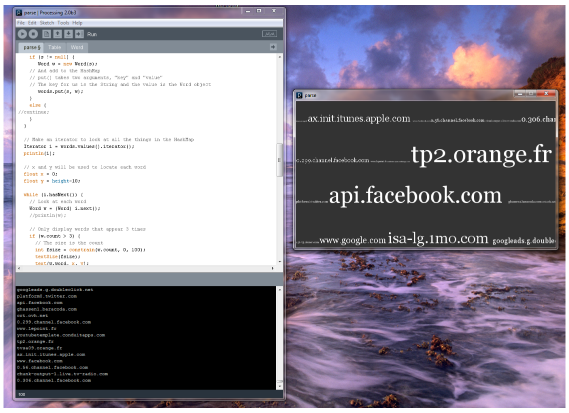


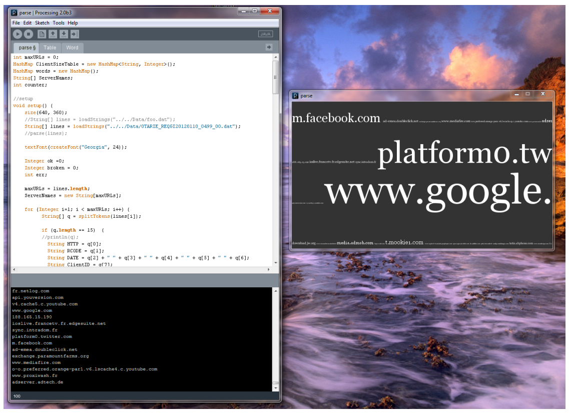
Figure 4.43: Details of the network graph showing clusters of base stations



```
    if (s != null) {
      Word w = new Word(s);
    // And add to the HashMap
    // put() takes two arguments, "key" and "value"
    // The key for us is the String and the value is the Word object
      words.put(s, w);
    }
    else {
//continue;
    }
  }

  // Make an iterator to look at all the things in the HashMap
  Iterator i = words.values().iterator();
  println(i);

  // x and y will be used to locate each word
  float x = 0;
  float y = height-10;

  while (i.hasNext()) {
    // Look at each word
    Word w = (Word) i.next();
    //println(w);

    // Only display words that appear 3 times
    if (w.count > 3) {
      // The size is the count
      int fsize = constrain(w.count, 0, 100);
      textSize(fsize);
      text(w.word, x, y);
```

```
googleads.g.doubleclick.net
platform0.twitter.com
api.facebook.com
ghassen1.baracoda.com
crt.ovh.net
0.299.channel.facebook.com
www.lepoint.fr
youtubetemplate.conduitapps.com
tp2.orange.fr
tvsa09.orange.fr
ax.init.itunes.apple.com
www.facebook.com
0.56.channel.facebook.com
chunk-output-1.live.tv-radio.com
0.306.channel.facebook.com
```

Figure 4.44: Step 1 in visualisation



```
int maxURLs = 0;
HashMap ClientSizeTable = new HashMap<String, Integer>();
HashMap words = new HashMap();
String[] ServerNames;
int counter;

//setup
void setup() {
    size(640, 360);
    //String[] lines = loadStrings("../../Data/foo.dat");
    String[] lines = loadStrings("../../Data/OTARIE_REQGI20120110_0499_00.dat");
    //parse(lines);

    textFont(createFont("Georgia", 24));

    Integer ok =0;
    Integer broken = 0;
    int err;

    maxURLs = lines.length;
    ServerNames = new String[maxURLs];

    for (Integer i=1; i < maxURLs; i++) {
          String[] q = splitTokens(lines[i]);

          if (q.length == 15)  {
          //println(q);
            String HTTP = q[0];
            String RCODE = q[1];
            String DATE = q[2] + " " + q[3] + " " + q[4] + " " + q[5] + " " + q[6];
            String ClientID = q[7];
```

```
fr.netlog.com
api.youversion.com
v4.cache5.c.youtube.com
www.google.com
188.165.15.190
ioslive.francetv.fr.edgesuite.net
sync.intradom.fr
platform0.twitter.com
m.facebook.com
ad-emea.doubleclick.net
exchange.paramountfarms.org
www.mediafire.com
o-o.preferred.orange-par1.v6.lscache4.c.youtube.com
www.proxiwash.fr
adserver.adtech.de
```

Figure 4.45: Step 2 in visualisation

Popularity models considering these characteristics are essential for evaluating collaborative caching strategies.

