Objective FP7-ICT-2009-5-257448/D.B.4

Future Networks

Project 257448

"SAIL – Scalable and Adaptable Internet Solutions"

D.B.4
(D-3.4) Prototyping and Evaluation

Date of preparation: **2013-03-07**
Start date of Project: **2010-08-01**
Project Coordinator: **Thomas Edwall**
Ericsson AB

Revision: **1.1**
Duration: **2013-02-28**

Document Properties

Document Number:	D.B.4
Document Title:	**Prototyping and Evaluation**
Document Responsible:	Dirk Kutscher (NEC)
Document Editor:	Petteri Pöyhönen (NSN)
Authors:	Bengt Ahlgren (SICS), Matteo D. Ambrosio (TI), Elwyn Davies (TCD), Anders E. Eriksson (EAB), Stephen Farrell (TCD), Claudio Imbrenda (NEC), Bruno Kauffmann (FT), Dirk Kutscher (NEC), Fabian Schneider (NEC), Anders Lindgren (SICS), Ian Marsh (SICS), Luca Muscariello (FT), Hugo Negrette (EAB), Börje Ohlman (EAB), Jean-François Peltier (FT), Karl-Ake Persson (EAB), Petteri Pöyhönen (NSN), Miguel Sosa (EAB), Patrick Truong (FT), Janne Tuononen (NSN), Vinicio Vercellone (TI)
Target Dissemination Level:	PU
Status of the Document:	Final Version
Version:	1.1

Production Properties:

Reviewers:	Benoit Tremblay, Gerald Kunzmann (DOCOMO)

Document History:

Revision	Date	Issued by	Description
1.0	2013-02-28	Petteri Pöyhönen	First complete version
1.1	2013-03-07	Petteri Pöyhönen	Small fixes and public version

Disclaimer:

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

Abstract:

This deliverable reports on the prototypes and testbeds that had been shown at a public demonstration event on *Future Media Distribution using Information Centric Networks*: http://www.sail-project.eu/future-media-distribution-information-centric-networks/.

Keywords:

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

Executive Summary

NetInf is the approach to Information-Centric Networking developed by the SAIL project. This deliverable is a so-called *prototype* deliverable and provides a report on the prototypes and testbeds that had been shown at a public demonstration event on *Future Media Distribution using Information Centric Networks*: http://www.sail-project.eu/future-media-distribution-information-centric-networks/.

Based on the NetInf architecture and protocols, SAIL has developed several prototypes for infrastructure nodes (NetInf routers and caches) and applications such as content distribution and information retrieval. SAIL has also developed a large-scale distributed testbed for NetInf that connects several partners sites and hosts thousands of NetInf nodes.

These systems have been shown in several live demonstrations and simulations at the mentioned public demonstration event. Regarding evaluating the system, SAIL focused on two main areas: 1) system scalability with respect to routing and name resolution and 2) performance with respect to data transport and caching.

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 enable multi-technology/multi-domain interoperability and to facilitate initial deployment and migration from today's networks to ICN. This deliverable is a so-called *prototype* deliverable[1], and it provides a report on the prototypes and testbeds that had been shown at a public demonstration event on *Future Media Distribution using Information Centric Networks*: http://www.sail-project.eu/future-media-distribution-information-centric-networks/.

The first project deliverable D.B.1 [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 [2], which has formulated the architecture invariants more specifically and which has advanced NetInf technologies such as routing and forwarding, transport protocols and caching. D.B.2 also defines a set of preferred use scenarios that have been further developed and investigated by the prototypes presented in this document.

The individual prototypes and their technical components as well as their evaluation have been described in the project deliverable D.B.3 [3] that also provided a comprehensive description of the overall NetInf architecture.

This deliverable is based on the NetInf architecture and protocols, SAIL has developed several prototypes for infrastructure nodes (NetInf routers and caches) and applications such as content distribution and information retrieval. SAIL has also developed a large-scale distributed testbed for NetInf that connects several partners sites and hosts thousands of NetInf nodes.

These systems have been shown in several live demonstrations and simulations at the mentioned public demonstration event. Regarding evaluating of the system, SAIL focused on two main areas: 1) system scalability with respect to routing and name resolution and 2) performance with respect to data transport and caching.

This document is reproducing the material shown at the demonstration event in Chapter 2. Appendix A provides the actual brochure published at the demonstration event.

[1] For prototype deliverables, the actual deliverable are the *prototypes*.

2 Live Event Prototypes

2.1 Network of Information (NetInf) Overview

Information-Centric Networking (ICN) is a promising approach for evolving the Internet towards an infrastructure that can provide an optimal service for accessing named data objects – one of the dominant applications today (Figure 2.1). In general, ICN is providing access to named data objects as a first class networking primitive and is leveraging unique naming techniques and ubiquitous in-network caching to provide more efficient and robust networking services than current approaches allow.

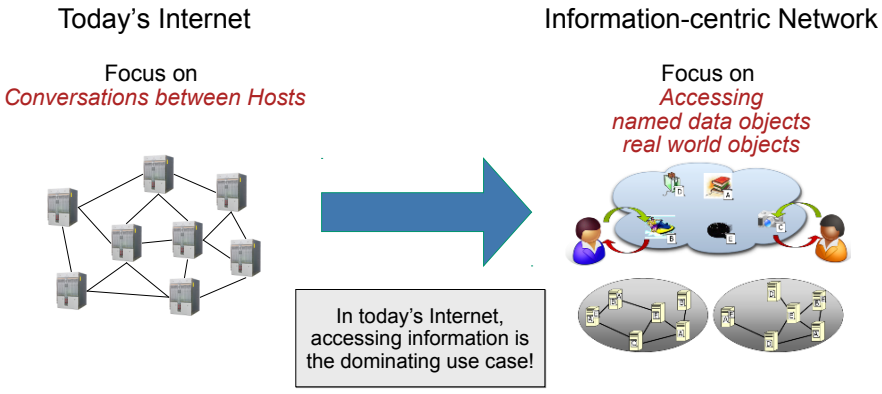


Figure 2.1: Vision: Moving towards ICN

The Scalable and Adaptive Internet Solutions (SAIL) project has been developing the NetInf approach (Figure 2.2) that is aiming at a highly scalable network architecture with particular support for robustness and reliability as well as at multi-technology/multi-domain interoperability. SAIL NetInf is putting particular emphasis on enabling networks that go beyond current de-facto architectures for broadband/mobile access and data center networks. While we want to support those deployment scenarios and their corresponding business requirements, we also want networks to go beyond inherited telco constraints and assumptions.

For example, ICN can be made to work with the existing network infrastructure, name resolution and security infrastructure – but that does not mean that all ICN networks should depend on such infrastructure. Instead, we want to leverage local, decentralised communication options to arrive at a solution that is easy to deploy at small scale and is able to extend to global scale but still resilient against network partitions, intermittent connectivity and potentially longer communication delays. Likewise, ICN is often characterised as a generalised content distribution approach, but in fact, has benefits beyond content distribution for example, better security properties through Named Data Object (NDO) security as well as better performance and robustness through in-network caching and localised transport strategies.

We believe that NetInf's going beyond next-generation Content Delivery Network (CDN) approach will finally result in a network that better accommodates current mass-market applications (for example for content distribution) and future mass-market applications such as smart-object communications in constrained networks. Key NetInf elements have been published as specifications, such as the NetInf protocol specification [4] - a conceptual specification of a NetInf Node-

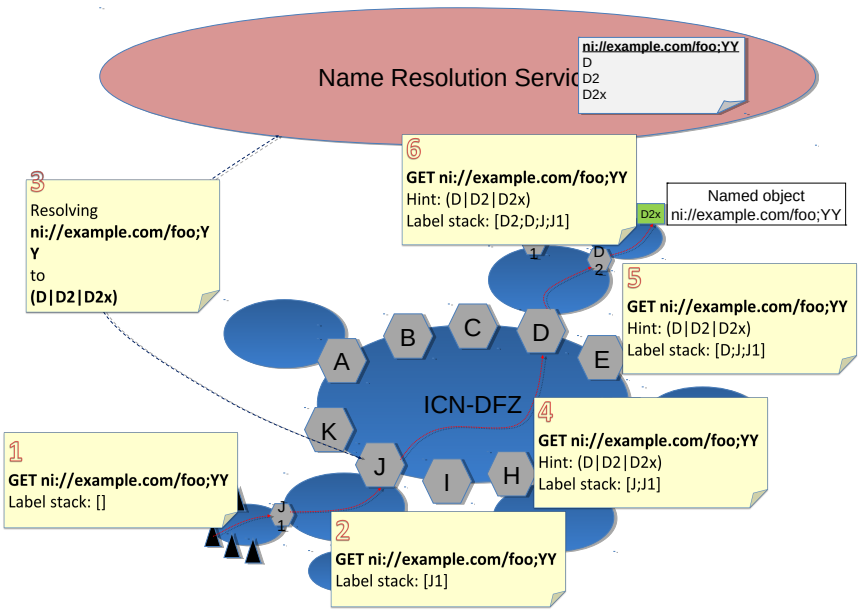


Figure 2.2: NetInf approach

to-Node communication protocol that includes an object model for 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 [5] and different Convergence Layer specifications. In the following, we are presenting different prototypes and evaluation scenarios that had been developed by the SAIL project, illustrating different aspects of the NetInf system.

NetInf approach shown in Figure 2.2 has the following main components:

- **NetInf Naming**: Identifying NDO and providing name-content binding validation and other security features.
- **NetInf Transport**: Information-Centric internetwork, transport protocols and Convergence Layer (CL) approach enabling migration and network diversity.
- **Interdomain Communication**: Name-based routing and name-resolution services.

2.2 NetInf for Events with Large Crowds

The Event with Large Crowds (EwLC) scenario has been chosen as a suitable scenario for demonstrating the benefits of NetInf over previous networking architectures. This demo shows 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

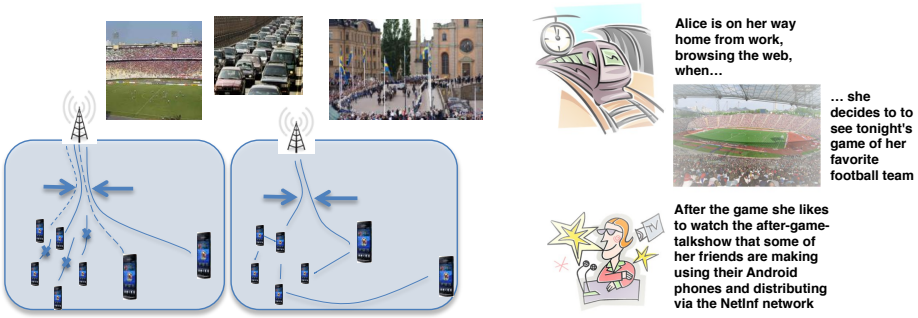
relatively short period of time (on the order of tens of minutes to hours), this creates a very high load on the cellular network.

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 wants to view a replay of a goal). Thus, there is 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.

In this demo, we show an integration of multiple partner prototypes into a working proof-of-concept EwLC system. In addition to the required NetInf infrastructure (routing, caching, and name resolution), a NetInf system for Android devices has been implemented, and three end-user applications are shown. These are collaborative web-browsing, photo sharing with a visual content directory, and video streaming over the NetInf protocol.

In addition, there is a visualisation server that makes it easier to see what is happening in the network. The demo configuration and a screenshot of the visualization tool can be found in Figure 2.10.

2.2.1 NetInf Demo Overview



(a) EwLC environment (b) Demo scenario

Figure 2.3: EwLC demo

Basic problem to be solved The common problem that we illustrate with the EwLC scenario (Figure 2.3a) is when the wireless infrastructure can not support the communication needs of a group of people. In certain situations, when the group wants to access the same set of content items sharing them locally using NetInf can be a very powerful technique. Figure 2.3b shows the demo scenario.

Commuter train The commuter train has a NetInf cache server that includes a Name Resolution System (NRS) (Figure 2.4). It both contains pre-cached material and such that is being cached from user traffic. Devices onboard the train can download via 3G (when available), get objects from the cache and exchange objects directly via, e.g., Bluetooth, in a p2p fashion.