The trace data analysis will continue for the rest of the project and the obtained results will serve as a basis for the continued evaluation and experimental setup. Concretely, we plan to evaluate the global NRS, the proposed content and service placement strategies (in particular considering regional popularity information), and to further profile and model the activity of mobile users. The benefits of the analysis are not only in terms of concrete number from the trace data provided by Orange but also in the tools developed for analyzing the traces. It is planned to also use these tools to extract values from similar trace files which are either available publicly or internally for partners representing operators.

5 Conclusion

This document has summarised the NetInf architecture and recent updates to it that have been prompted by the prototyping and evaluation activities in the SAIL project — which we have also described. Amongst the different ICN research approaches that exist to date, NetInf is the only one that is actually pursued by a substantial industrial consortium of leading global vendors and operators. It is the only activity that has opened up its work through publishing specifications and releasing different inter-operable Open Source Software implementations.

Developing a consistent approach with a large group of industry and academic partners is naturally a non-trivial endeavour, considering different commercial interests, research agenda and pet topics. On the other hand, technical diversity can also be a virtue, potentially leading to an approach that considers different requirements and constraints.

SAIL NetInf has advanced the 4WARD foundations on naming, object models and name resolution to a network system architecture that supports heterogeneous networks as well as migration from today's networks to ICN-dominated deployments, also taking noteworthy progress in the state of the art in the field into account, such as the emerged CCN approach. The NetInf architecture, its design principles and the corresponding specification have been the basis for different system designs and prototype implementations.

Assessing the quality of an architecture is not easy as evaluating design decisions often requires experience with running systems and deployment. In SAIL, most of the prototype implementations inter-operate on the basis of the NetInf protocol and the NetInf naming scheme, such as the OSS NetInf software, OpenNetInf, the Android client, the NRS Rendezvous server, and the NNRP router platform. Experience in Internet standardisation, i.e., in the IETF, has shown that basing specification on experience with running code and actual experiments leads to better results. Considering different requirements and designing support for heterogeneity also leads to evolvability and thus a more future-proof design.

Technical discussion Technically, the thesis for NetInf is that non-hierarchically structured names on the ICN network layer are beneficial — and that the approach can actually scale to an Internet infrastructure level. Intuitively, hierarchically structured names look like a natural approach to ICN naming, however, research in SAIL and other projects have shown that:

- It is not wise to tie organisational names to names on the network layer (and to assume some topological relevance of those names) because network topologies do constantly change due to mobility and multi-homing — and because organisational structure also change over time. Hence, it is better to separate such organisational information from the network layer.
- Name-content binding validation is a fundamentally required feature for ICN to prevent denial-of-service attacks and other pathological attacks. Assuming a trust hierarchy with a single-rooted trust chain as promoted by CCN's hierarchical naming scheme to provide name-content-binding validation services is essentially equivalent to employing a single PKI for network layer interactions on the Internet — totally unrealistic and undesirable in a world of federated networks.

The NetInf approach is to define a common naming format that can enable fundamental content-hash-based name-content-binding validation and so-called "self-certifying" names — without any additional infrastructure dependencies. Network layer names have neither organisational nor application context significance — a good compromise between simplicity and ICN naming security.

We have discussed the scalability and performance trade-offs for using such names in the analytic and simulation-based work for HSkip, MDHT and GIN. The results are promising, and we believe that the next step for this technology would be real experiments on global scale.

The development of NetInf receiver-driven transport strategies on top of the Convergence Layer in running code (i.e., modules for NNRP) has illustrated NetInf's benefits and good performance for ICN object retrieval. In single-source end-to-end scenarios, TCP-like performance can be achieved, and in multi-source scenarios, NetInf is able to leverage the path diversity and scale up throughput accordingly.

NetInf in SAIL In the SAIL project, NetInf has been used in different integrated prototypes, such as the OConS Multipath work that has leveraged NNRP for ICN multipath experiments. Moreover, NNRP has been used in an elastic ICN-based content distribution experiment that is using Cloud Networking to dynamically scale out a network of NetInf nodes.

Impact So far, NetInf's impact is unique among the existing ICN-related EU projects. The NetInf naming specification has seen broad review by many experts in the IETF , has passed IESG review and is awaiting publication as a standards track RFC. The SAIL naming technology is considered by the IoT community (IETF CORE WG) and P2P community for adoption — thus enabling interoperability for information-centric communication across different application domains.

The NetInf Open Source software has been published under the Apache-2.0 OSS license and has been accessed many researchers and experimenters world-wide.