Stadium scenario The stadium often hosts events that gather large crowds. As people come there for the same events, they have similar interests in the content they consume. They are likely to

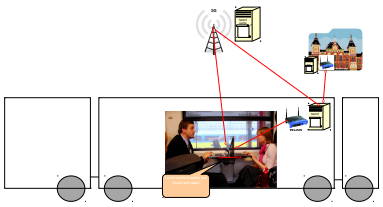


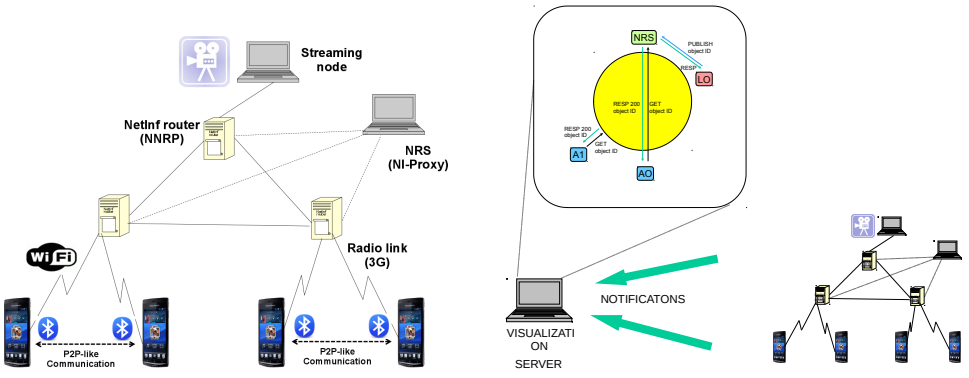
Figure 2.4: Commuter train

request information about the game, share locally produced media content from the event, and watch video streams from the event

The stadium has 3G coverage, and its infrastructure includes WiFi networks that are connected to a local NetInf infrastructure of caches and NRS, helping to reduce the otherwise high load on 3G and WiFi infrastructure. Network load can be further reduced by sharing content over local communication channels to people in adjacent parts of the stadium.

Figure 2.5: Stadion scenario

Visualization tool The visualization server stores and analyzes notifications related to NetInf node signalling, and displays the signals in real-time. The sequence of signals can also be stepped through in a non-realtime display mode. The visualization server provides a network perspective of the signalling between the NetInf nodes, as opposed to a traditional protocol analyzer, which only provides a link local view. The visualization server is useful both for debugging and demonstration purposes.



(a) Demo configuration (b) Visualization tool

Figure 2.6: EwLC demo visualization

2.2.2 Physical Node Demo

2.2.2.1 Overview

To showcase some of the potential for NetInf in an event with large crowds, a proof-of-concept system has been implemented with components from different partners. The demo setup is shown in Figure 2.7, where both mobile clients and infrasturcture nodes are visible.

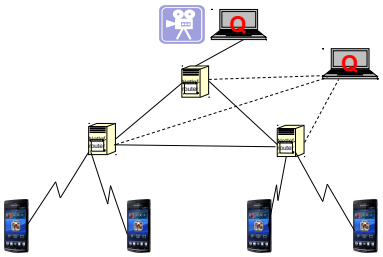


Figure 2.7: Physical Node Demo setup

In this setup, routing and on path caching are provided by NEC NetInf Router Platform (NNRP) and Ericsson Erlang NetInf routers. The Erlang routers are able to do request aggregation which is needed to avoid request storms towards the streaming node. This is especially important in flash crowd scenarios. NNRP's functionality is shown in Figure 2.8, where a local message processing is illustrated to show what kind of processing is involved between input and output CLs.

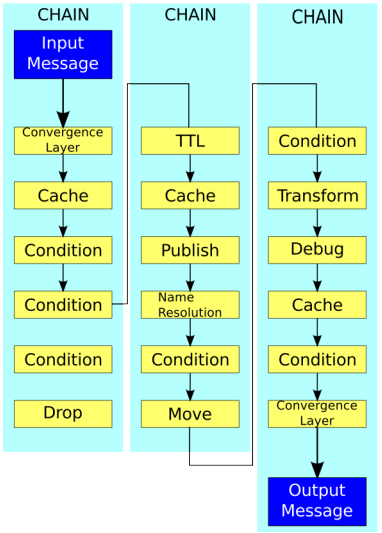


Figure 2.8: NNRP message processing

The Name Resolution Service (NRS) is provided by the NiProxy implemented by NSN. The NiProxy is also provides a storage service which allows clients to make a "Full PUT" which means that in addition to register the NDO in the NRS with its metadata (including locators) the NDO can also be explicitly cashed at the NiProxy. It also provides a search services by which NDOs matching certain metadata attributes can be found.

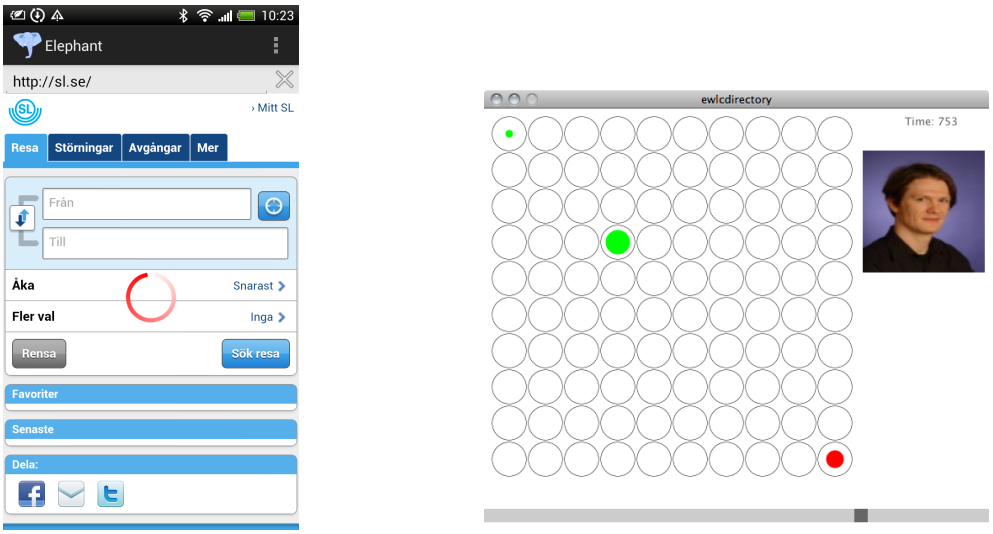
The NetInf Android mobile clients are implemented by Ericsson and can connect to the NetInf network via WiFi or 3G to retrieve or publish NDOs. They can also do this in a peer-to-peer fashion by directly communicating with each other via Bluetooth. The Android devices implements a generic Netinf service that provides an HTTP-based local API to applications which allows anyone to create a NetInf-enabled Android App. Finally there is a visualization tool, implemented by Ericsson, which

shows the signalling messages that are exchanged between the nodes in the demo.

Three example apps are shown in the demo web browsing, photo sharing and live streaming, we describe them next.

2.2.2.2 Collaborative web browsing

When a web page is requested, the NiProxy is searched for an NDO corresponding to that page. If found, the NDO is retrieved from appropriate locator (either NiProxy, NNRP cache, or over Bluetooth from mobile peer. If not found, the web page is retrieved using legacy HTTP. An NDO is then created and registered with the NiProxy for future requests User selects if retrieved content should be shared at central repository and/or locally. Status indicators show what source a web page is being retrieved from.



(a) Collaborative web browsing UI (b) Visual content directory UI

Figure 2.9: EwLC demo UIs

2.2.2.3 Local photo sharing and visual content directory

Problem: How does a user know what locally produced content is available? A user may for example be interested in photos of a goal occurring at a certain point in time. Each user is assigned a location in a stadium environment (seat number). Content that is shared by this user is tagged with metadata indicating the location of the user.

Users publish photos as NDOs at the NiProxy. Visual content directory allows browsing of available content over space and time. Use slider to show which locations published NDOs at different times. Tap the seat location to retrieve NDO.

2.2.2.4 Streaming Demo

Overview and Features NetInf Live Streaming demo for the EwLC scenario has some key benefits of NetInf over previous networking architectures. This demo will show some of the benefits that makes NetInf an excellent platform for ad-hoc video distribution as well as an alternative infrastructure for regular media broadcast. Some of the key features include:

- Any node can be the source of a live stream

- No advance or special configuration of the network is needed as NetInf natively supports multicast functionality through caching and request aggregation. Flash crowd problems are thus avoided.
- Each viewer can independently choose to watch stream live, or from the beginning. Pausing/-timeshifting the stream is also supported.
- Stream chunks can be retrieved from any node in a p2p fashion

Technical details Each stream is named by a stream identifier which is constructed by hashing the stream name (e.g., a human readable filename). The chunk names are constructed by appending the chunk number to stream id. The chunks are grouped into blocks that are signed. The block size is recorded in the metadata of the NDO identified by the stream id (which also contains information such as latest produced chunk). The meta data of each chunk NDO contains the signed block digest and the digests of the other chunks in the block. This allows for verification of each chunk independently immediately when received. For details, see [6].

For a user to connect to a stream:

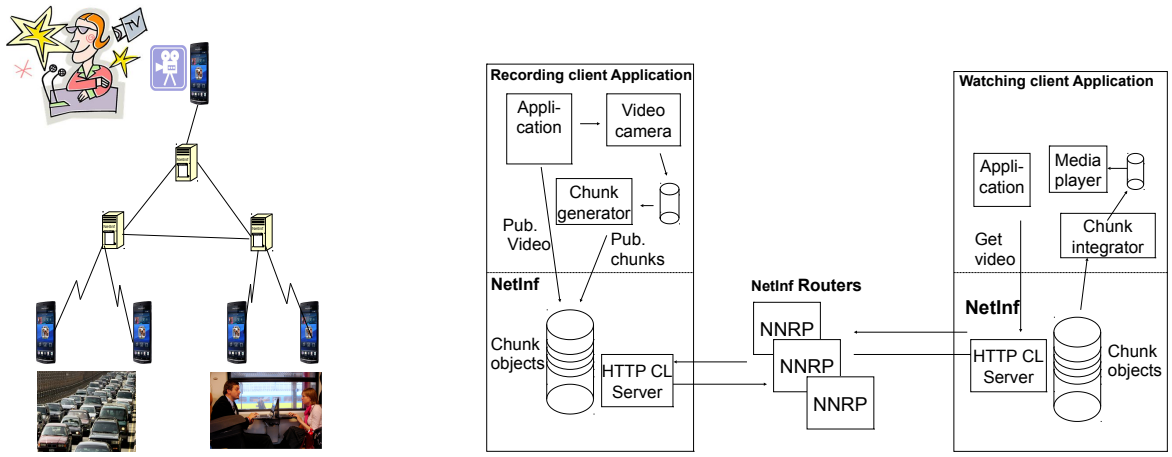
1. Hash the name of the stream to get the stream NDO ID
2. Request the stream NDO
3. Decide where to start playing the stream.
   - Live: chunk=current
   - Start: chunk=1
   - Starting from minute x: chunk=x*(chunk length/min)
4. Request subsequent chunks

Responding to a stream request:

1. When responding to a GET request for the stream NDO, that NDO MUST be marked as non-cacheable.
2. When responding to a GET request for the stream-chunk NDO, that NDO MUST NOT be marked as non-cacheable.

All nodes MUST understand the non-cacheable marking. There is a trade-off between the block size and delay. The larger the block size the longer the delay before the block can be transmitted. On the other hand the larger the block the less processing overhead (and delay) due to the signing process.

Streaming scenario As described in the overall demo scenario Alice likes to watch the after-game-talkshow that some of her friends are making from a bar nearby the stadium. Half a year ago it was only Alice and a few friends that were watching the show, but now, after a big game there can be up to 10 000 people watching the show while commuting home or from the couch in their living rooms. Her friends are only using their Android phones to live stream the show. Thanks to the NetInf enabled network the streaming scales transparently from 10 to 10 000 viewers without putting any significant strain on the network.



(a) Streaming scenario (b) NetInf live streaming

Figure 2.10: Streaming demo

NetInf live streaming in the demo The live streaming demo consists of two clients; i) recording client and ii) watching client as shown in Figure 2.10b.

Recording client

- Video is captured and stored in an MPEG2 file
- Extract chunks from the file and publish in the local cache under the NDO name “VideoName"+“ChunkNumber"
- Update “latest chunk" info in the metadata of the main NDO name “VideoName"

Watching client

- The application sends a GET request for the main NDO name “VideoName" to its “local NetInf node".
- The local NetInf node contacts the NRS to resolve the NDO name “VideoName" into a (set of) locator(s). It then forwards the GET request towards the Recording client.
- The recording client returns the main NDO name “VideoName" including the latest recorded chunk information.
- The watching client starts retrieving chunks by making GET requests for the NDO names “VideoName"+“ChunkNumber"
- Chunk integrator reassembles chunks to media file
- Media player start reading media file and play out content

Named-data-integrity for NetInf Streaming The key issue is that the name-data-integrity is lost with sequential chunk numbering. This can be solved by signing the chunks. Yet, thereby a new issue arises: Signing individual chunks is computationally heavy. Alternative solutions are:

- The chunks are grouped into blocks that are signed, see Figure 2.11. The block size is recorded in the metadata of the NDO identified by the stream id. The metadata of each chunk NDO contains the signed block digest and the digests of the other chunks in the block. This allows for

verification of each chunk independently immediately when received, e.g., D3 can be verified by having the signed digest D1-8 and the digests D1-D2, D4-D8, D1-4 and D3-4. For details, see [6].

- In the future in many scenarios signing individual chunks might be feasible (we like IETF PPSP WG stays open for both these options).
- For applications that have their own security mechanisms at higher layers signing might not be needed, e.g. like distribution of broadcast TV with dedicated set top boxes.

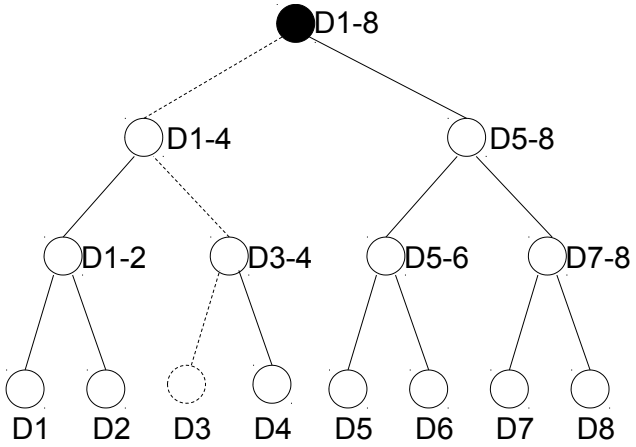


Figure 2.11: Block signing process

2.3 NetInf Multi-partner Testbed Configuration and Management

The NetInf testbed runs on machines dedicated by the partners to execute a set of NetInf prototype nodes. Targeting a large testbed running several hundreds of NetInf nodes, virtualizing nodes is the only feasible option. Thus WPB created an lxc image that allows to run different configurations of the NNRP along with a setup and configuration framework. Using this framework it is possible to deploy 50-150 interconnected nodes on a new machine and integrating them into the existing testbed. This should not take more than 5min. The framework allows easy updates to the installed NNRP binaries.

At a high level the testbed consists of two tiers. The inner tier connects the different partner sites over the Internet. Each sites is represented by one Gateway (GW) in this tier. The GWs primary task is to connect the different partner sites. In the outer tier the partners can deploy their preferred configuration and use the GW nodes to reach remote nodes and content. The suggested option is to connect the virtualized nodes running in a machine via a dedicated Access Point (AP) to the GW. That way it is easy to add both new NetInf testbed sites and more machines to a specific site. To ensure reachability in the inner tier the testbed relies on the Routing Hints extension to NNRP (which has been developed by SICS, see also Routing Hints poster for more info).

At the moment the testbed is configured with the EwLC emulation scenario in mind (see also EwLC emulation demo poster). For this all the nodes are not only connected to the AP, but also in adhoc groups with each other. To make the EwLC scenario more realistic traffic shaping is used to emulate the bandwidth and delay of 3G and Adhoc WLAN networks. The current configuration framework instantiates a configuration in which both the networks are used and the in-network caching capabilities of NetInf are leveraged. Targeting the EwLC emulation we make the following assumptions:

- Objects are published only on nodes and locators are forwarded to APs which store them
- APs and GW for caching and routing (Naming scheme reflects organizational context)
- APs fetch the object when matching a locator and serve object, instead of only returning the locator
- When a node receives a request it will first broadcast it in its Adhoc group (UDP, 1/2 sec timeout), and if unsuccessful forward it to the AP.
- The AP in turn checks his caches and locators and if that fails uses the routing hints to determine the next hop. This continues until the object is found.

To allow for centralized object publishing and inserting requests at dedicated nodes, the configuration framework also features a control network that can be extended across different machines and sites using L2 tunnels (in our case SSH).

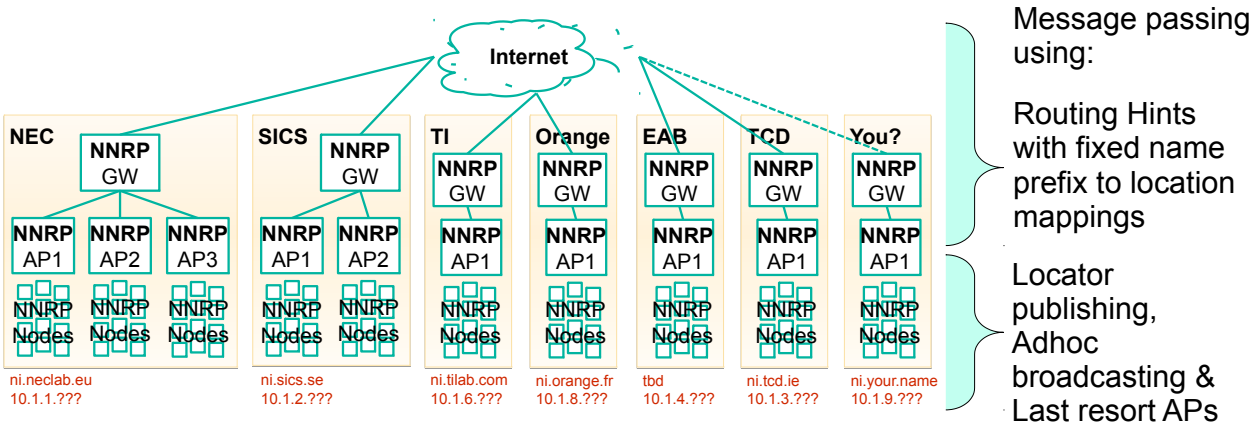


Figure 2.12: Testbed topology

Testbed Topology & Message passing mechanism Figure 2.12 shows the topology of the multi-partner NetInf testbed.

- Each site is running virtual network and virtual nodes (Linux Container (LXC))
- NEC NetInf Router Platform (NNRP)
- SICS Router Module for NNRP
- Orange Transport Module for NNRP

Testbed Adaptation to Different Scenarios Support for different topologies, network constraints, and node behavior. Event With Large Crowd Configuration used here:

- Objects are published on nodes only; Locators are forwarded to APs
- APs and GW for caching and routing (Naming scheme reflects organizational context)
- APs fetch the object when matching a locator and serve object, instead of only returning the locator

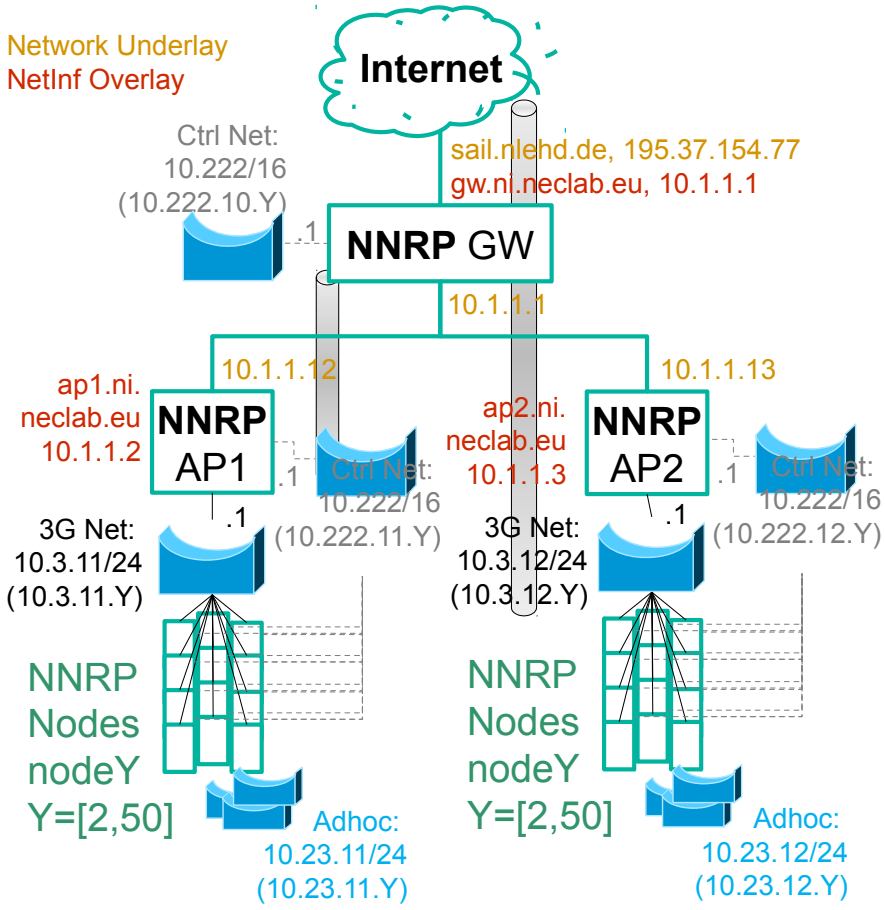


Figure 2.13: Single Location Setup

Single Location Setup Figure 2.13 shows the testbed configuration at a single partner site. In this setup, only GW needs Internet access, GW can reside on same machine as AP+Nodes and one large L2 control network connected via tunnels (Secure SHell (SSH)).

NetInf GET Message Processing Stop if any of the steps yields the object:

- Node checks local caches
- Node broadcast request to adhoc network
- Forward request to AP, await response
  - AP checks locator database
    - On hit GET the object from locator, put in cache and serve back
  - AP checks local caches
  - AP uses routing hint to locate next hop
    - Can be another local AP or the GW

Routing example GET for ni://ap3.ni.neclab.eu/sha-256;4âĂę.

- Prefix resolves to 10.1.1.4
- TI AP → nexthop is TI GW

- TI GW → nexthop is NEC GW
- NEC GW → nexthop is NEC AP3
- NEC AP3 knows where to get the object local

Management Tools Scripts to start and stop virtual machines from single LXC image → many nodes possible. Scripts to configure LXC, NNRPs and network on each machine. Framework to publish and request objects and query cache status

2.4 NetInf EwLC Emulation Demo Caching & Adhoc networking Benefits

Idea

- Emulating specific network setups to evaluate NetInf protocol performance in different load scenarios
- Motivation: running real code in controlled environment for more meaningful and accurate evaluation
- Event with Large Crowd: Many mobile NetInf nodes connected to wireless infrastructure network and enabled to communicate locally (see Figure 2.14)
- Configuring different mobility patterns, publish/requests patterns, popularity distribution etc.

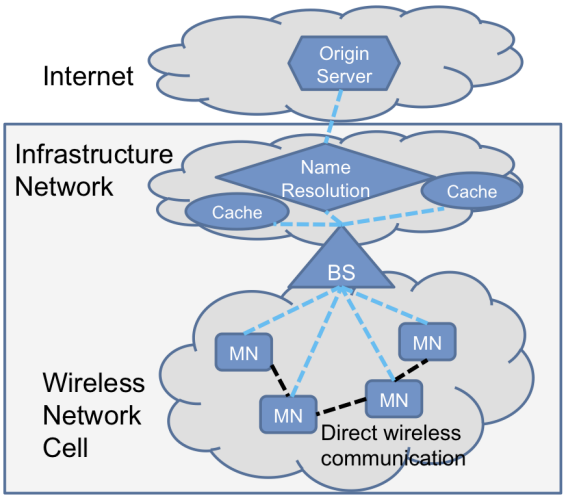


Figure 2.14: EwLC environment

Approach Use the NetInf Testbed as execution platform

- NNRPs in LXC virtual machines, 50+ nodes per machine
- Multiple physical machines to scale up network
- Testbed APs correspond to Base Station (BS), nodes to Mobile Node (MN), GW for connectivity, Remote location nodes serve as origin servers

Emulate two networks per node + Control to issue request

- 3G: connecting all nodes of a BS, traffic shaped to 7.2Mbps/150ms
- Adhoc: connecting 8-12 nodes directly, traffic shaped to 54Mbps/2ms

Script node behavior, request generation, communication link and storage constraints to ensure reproducible experiments. Collect performance measurements for real-time and offline evaluation (Not in demo) Mobility through changing Adhoc group memberships.

Lessons learned:

- NS3 is too slow to emulate Adhoc WLAN connectivity and/or mobility for 20+ nodes
- NNRP can process (relay) a request in less than 20ms
- LXC very light-weight and scalable

Scenario For the evaluation we assume the scenario of multiple soccer games being played at the same time in multiple arenas. Each testbed AP corresponds to one arena, and the ICN nodes attached to the AP correspond to users in that arena (app. 80 nodes per AP).