Interesting areas for potential future work include active information objects (execution platforms for objects that represent services or locally generated content), access rights, and NDO life-cycle management (e.g., object deletion). Moreover, we recommend to further investigate business incentives for ICN adoption (see also SAIL's work package on Impact and Collaboration Enabling that has started some initial work on that).

List of Abbreviations, Acronyms, and Definitions

ADU	Application Data Unit
AIMD	Additive Increase Multiplicative Decrease
AN	Access Node
API	Application Programming Interface
AS	Autonomous System
BF	Bloom Filter
BGP	Border Gateway Protocol
BPQ	Bundle Protocol Query
BP	Bundle Protocol
BPA	Bundle Protocol Agent
CAM	Content Address Memory
CDNI	Content Delivery Network Interconnection
CDN	Content Delivery Network
CL	Convergence Layer
CLB	Constrained Load Balancing
CloNe	Cloud Networking
DFZ	Default Free Zone
DHT	Distributed Hash Table
DIS	Destination ID Stack
DNS	Domain Name System
DRAM	Dynamic Random Access Memory
DREC	Dictionary Record
DTNRG	Delay-Tolerant Networking Research Group
DTN	Delay- and Disruption-Tolerant Networking
DTN2	Reference implementation of the DTN BP

EID	Endpoint Identifier
EwLC	Event with Large Crowds
GIN	Global Information Network
GINP	Global Information Network Protocol
HDD	Hard Disk Drive
HSkip	Hierarchical SkipNet
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
ICN	Information-Centric Networking
ICS	Intermediate Cache Server
ID	IDentifier
I-D	Internet Draft
IETF	Internet Engineering Task Force
IOPS	Input/Output Operations Per Second
IP	Internet Protocol
IRTF	Internet Research Task Force
ISP	Internet Service Provider
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LCS	Leaf Cache Server
LLC	Late Locator Construction
LP	Level Probability
MAC	Media Access Control
MDHT	Multilevel DHT
MDB	Mobile Database
MDS	Mobile Data Storage
MIME	Multipurpose Internet Mail Extension
MSF	MDHT Stretch Factor

NAT	Network Address Translation
NCS	NetInf Content Server
NDO	Named Data Object
NE	Network Edge
NHT	Next Hop Table
nid	networkID
NNRP	NEC NetInf Router Platform
NRS	Name Resolution System
NetInf	Network of Information
OConS	Open Connectivity Services
OSPF	Open Shortest Path First
OTT	Over-the-Top
PDU	Protocol Data Unit
PHP	PHP: Hypertext Preprocessor - a recursive acronym
PKI	Public Key Infrastructure
POP	Point of Presence
PRR	Peak Rate Reduction
PSIRP	Publish-Subscribe Internetworking Routing Paradigm
QoE	Quality of Experience
QoS	Quality of Service
QRS	QR-code Server
REX	Resolution EXchange
RFC	Request For Comments
RTT	Round Trip Time
SAIL	Scalable and Adaptive Internet Solutions
SDN	Software Defined Networking
SLA	Service Level Agreement
SRAM	Static Random Access Memory
SRV	Service
SSD	Solid State Disk

TAG	Technical Architecture Group
TCP	Transmission Control Protocol
TTL	Time To Live
UDP	User Datagram Protocol
UGC	User Generated Content
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VoD	Video on Demand
W3C	World Wide Web Consortium
WG	Working Group
WSGI	Python Web Server Gateway Interface

List of Figures

List of Tables

Bibliography

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

[2] S. Farrell, D. Kutscher, C. ChristianDannewitz, B. Ohlman, A. Keranen, and P. Hallam-Baker. Naming Things with Hashes. Internet-Draft draft-farrell-decade-ni-10, Internet Engineering Task Force, August 2012. Work in progress.

[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] SAIL Project. NetInf content delivery and operations (D.B.2). Deliverable D-3.2, SAIL EU FP7 Project, 2012. FP7-ICT-2009-5-257448/D-3.2.

[6] Christian Dannewitz, Dirk Kutscher, Börje Ohlman, Stephen Farrell, Bengt Ahlgren, and Holger Karl. Network of information (NetInf) – an information-centric networking architecture. *forthcoming*, 2013.

[7] Ali Ghodsi, Teemu Koponen, Jarno Rajahalme, Pasi Sarolahti, and Scott Shenker. Naming in content-oriented architectures. In *Proc. ACM SIGCOMM Workshop on Information-Centric Networking*, pages 1–6, New York, NY, USA, 2011. ACM.