The users are not only interested in their local game, and especially in periods of "boring" gameplay they might be interested in seeing what is going on in the other arenas.

Our scenario procedure is as follows:

1. A goal is scored in one of the arenas → 5 people in that arena publish an object
2. Five very curious people in the other arenas access one of these objects each. Then pause for 10 seconds.
3. The news spreads and some rounds of increasing numbers of users access decreasing numbers of objects (some object are more popular than others. In our scenario we use
   a) Access top 4 objects, with 4 requests per object per arena
   b) Access top 3 objects, with 10 requests per object per arena
   c) Access top 2 objects, with 50 requests per object per arena

Results To demonstrate the benefits of caching an adhoc communication we show CDFs (cumulative distribution functions) of the response times to the request pattern described in the scenarios subsection. In addition we show the network load in terms of number of requests.

In the first round of requests after a new goal (c.f., step 2 of scenario) all the requests are served from the originator himself as only he has the content. This is shown in Figure 2.15. We expect response times around 2.5 seconds for remote delivery (see blue area in plot). These 2.5 seconds result for 3 contributing delays:

- 500ms waiting for responses from the broadcast to the adhoc domain.
- One second for the TCP handshake and slow start over a 150ms delay 3G network to the AP.
- Another second for the TCP handshake and slow start over a 150ms delay 3G network from the remote AP to the originator node.

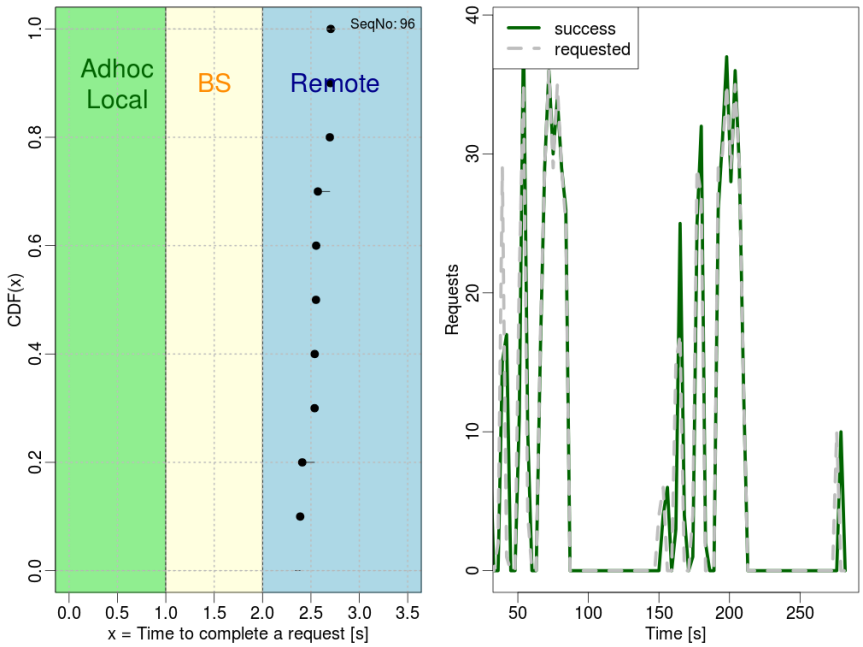


Figure 2.15: Emulation results (left: CDF of response times, right: network load) from first round of requests (see scenario description step 2). All requests are served from content originator in remote stadium (blue area).

In the following round of requests (c.f., step 3a of scenario) all the AP have all the objects cached. Thus, in Figure 2.16 the majority of requests is served from the caches of the local APs (yellow area in plot). According to the same delays as in the first round, delivery from the AP is expected to require 1.5 seconds (saving the TCP connect to the originator node). However some requesters are lucky and already find the object in their local adhoc group (green area in plot).

The additional requests round fills the caches of the adhoc group nodes even more, so that in the final round of requests (c.f., step 3c of scenario) is mainly served from the adhoc group nodes which respond immediately. This is shown in Figure 2.17.

Overall we observe a significant offload potential through ICN in-network storage and NetInf local communication. Mobility can be a further feature by disseminating locally generated or cached data objects.

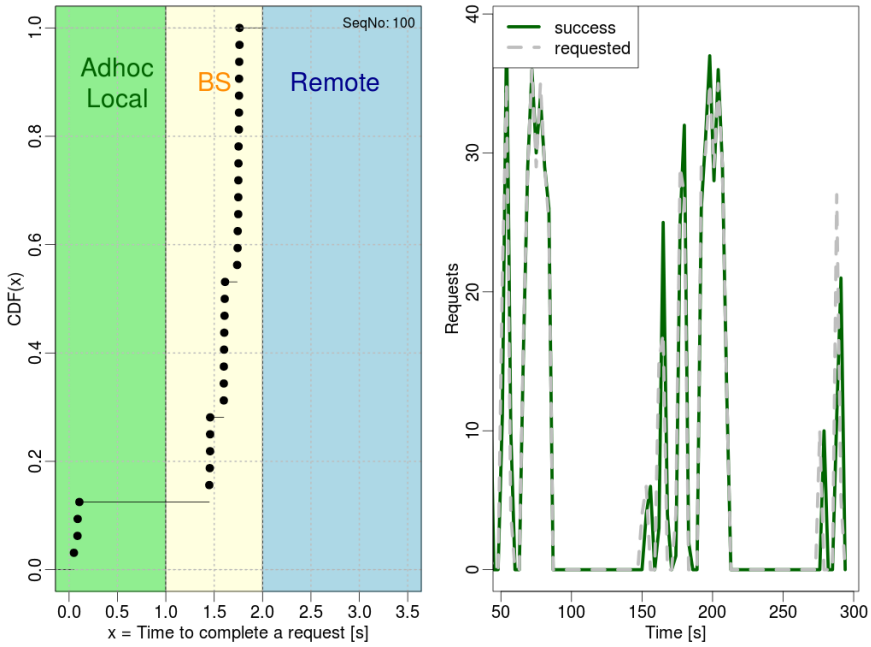


Figure 2.16: Emulation results (left: CDF of response times, right: network load) from second round of requests (see scenario description step 3.a). Most requests are served from the AP cache (yellow area) and few are served locally from other nodes in the adhoc group (green area). That downloaded the object before.

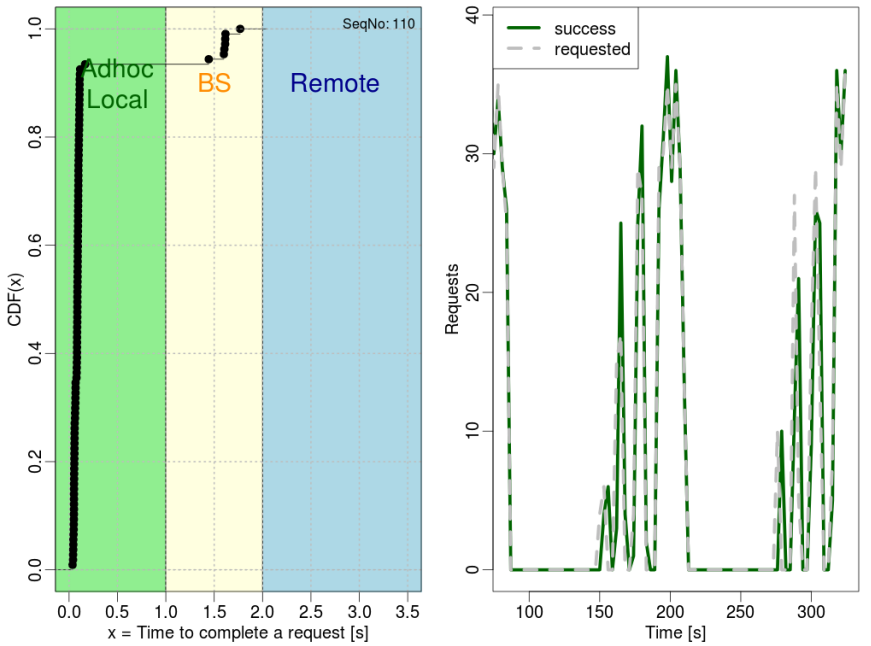


Figure 2.17: Emulation results (left: CDF of response times, right: network load) from last round of requests (see scenario description step 3.c). Most requests are served locally from other nodes in the adhoc group (green area)

2.5 NetInf Global Routing Using Hints

The global routing mechanism for the NetInf protocol makes use of two levels of aggregation in order to achieve a high level of scalability. The mechanism is an adaptation of the Global Information Network (GIN) Architecture [7] and Narayanan and Oran's ideas [8] to the NetInf protocol. The mechanism is mainly concerned with the global aspects of routing requests for NDOs towards the corresponding publisher, i.e., routing in the NetInf default-free zone, comparable to the current Internet's BGP-routed infrastructure. Just like in the current Internet, edge domains can utilise different routing schemes that are adapted to particular needs of the respective domain.

NDO aggregation Before going into the detailed design we briefly review the prerequisites. An ICN needs in principle to keep track of all NDOs available in the global Internet, just like the current Internet in principle needs to keep track of all hosts. Estimates of the number of objects range from 7 billion to one trillion (10^{12}). To have margin for growth, we need to aim higher, at least to 10^{15}. The key to be able to globally route requests for this large number of NDOs is *aggregation*. We thus introduce the notion of NDO aggregation, meaning that a set of NDOs are grouped together. For routing and forwarding purposes, the NDOs in an aggregate are then treated the same. Such NDO aggregates, with the same origin, occur naturally in reality, for instance, chunks of a video, photos in a collection, individual objects on a web page and/or site, and so on. NDO aggregation increases performance in that a name resolution cost need only be taken for the first NDO of the aggregate. It likewise increases scalability in that routing information is only needed for the aggregate as a whole. We use the authority part of the ni: URI [5] to name NDO aggregates.

Components 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 (see example in Figure 2.18).
- *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: URI and/or routing hints into the address of the next hop to forward the request to. An example forwarding table is illustrated in Figure 2.19.

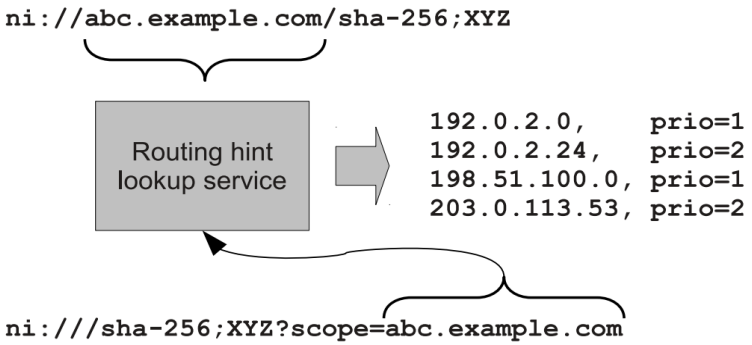


Figure 2.18: Routing hint example

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

Figure 2.19: Forwarding table example

NDO aggregation and routing hints

- Adaptation of the GIN Architecture and Narayanan and Oran's ideas to the NetInf protocol.
- Aggregation of routing information key to scalability.
- Named Data Objects (NDOs) are grouped into aggregates.
- NDO aggregates are mapped to routing hints thought a lookup service (can be DNS).
- Multiple hints with priorities.
- Global routing only needed for lowest priority hints.

Forwarding process

- Check the object table (cached and permanently served NDOs)
- Check ni: name forwarding table; if match, forward to that next-hop
- If needed, perform lookup of routing hints
- Lookup all hints in routing hint forwarding table; if match, forward to next-hop of hint with highest priority

Implementation

- NEC NetInf Router Plaftorm (NNRP)
- SICS Router Module for NNRP

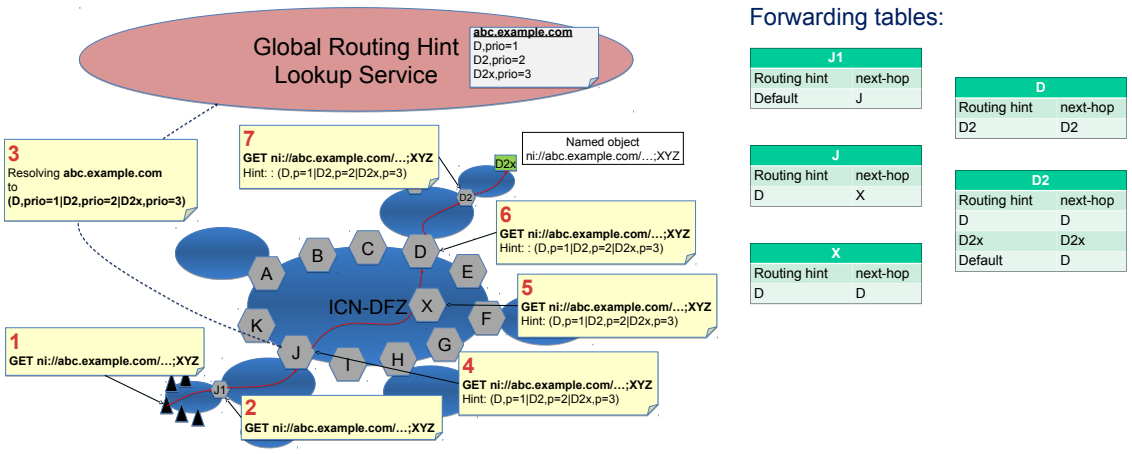


Figure 2.20: Routing hints usage example

Figure 2.20 shows an example on how routing hints are used in NetInf global routing. The client at the bottom left requests an NDO. The request is forwarded using default routing to the edge of the default free zone (DFZ), where router J looks up routing hints with the result that three hints with different priorities are added to the request. The request is then forwarded by looking up the hints in the respective NetInf router's forwarding tables until a node with the NDO is reached.

2.6 NetInf Congestion Control Protocol

Congestion control in ICN paradigm is still a challenge because there is no classical Internet Protocol (IP) flow in such networks. We propose here a NetInf Congestion Control Protocol, which is based on recent work [9, 10]. Its main aim is to keep the network stable by limiting the number of GET requests issued by end-nodes, whilst still using most of the capacity. It is implemented as an NNRP module and integrated in the testbed.

The protocol is based on the assumption that Application Data Unit (ADU) are published as a set of small NDOs, which we call *chunks*. In particular, the chunks can be requested by any node, and their integrity can be validated. The chunking operation is assumed to be performed by the publisher, so that it can be adapted to the need of the applications (e.g. specific framing for video application). We however expect that some open software will provide default chunking for publishers who do not want to care about it. The benefit for doing chunking and congestion control at the chunk level is that it allows to deal with small NDOs, and hence reduce the burden of reassembling and fragmenting the NDOs at each CL link.

The module transforms an application GET request in a set of requests for the corresponding NDOs, controlling their pace by adapting to loss events. These requests are then carried along the network as any NetInf message. The module is also responsible for re-requesting unanswered GET requests, hence performing error recovery function. It finally reassembles the set of NDOs into the requested ADU and returns it to the application. Note that the module is agnostic about the CLs which are used to fetch the NDOs. They may in particular perform also congestion control and NDO reassembly. The module just assume that the CL is able to return the requested NDO.

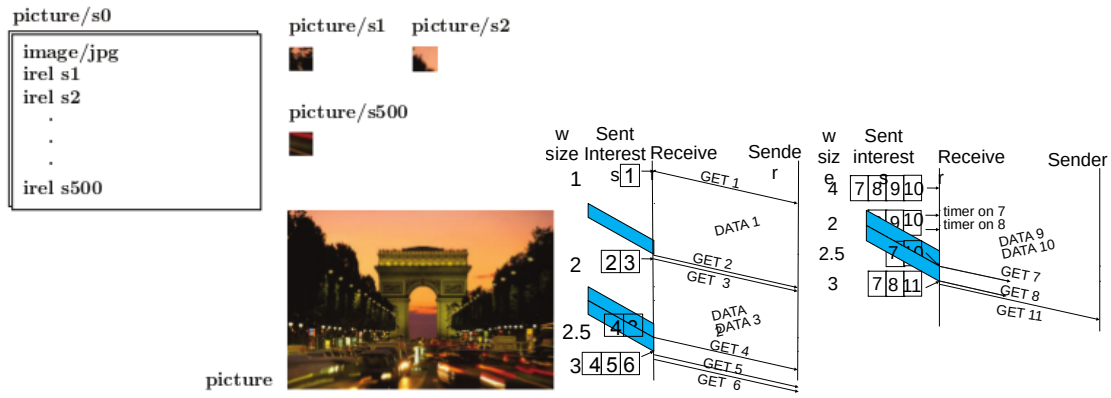
Finally, as this protocol runs at the client side, and does not assume that all NDOs are fetched from the same source, it can be used in a multipath context without need of adaptation, as we show in a proof-of-concept experiment.

Key design choices:

- Receiver-based
- Object published as a set of small chunks
- Chunks are regular NDOs
- AIMD window controls the rate of GET messages
- Re-requests unanswered GET

POC for multipath:

- Developed as a NNRP module
- Round-robin forwarding based on a set of next hops



(a) Chunking (b) Protocol

Figure 2.21: Chunking and Congestion Control Protocol

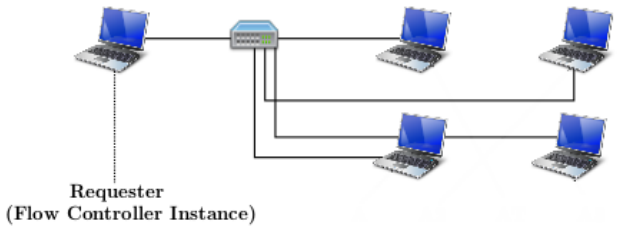


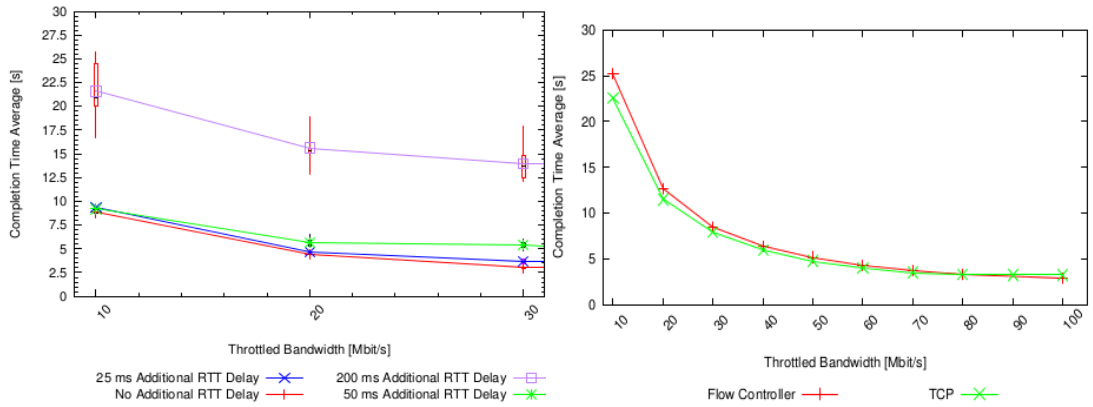
Figure 2.22: NetInf Congestion Control Protocol Testbed

2.7 Caching in a Network of Information (with visualization)

2.7.1 Principle

Network caching of data retrieved from a server has been investigated both from a research and industry perspective. From work in the early 90's on web server caching, to more recent work (Video on Demand (VoD) file distribution and 3G access networks) caching has become a viable solution within the network to save resources. In this demonstration, we show how to cache objects in an Information Centric Network. The significant difference between an ICN model and today's networks is that each web object has a unique identifier that is used to create, locate, distribute and verify the object. The overwhelming advantage is that the traditional client-server model can be split allowing objects to be moved, replicated, cached without an end-to- end connection. Additionally metadata can be used to store attributes about the object. We leverage this architecture to design a new type of cache management (or eviction) policy. We use the metadata field to store the hitrate of each object and compare this field to the other cached items when deciding which object to replace as caches fill. We compare our approach, called Forward Meta Data (FMD), to Least Recently Used (LRU), Least Frequently Used (LFU) and random replacement policies.

We consider a hierarchy of caches in a tree structure with a source server at the top and clients requesting NDOs at the leaves. The basic concept is to decouple the client-server approach by allowing 'self-certifying' objects to be be transported rather than the TCP bitstream for reliable traffic or the UDP datagrams in unreliable transfers. Obviously, it is desirable to store highly popular content in caches close to the clients and less popular NDOs higher up in the cache hierarchy in order to reduce overall latency and load on the network. We assume that the caches have limited storage space and an eviction policy is applied.



(a) 3 sources measurements (b) Single source measurements

Figure 2.23: Measurements on the Congestion Control Protocol

We simulate a hierarchical infrastructure to determine which NDO's to evict from a full cache when a new NDO is visible to the cache. Two of the simplest and also most widely used policies are LFU and LRU. The LFU selects the Information Object (IO) with the lowest hit rate while LRU evicts the IO that has not been requested for the longest time.

A problem with LFU and LRU is that these policies tend to evict potentially popular content before they have an opportunity to grow popular when storage space is small compared to the amount of IOs visible to the cache. To mitigate this effect caches higher up in the hierarchy are provided with larger storage space. This, however, works to a limited extent since the cache memory becomes to large and latency for searching the cache becomes prohibitive. Ideally, the most popular content should be stored in the cache closest to the client while the next popular content is stored at the next level in the cache hierarchy etc. Furthermore, one could argue that once popular content has been distributed out to the caches close to the edge of the network this content could be expunged from caches higher up the hierarchy to provide storage space for new content. To achieve this we propose to attach meta-information about the popularity of the IO when it is sent from a source towards the requesting client. The meta-information is updated at each cache and the updated meta-information is used by the re-placement function. We call our proposal FMD. At each cache on the path to the client the meta data is examined and if the popularity of the IO is higher than the least popular content in the cache the IO is replaced with the newly arrived IO.

We show this dynamic behavior through a poster explanation and a demonstration based on a Java simulator.

Basic problem can be defined as follows:

- Investigate caching policies in a NetInf.
- In a system with many edge nodes & few storage servers.
  - Form a tree-like structure with requesters at the leaves as shown in Figure 2.24.
  - All items are stored in the server.
- Distribute copies of the items within the tree.
  - Minimize the access time of the data items by reducing the load on the (few) servers and on the bottleneck link.
- Classical problem is which items to evict as the caches fill.

- Other works investigate the policies and system requirements.

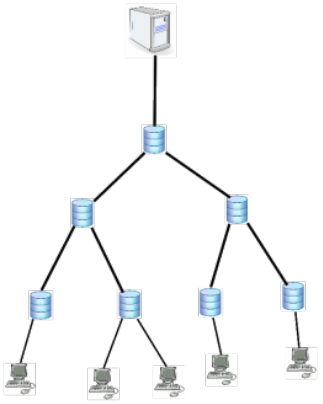


Figure 2.24: Tree-like structure

2.7.2 Philosophy

Network of Information caching can be summarized as follows:

- The NetInf idea is to decouple self-contained objects from servers.
- Popular items should be cached as above.
- Important mechanism is that popular items should be replicated.
- Since objects are self-contained the access to each object needs to be kept.
- The metadata is one place to hold the number of accesses.
- We call this FMD.

In order to fulfill the demands of the bullet list above, we need to design an algorithm that will work on a large scale and is more effective than comparing the popularity of each item in each cache as it is migrated or moved. Even so, we are not guaranteed that the items with the highest popularities that fit in a cache are closest, or even can be served faster than accessing the server, this depends on the object size, search time in the cache, and congestion on the links. However, what we can achieve, is fewer comparisons for each item at each cache, especially when determining whether to store the item as well. This is conceptually simple, each items **knows** its popularity by leveraging the meta-data concept developed within SAIL.

The FMD algorithm can be stated as follows:

- Requesters at leaves request an object.
- A search is conducted up the tree.
- If the item is found its access count is updated by one.
- On delivery to the client its access count is compared to the lowest hit rate in the cache.

- If its hit rate is higher than the lowest hit rate, it replaces the cached item.
- Repeated for each cache down the tree.

Simulation/visualization environment A custom C++/Processing environment has been implemented supporting five different cache management algorithms: LRU, LFU, Random (RND), Least Requested Rate (LRR) and our FMD (forward meta data) approach. We have a C++ standalone simulator for larger scale simulations and a visualization tool, built on the processing language to show *how* items are stored and evicted over time, this is shown in Figure 2.26. Items to be stored follow two different popularity distributions:

- A theoretical Zipf's law $z(k;\alpha) = 1/k^{\alpha}$
- Trace driven from Orange's 3G mobile data collection

Results Figure 2.25 shows the average number of hops for cache sizes (relative to document size) and the effect of the distribution of the popularity (α). Respectively Figure 2.26 shows average rank of the documents in memory at each level (100,000 docs, 100 cache size, a very small cache compared to the total number of documents. We will see other permutations for video files below from Orange Labs.), lower numbers indicate how the most popular items are placed "more importantly" in the caches.