[8] David D. Clark, John Wroclawski, Karen R. Sollins, and Robert Braden. Tussle in cyberspace: Defining tomorrow's Internet. In *Proc. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 347–356, New York, NY, USA, 2002. ACM Press.

[9] Anders Eriksson and Börje Ohlman. Dynamic internetworking based on late locator construction. In *10th IEEE Global Internet Symposium*, May 2007.

[10] D. Kutscher, S. Farrell, and E. Davies. The NetInf Protocol. Internet-Draft draft-kutscher-icnrg-netinf-proto-00, Internet Engineering Task Force, October 2012. Work in progress.

[11] Matteo D'Ambrosio, Christian Dannewitz, Holger Karl, and Vinicio Vercellone. MDHT: A hierarchical name resolution service for information-centric networks. In *Proc. ACM SIGCOMM Workshop on Information-centric Networking*, pages 7–12, New York, NY, USA, August 2011. ACM.

[12] Christian Dannewitz, Matteo D'Ambrosio, Holger Karl, and Vinicio Vercellone. Hierarchical DHT-based name resolution for information-centric networks. *Computer Communications*, Special Issue on Information-Centric Networking, 2012. (under review).

[13] Christian Dannewitz, Holger Karl, and Aditya Yadav. Report on locality in DNS requests – Evaluation and impact on future Internet architectures. Technical Report TR-RI-12-323, University of Paderborn, Paderborn, Germany, July 2012.

[14] P. McDaniel and S. Jamin. A scalable key distribution hierarchy. Technical report, University of Michigan. Department of Electrical Engineering and Computer Science, 1998.

[15] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[16] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A scalable overlay network with practical locality properties. In *Proc. USENIX Symposium on Internet Technologies and Systems (USITS)*, page 9, Berkeley, CA, USA, 2003. USENIX Association.

[17] M. Särelä, C. R. Esteve, A. Zahemszky, P. Nikander, and J. Ott. Bloomcast: Security in bloom filter based multicast. In *Nordsec 2010. Joint work with FI-SHOK project funded by TEKES.*, 2010.

[18] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[19] E. Rescorla. HTTP Over TLS. RFC 2818, Internet Engineering Task Force, May 2000.

[20] M. Nottingham and E. Hammer-Lahav. Defining Well-Known Uniform Resource Identifiers (URIs). RFC 5785, Internet Engineering Task Force, April 2010.

[21] http://www.bittorrent.com/.

[22] Christian Dannewitz, Matthias Herlich, Eduard Bauer, Matthias Becker, Frederic Beister, Nadine Dertmann, Razvan Hrestic, Michael Kionka, Mario Mohr, Matthias Mühe, Deepika Murali, Felix Steffen, Sebastian Stey, Eduard Unruh, Qichao Wang, and Steffen Weber. OpenNetInf documentation - design and implementation. TechReport TR-RI-11-314, University of Paderborn, Sep. 2011.

[23] Bernard Aboba, Jon Peterson, Henning Schulzrinne, and Hannes Tschofenig. The future of web applications or how to move into the post standardization area. Position Paper to the RTC Web Workshop, October 2010. http://rtc-web.alvestrand.com/home/papers/hannes-post-standards.pdf.

[24] Jarno Rajahalme. *Inter-Domain Incentives and Internet Architecture*. PhD thesis, Aalto University, Aug. 2012. `http://urn.fi/URN:ISBN:978-952-60-4728-7`.

[25] Amogh Dhamdhere and Constantine Dovrolis. Can ISPs be Profitable Without Violating "Network Neutrality"? In *NetEcon'08*, pages 13–18. ACM, 2008.

[26] Lixin Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Trans. Netw.*, 9(6):733–745, Dec. 2001.

[27] R. Johari and J.N. Tsitsiklis. Routing and Peering in a Competitive Internet. *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, 2:1556–1561 Vol.2, Dec. 2004.

[28] William B. Norton. Internet Service Providers and Peering. Draft 2.8, `http://www.nanog.org/papers/isp.peering.doc`, August 2002.

[29] Xenofontas Dimitropoulos, Paul Hurley, Andreas Kind, and Marc Stoecklin. On the 95-Percentile Billing Method. In Sue Moon, Renata Teixeira, and Steve Uhlig, editors, *Passive and Active Network Measurement*, volume 5448 of *Lecture Notes in Computer Science*, pages 207–216. Springer Berlin / Heidelberg, 2009.

[30] P. Faratin, David D. Clark, P. Gilmore, S. Bauer, A. Berger, and W. Lehr. Complexity of Internet Interconnections: Technology, Incentives and Implications for Policy. In *TPRC. Proceedings*, 2007.