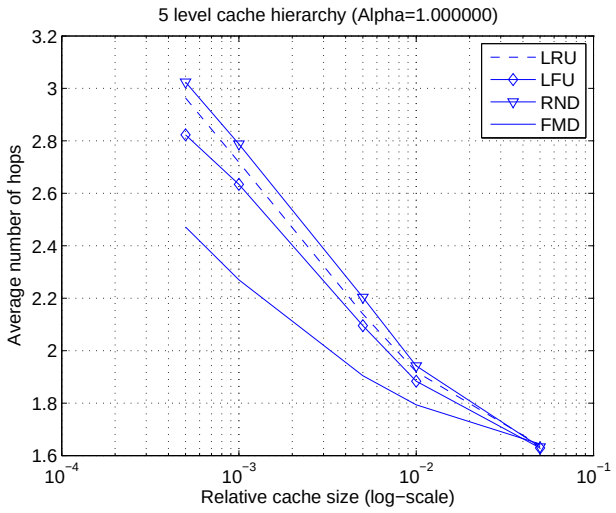


Figure 2.25: Popularity distribution

	LFU	LRU	RND	FMD
Level1	43428	98729	75171	183
Level2	62765	98357	95019	164
Level3	75756	84124	83547	145
Level4	84159	80345	80459	80

Figure 2.26: Document ranking

2.7.3 Visualisation

A visualization of the hierarchical caching process is shown in Figure 2.27. The visualization takes

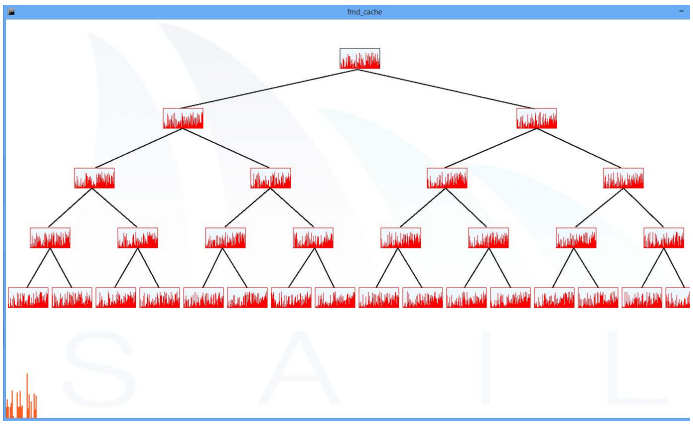


Figure 2.27: Cache hierarchy and occupancy (top), Orange networks trace file (bottom)

a request from either the Zipf-law distributed data or a request from the Orange trace file and allocates it one by one to one of the bottom caches (this could be a 3G base station). We could have looked in the trace file and could have estimated which requests were going to each, but did not, we simply allocated a request to each base station in turn. In a Internet scenario each request could be a DSLAM. If the item is in the cache it is served back to the requester. If not, the next cache up the tree (an operator cache for example) is searched, again if found it is served back down the tree. The process continues up the tree until the item is found or retrieved from the original server.

The lines show how the caches fill from an "empty" system, but over time only the popular items will be shown (Facebook, YouTube and Orange specific items in this case). Moving the mouse over will show which items are in each cache. Also we can see whether it is beneficial to store one large item or many small ones of the same size. This is more difficult to see in a pure simulation, we see only the hitrates per item per cache, not necessarily tradeoffs.

A realistic (video) example In June 2011 Cisco reported *Internet video is now 40 percent of consumer Internet traffic, and will reach 62 percent by the end of 2015, not including the amount of video exchanged through Peer-to-Peer (P2P) file sharing [11]. The sum of all forms of video (TV, VoD, Internet, and P2P) will continue to be approximately 90 percent of global consumer traffic by 2015.*

Given a sample network with 100 pools of Digital Subscriber Line Access Multiplexer (DSLAM) equipment that in turn are connected by 10 Broadband Remote Access Servers (BRAS) that connects to the root router. For simplicity all routers are equipped with the same amount of cache memory. Orange Labs reports video accesses summarized in table 2.7.3. One of the traces in the report describes accesses to a mixture of movies and trailers during an 8 day period. The average content size in this trace is 703 Mbytes which with a 360 Gbyte cache corresponds to a cache capacity of 512 items.

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

With these parameters as input to the model proposed in D.B.2 and published in the ICN Sigcomm workshop "On the Effects of Caching in Access Aggregation Networks" [12] we calculate a hit rate

of 45% at the routers closest to the terminals. At the next level, the hit rate will be 26% and at the top 22%. From the graphs of the Orange report a simulated LRU cache of the same size at the first level shows a better than 60% hit rate on real data. These percentages are conservative as our model assumes accesses to be independent and hence more spread out than accesses which are correlated in both space and time. The cache model does not take correlations into account and real hit rates will be higher. The results (hit rates) of the calculations are summarized for five different sizes of caches below.

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

2.7.3.1 Further considerations

An important question to consider is if flash memory will provide enough capacity to satisfy all hits. For reference we have picked a standard PCI-Express card with inexpensive flash memory that costs 700 EUR. It provides for 540 Mbytes/s of sustained read capacity. From the trace we can deduce an average VoD rate of 30 Mbyte/s (240 Mbit/s) of which 13.5 Mbyte/s should be read from our cache. In our experience, peak VoD load can be as much as 5 times the average load. In this case there is sufficient capacity to satisfy the demand. At the next level, 44 Mbyte/s will have to be read from the cache (remember to multiply by 10 because we are aggregating 10 subtrees). There is still plenty of capacity. Similarly, at the root level we must read data from the cache at an average rate of 267 Mbyte/s. This is possible, but if we expect peak load to be as much as 5 times the average load we need to use flash memory with higher read performance. Such memory exists and considering that this must be a rather high end router anyways this additional cost can likely be motivated. Consequently, introducing the comparatively small caches mentioned above should already today save some 27% of all consumer Internet traffic and extend to 42% savings by the end of 2015.

Other Observations Requests will be Zipf distributed using $\alpha = 0.7$. The table shows the hit rate of individual levels for a very deep tree network of 10 levels and cache sizes corresponding to 0.1% of the catalogue[1]. We note that the first level (leaf) cache is of highest importance. Caches at higher levels can still contribute to the aggregate cache effect but only to a much lesser extent as it is not additive.

Thus, when designing future in-network caching architectures one should consider using either larger caches at higher levels or to make groups of caches collaborate to create a larger virtual cache (paying the internal communication costs but increasing content availability). This holds for the whole parameter range but the relative performance of level 1 decreases with catalogue size. This is intuitive since for very large catalogue sizes each level in the cache will store very popular items and will have relevance on performance.

2.8 NetInf: Using Delay- and Disruption-Tolerant Networking (DTN) with Nilib

NetInf DTN Convergence Layer and DTN↔HTTP Gateway The NetInf architecture is designed to allow NDOs to be transported across paths that span multiple network domains. This work demonstrates NetInf operating in a DTN domain and connecting to the Internet.

Transport of NetInf requests and responses in the DTN domain uses a CL that carries NetInf messages in bundles using the bundle protocol [13]. The DTN CL integrates with the HTTP (and UDP) CLs specified in draft-kutscher-icnrg-netinf-protocol [4].

Code for the DTN CL is available in Nilib with C and Python interfaces. The Nilib code uses the DTN2 Open Source Software reference implementation to provide the bundle protocol interface. NetInf messages use bundle protocol Bundle Protocol Query (BPQ) and Metadata blocks to carry the affiliated data for NDOs.

A typical interdomain deployment scenario using NetInf Hypertext Transfer Protocol (HTTP) and DTN CLs is shown in Figure 2.28.

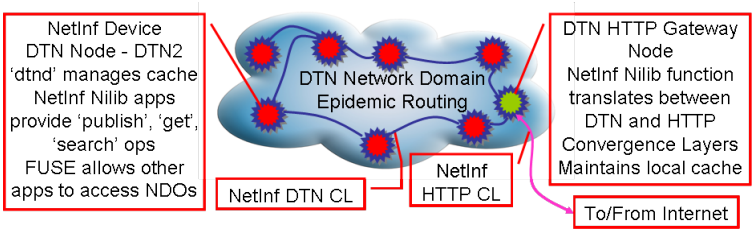


Figure 2.28: DTN scenario

NetInf ICN Device To demonstrate the DTN CL, TCD have developed a NetInf Device. This can instantiated in a tablet or netbook-style computer communicating with the aim of using ICN over DTN as its sole communication mechanism. The overall archicture of the NetInf device is shown in Figure 2.29. The primary components which are available as Open Source Software are:

- DTN2
- NetInf Nilib
- FUSE userspace file system

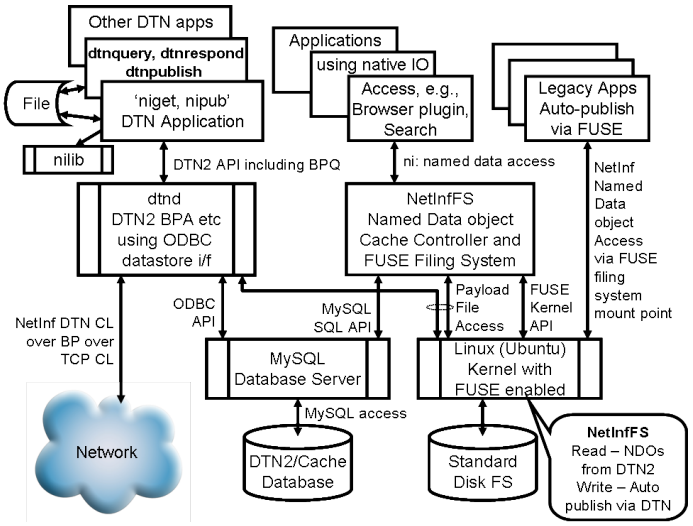


Figure 2.29: NetInf ICN device

NiLib Python The NetInf Device relies on the Python implementation of the NetInf protocol in Nilib. The Python implementation provides

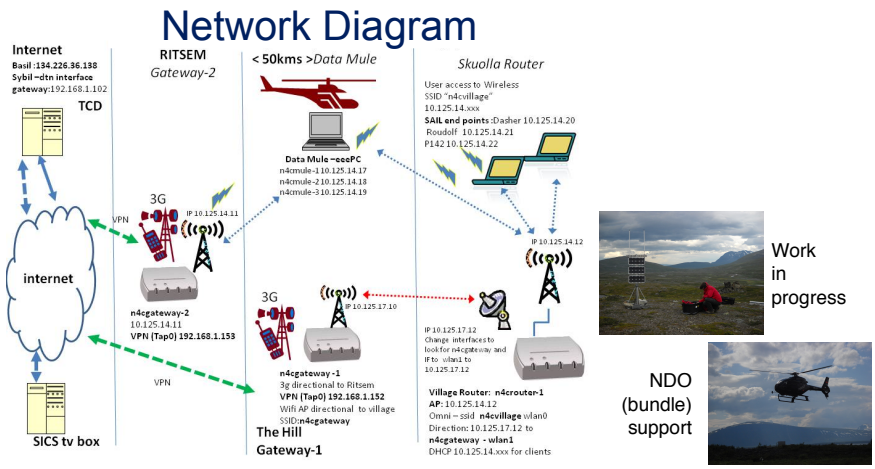
- NetInf HTTP and DTN CLs with GET forwarding
- PUBLISH, SEARCH operations intra- and inter-domain
- Using the filesystem for NDO content storage
- Using either filesystem or REDIS “NOSQL" DB for NDO meta-data
- Command line clients to initiate NetInf operations
- Apache module plug-in (via WSGI) or standalone Python for HTTP server
- DTN↔HTTP Gateway
- Running on tcd.netinf.eu SAIL testbed

DTN in use: SAIL Summer 2011 Trial - Padjelanta in Arctic Sweden - TCD and SICS The goals of this trial were as follows:

- to deploy hardware and software using the DTN bundle protocol for communication that was originally developed for the N4C project and improved for this trial,
- to provide demonstrations of applications including Twitter, Facebook, and email transported over DTN for reindeer herders operating in the area
- to make initial experiments with some new additions to DTN bundle protocol that support ICN over DTN using the Bundle Ptotocol Query extension block described in draft-irtf-dtnrg-bpq [14]

Figure 2.30 shows the network assembled for the trial which used a combination of long range but low bandwidth wireless relays and carriage of DTN bundles in storage using the helicopters that provide service to the semi-nomadic reindeer herders of Padjelanta during the summer months. The figure also includes images of work in progress during the trial including the solar powered ’village router’ that acted as a ’post office’ for the data being sent to and from the remote part of Padjelanta that is over 50km from roads, mains power supply and permanent communications infrastructure.