[31] Y. Rekhter and T. Li. A Border Gateway Protocol (BGP-4). RFC 1771, IETF, Mar. 1995.

[32] Geoff Huston. The Flat World of BGP. Presentation at RIPE 63, October 2011. `http://ripe63.ripe.net/presentations/61-2011-10-31-bgp2011.pdf`.

[33] Jarno Rajahalme. Incentive-Informed Inter-Domain Multicast. In *INFOCOM IEEE Conference on Computer Communications Workshops*, Global Internet Symposium, San Diego, CA, USA, March 2010.

[34] Jarno Rajahalme, Mikko Särelä, Pekka Nikander, and Sasu Tarkoma. Incentive-Compatible Caching and Peering in Data-Oriented Networks. In *CoNEXT ReArch'08*. ACM, Aug. 2008.

[35] SAIL Project. Evaluation of the business models. Deliverable D-1.8, SAIL EU FP7 Project, 2011. FP7-ICT-2009-5-257448/D-1.8.

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

[37] G. Carofiglio, M. Gallo, and L. Muscariello. Icp: Design and evaluation of an interest control protocol for content-centric networking. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 304 –309, march 2012.

[38] Christian Dannewitz, Matthias Herlich, and Holger Karl. OpenNetInf – Prototyping an information-centric network architecture. In *Proc. IEEE LCN – Workshop on Architectures, Services and Applications for the Next Generation Internet (WASA-NGI)*, October 2012. (to be published).

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

[40] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050, Internet Engineering Task Force, November 2007.

[41] S. Farrell, A. Lynch, D. Kutscher, and A. Lindgren. Bundle Protocol Query Extension Block. Internet-Draft draft-irtf-dtnrg-bpq-00, Internet Engineering Task Force, May 2012. Work in progress.

[42] S. Symington. Delay-Tolerant Networking Metadata Extension Block. RFC 6258, Internet Engineering Task Force, May 2011.

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

[44] 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.

[45] Junghoo Cho and Hector Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology (TOIT)*, 3(3):256–290, August 2003.

[46] Abdulhalim Dandoush, Sara Alouf, and Philippe Nain. Lifetime and availability of data stored on a P2P system: Evaluation of recovery schemes. Research Report RR-7170, INRIA, January 2010.

[47] Moritz Steiner and Ernst W. Biersack. Where is my peer? Evaluation of the Vivaldi network coordinate system in Azureus. In *Proc. 8th IFIP-TC 6 Networking Conference*, pages 145–156, Berlin, Heidelberg, 2009. Springer-Verlag.

[48] Dmitri Krioukov, k c claffy, Kevin Fall, and Arthur Brady. On compact routing for the internet. *SIGCOMM Comput. Commun. Rev.*, 37(3):41–52, July 2007.

[49] Ittai Abraham, Cyril Gavoille, Dahlia Malkhi, Noam Nisan, and Mikkel Thorup. Compact name-independent routing with minimum stretch. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '04, pages 20–24, New York, NY, USA, 2004. ACM.

[50] Bengt Ahlgren, Börje Ohlman, Erik Axelsson, and Lars Brown. Subversion over OpenNetInf and CCNx. In *4th International Workshop on Architectures, Services and Applications for the Next Generation Internet (WASA-NGI-IV)*, Bonn, Germany, October 4, 2011. In conjunction with IEEE LCN.

[51] Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato. *Version Control with Subversion*. O'Reilly Media, 2004. ISBN 0-596-00448-6, online at http://www.svnbook.com/.

[52] 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.

[53] Ashish Goel and Pankaj Gupta. Small subset queries and bloom filters using ternary associative memories, with applications. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '10, pages 143–154, New York, NY, USA, 2010. ACM.

[54] J. M. Chambers, C. L. Mallows, and B. W. Stuck. A method for simulating stable random variables. *J. Amer. Statist. Assoc*, 71:340–344, 1976.

[55] M. Cha, H. Kwak, P. Rodriguez, Y.Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC '07)*, 2007.

[56] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. YouTube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC'07)*, 2007.

[57] M. Wattenhofer A. Brodersen, S. Scellato. YouTube around the world: geographic popularity of videos. In *Proceeding of WWW 2012*, pages 241–250, 2012.

[58] H. Yu, D. Zheng, B.Y. Zhao, and W. Zheng. Understanding user behavior in large-scale video-on-demand systems. In *Proceeding of ACM Eurosys*, 2006.