The NetInf DTN Convergence Layer The NetInf Protocol [4] adopts a CL architecture that allows NDOs to be transported between pairs of nodes using a CL protocol that is chosen for the nodes’ network domain. The NetInf protocol is well-suited for moving NDOs across disparate domains using gateways at the domain boundaries where they choose a CL protocol for the next transport hop: an incoming request using (say) the HTTP convergence layer in the well-connected Internet could be forwarded into a domain where DTN is appropriate. The work represented by this demonstration shows how a NetInf CL can be implemented on top of the DTN Bundle Protocol (BP) using the Bundle Protocol Query Extension Block (BPQ block) developed during the SAIL project to carry the NetInf specific information [14] in conjunction with Metadata Extension blocks. The DTN architecture supported by the BP [15] is described as a ’store, carry and forward’ so that all DTN nodes already support a cache for bundles that are in process of forwarding by the BP. Unlike conventional routers, these copies are not necessarily transient but may be held for extended periods of time to allow for the delays and disruptions foreseen in a DTN environment. During the SAIL project we have added functionality to the DTN2 Bundle Protocol Agent (BPA) daemon



(a) Trial scenario (b) Trial work

Figure 2.30: Network diagram of the “summer trial"

('dtnd') that uses the information in the BPQ block of bundles carrying NetInf requests to either 'publish' an NDO by retaining the content in its cache or create a response to a 'get' request from this cache when the NDO name matches with the NetInf request parameter.

HTTP↔DTN CL Gateway In addition to the HTTP CL components in the NetInf Nilib OSS, the repository now includes extensions to the standalone NetInf server that accept NetInf protocol messages carried over the DTN CL together with extended applications that can select the CL to use according to the format of locator hints provided.

NetInf DTN Device and FUSE NetInf Cache Access Component Computers used in the demonstration as DTN nodes are setup to manage external data access as far as possible using only NetInf access over the DTN BP. NDOs cached by the DTN BPA (dtnd) on these nodes can be read using their ni Uniform Resource Identifiers (URIs) as file names through a FUSE filesystem. Local applications can write files into this filing system which will be automatically published as they are completed ('closed') and will then be accessible either via the correct ni URI name or the local name used when they were created.

NetInf NiLib Python Performance Test Harness Nilib[2] is a set of open-source (Apache licensed) implementations of the NetInf Protocol developed as part of the SAIL project with various language bindings including C, PHP, Python, Ruby and Java. We use the Python client and Apache server implementation for this, and are developing a set of tools that will enable comparisons between NetInf protocol implementations and network designs, but can also be used to compare different ICN approaches. The basic approach is to have a standard corpus[3] and to develop tooling and measurements to enable the above comparisons to be made and to be independently replicated. We plan to continue this work within the IRTF ICN research group. We will demonstrate and discuss this early-stage work.

[2]http://sourceforge.net/projects/netinf/
[3]http://www.soschildrensvillages.org.uk/about-our-charity/archive/2008/10/2008-wikipedia-for-schools

2.9 GIN: A Global Information Network for NetInf

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). GIN aims to interconnect NDOs over heterogeneous L3/L2 underlayers, in the global network, by means of an integrated name-based resolution and routing mechanism. Data are routed by names into the GIN network on a double path: the resolution path is used to route 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 the Multilevel DHT (MDHT) architecture [5]. 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 Table (NHT). Figure 2.31 provides a view of the integrated name resolution and forwarding process in a GIN node.

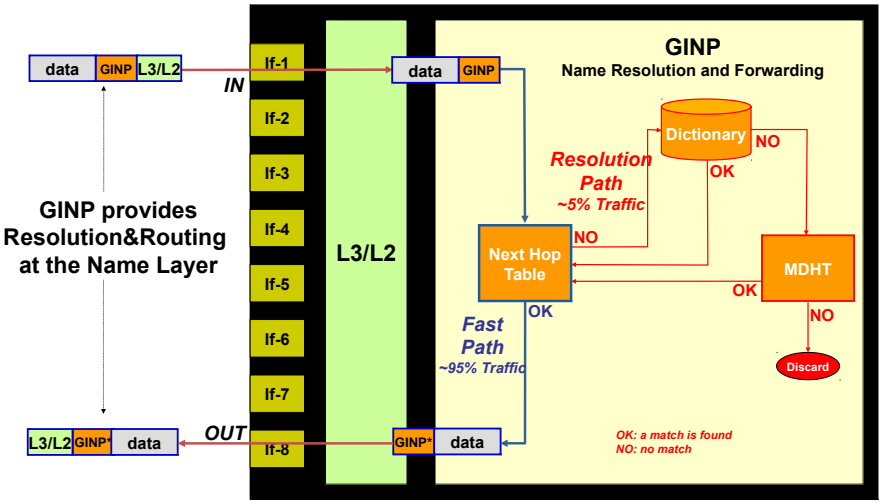


Figure 2.31: GIN node

Telecom Italia has developed a proof-of-concept prototype of a GIN infrastructure node. The current GIN demo prototype provides the following features and services:

- name based routing and forwarding over heterogeneous networks (IPv4, IPv6, ETH);
- integrated name resolution by means of a distributed MDHT dictionary;
- in-network registration and storage of named objects;
- multicast ping of named objects (called gping);
- end-to-end receiver-based multipath retrieval of named objects;
- support for search of data objects by keywords and names by means of a simple search engine.

The GIN node, developed on FreeBSD, consists of two subsystems (Figure 2.32): the GIN Switch and the GIN Dictionary.

The GIN Switch is implemented with a multithreaded program in C language. 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 enable GINP over Ethernet, IPv4 and IPv6; to add, delete, print GINP static routes; to shape GIN interfaces, to print or reset GIN interface counters; etc.

The GIN Dictionary is implemented in PHP language and runs over an Apache HTTP server providing proxy services. The GIN Dictionary is composed of a Dictionary DB, holding bindings for registered object IDs, and three main modules:

- the "Resolver" module handles resolutions for GINP packets and multicast GIN echo requests (called gpings).
- the "PUT" module implements the registration protocol (client and server side).
- the "GET" module implements reliable end-to-end multisource retrieval protocol (client and server side).

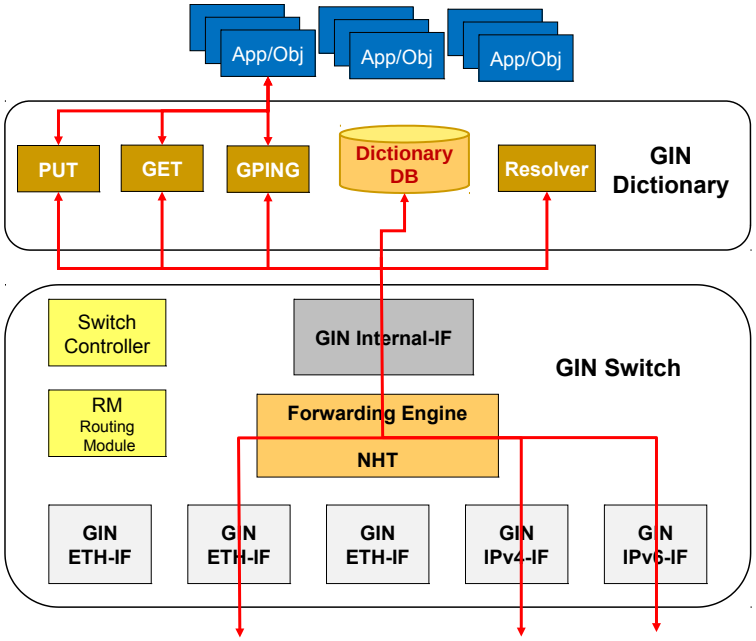


Figure 2.32: GIN prototype architecture

A network testbed of virtual GIN prototype nodes has been setup on a VMware platform. The testbed 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 underlayers (IPv4, IPv6 and Ethernet) and no IP connectivity is provided end-to-end. GINP packets flow seamlessly over different underlayers. GINP provides the common network communication level. Client data objects can be stored in GIN access nodes, and near object copies can be retrieved and gpinged from the GIN access nodes, showing locality and anycast behavior. Current prototype software and documentation have been publicly released and are available on GIN web site http://gin.ngnet.it. 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, 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).

2.9.1 Architecture

GIN view

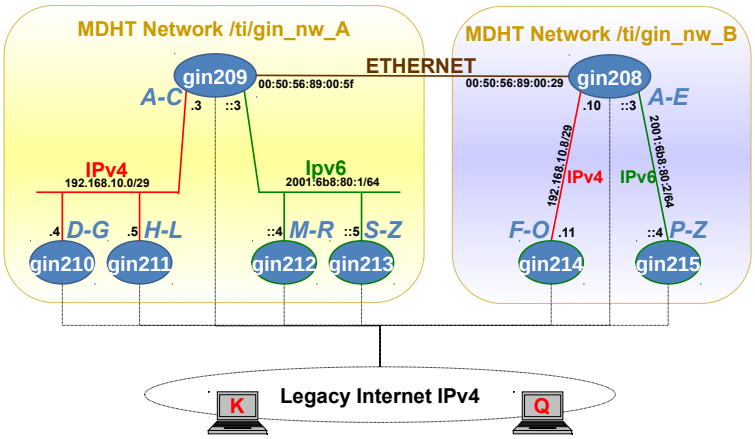


Figure 2.33: GIN testbed setup

- GIN is a hybrid networking architecture for ICN.

  GIN supports both host-centric and information-centric communications.

- GIN interconnects Information Objects, addressed by user-level names.

  GIN supports almost any user-level naming scheme (Internet URIs, NetInf, DONA, CCN, etc.).

- GIN implements a packet-based Name Networking Layer.

  GIN forwards data by name in the global Internet, in packets over L2/L3 heterogeneous sublayers.

  GIN Protocol packets are encapsulated directly in the L3/L2 frames.

- GIN adopts end-to-end transport services.

  Any nearby copy of a named Information Object can be used for direct download.

  Object copies are registered in a Network-distributed Dictionary.

- GIN integrates innovation as it appears in the overlays.

  Services can be added to an open and flexible SDN network platform equipped with storage and processing.

GIN node A Control Plane provides local and global resolution and routing and support for network services (caching, storage, multicast, mobility, etc.). A Data Plane provides fast forwarding on shortest paths with ID switching.

GIN Architecture Overview The GIN network architecture is shown in Figure 2.34. At the core of the architecture is a Name-based Networking Layer. The foundation of the Name Layer is GINP, the core protocol used to connect named objects over GIN. GINP is a stateless delivery protocol similar to IP. Information Objects (GINP endpoints) are identified by means of ID stacks. GINP packets are routed by object IDs though the network. GINP is the unifying name networking layer over heterogeneous L3 or L2 networks. The PUT protocol is used as a signaling protocol between clients and nodes in order to register/update/delete binding information into the GIN Dictionary (i.e., the GIN distributed Name Resolution Database). The GET protocols are request/response protocols, used by the clients to ask objects by name to the network. The MPGET (MultiPath GET) protocol is a multisource retrieval protocol.

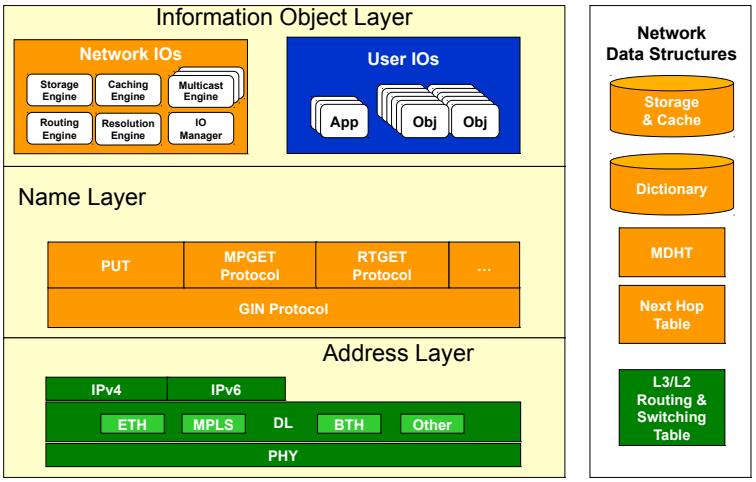


Figure 2.34: Protocol stack of the GIN Architecture

Name resolution and forwarding In GIN, name resolution and forwarding are two distinct but strictly integrated processes. They use different data structures and provide two distinct paths for delivering data and control packets (Figure 2.35).

GIN Resolution Path

- In the Dictionary, Object IDs are mapped to network IDs
- Object IDs are registered by clients with the GIN registration protocol (PUT)
- MDHT maps ranges of Object Ids to next hop Dictionaries if no resolution is locally available

GIN Fast Path

- In the NHT, Network IDs are mapped to L3/L2 next hop information
- Network IDs are advertised in traditional intra/inter-domain routing protocols (e.g., ISIS and BGP)
- Most of the GIN traffic is routed over the Fast Path by means of ID switching

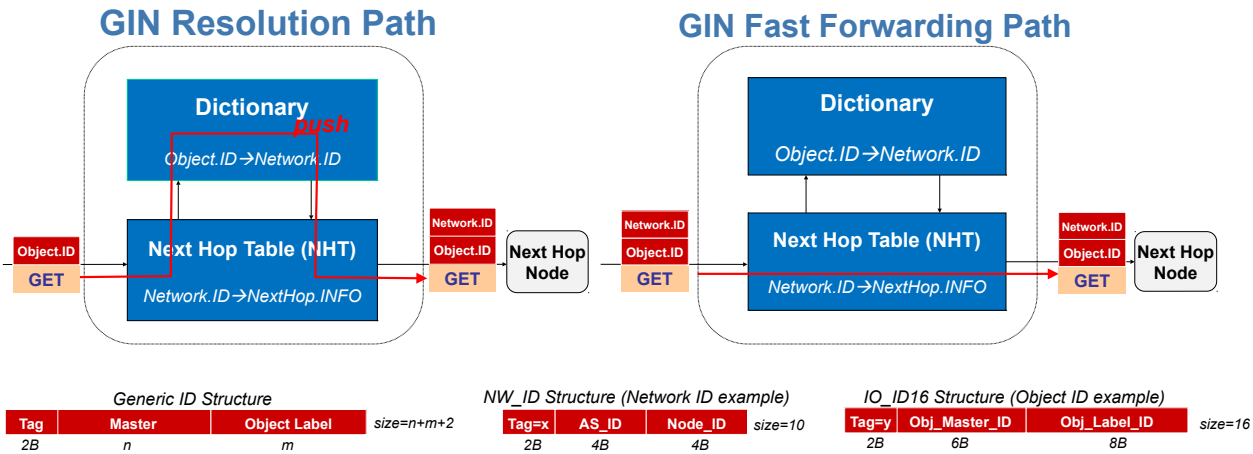


Figure 2.35: GIN forwarding and resolution

2.9.2 Demo

Prototype architecture The GIN prototype is licensed as open software, for demo and testing purposes under Apache License 2.0, over FreeBSD 8.2. The Forwarding system has been written in C language; the Dictionary system has been developed in PHP language.

Testbed setup An example GIN testbed has been set up (Figure 2.33): GIN nodes are interconnected by means of heterogeneous sublayers: IPv4, IPv6, Ethernet. Two double-level Multilevel Distributed Hash Table (DHT) Domains (A and B) are statically configured and provide a distributed Name Resolution System in each network domain .

ICN services on the GIN testbed The GIN prototype testbed provides fundamental ICN services to legacy IP clients by means of GIN proxy nodes.

Registration and Upload (Figure 2.36)

- Client Q uploads object /Q/X to Access Node gin215
- Object /Q/X is registered in MDHT (locally and on another upper level node) and in the Search Engine DB

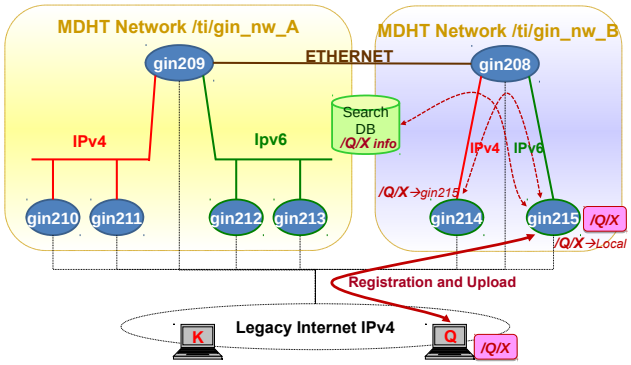


Figure 2.36: Registration and upload

Resolution (Figure 2.37)

- Client K sends a GET request for /Q/X to access node gin211
- Node gin211 sends a GPING request for /Q/X and receives a GPING Reply from gin215

Retrieval (Figure 2.38)

- Object /Q/X is downloaded from node gin215, cached on gin211 and registered in network A and delivered to the requester
- Following requests for the same object in network A will be satisfied locally

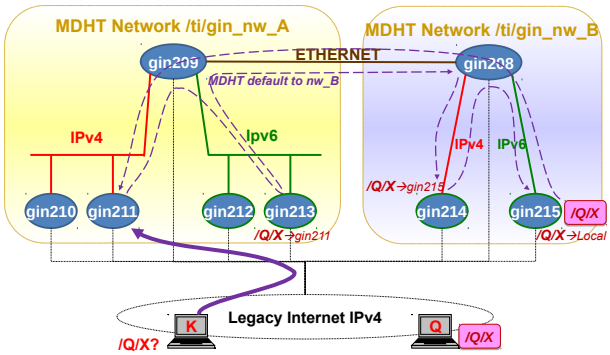


Figure 2.37: Resolution

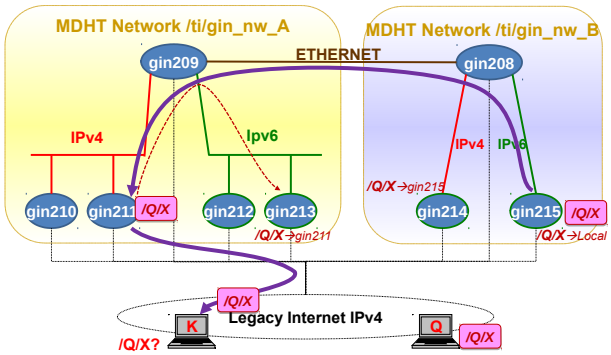


Figure 2.38: Retrieval

An experiment: multisource retrieval performance gains The GIN testbed has been used for a preliminary assessment of the GIN multisource retrieval protocol (MPGET).

Experimental setup (Figure 2.39)

- Node gin208 is downloading object /Q/X from 1 to 4 sources in parallel
- Each source is shaped at 1000 pps in upstream
- Ethernet link to gin208 is throttled at X pps
- A 50 MB file has been retrieved from 1-4 sources in parallel
- The download times and bandwidths are illustrated in the charts (Figure 2.40, Figure 2.41, Figure 2.42)

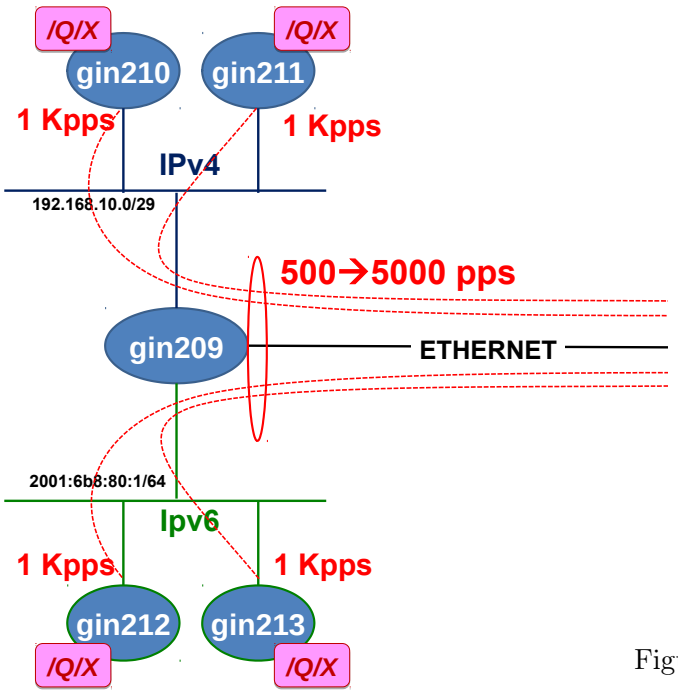


Figure 2.39: Experiment setup

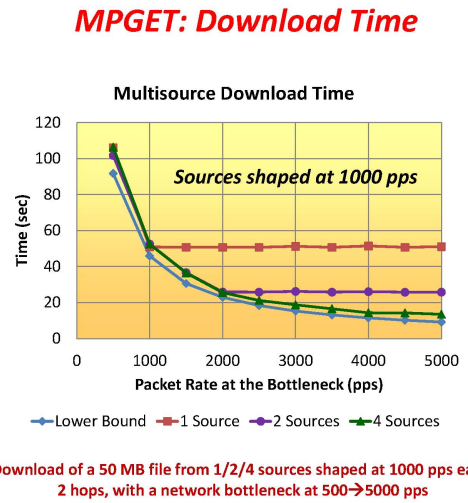


Figure 2.40: Download times as a function of the bottleneck rate and number of sources

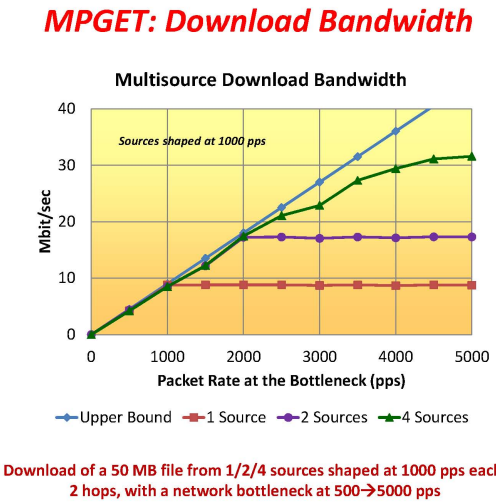


Figure 2.41: Download bandwidth as a function of the bottleneck rate

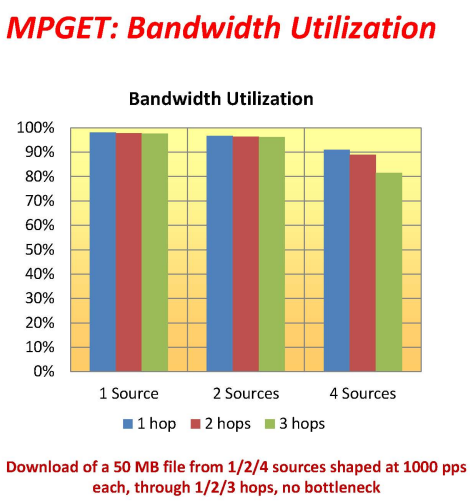


Figure 2.42: Bandwidth utilization on the download link

2.10 NetInf Open Source Software

SAIL has developed a rich set of prototype implementations of the NetInf protocol and corresponding applications. A significant fraction of these implementations have been released under Open Source Software licenses and are used by the ICN community for experiments and new research activities.

NetInf Software on SourceForge The SAIL project has released an open-source (subject to the Apache v.2 license) set of tools for NetInf. These implement various aspects of the NetInf protocol in different languages. At the time of writing, there are C, Python, PHP: Ruby, Clojure and Java implementations with the Python, Puby and PHP code having seen the most development so far.

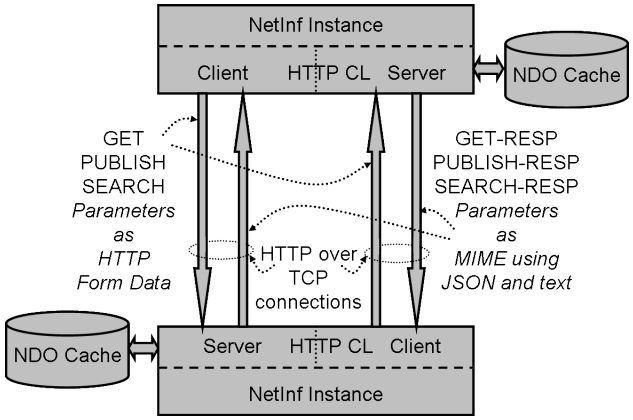


Figure 2.43: NetInf software

OpenNetInf The OpenNetInf prototype building on and extending earlier work from the 4WARD project [16]. The OpenNetInf implementation is a proof-of-concept implementation of the major NetInf elements, including the NetInf API, inter-NetInf-node interface, information model, naming concepts, security concepts, name resolution, caching, and data transfer. The goal is to evaluate the major NetInf design decisions and the overall NetInf architecture in practice. OpenNetInf contains a

hierarchical name resolution system (MDHT-based) and integrated caching functionality. Another focus is on the NetInf API and on the inter-NetInf-node interface. The software contain browser plugins and video streaming software.

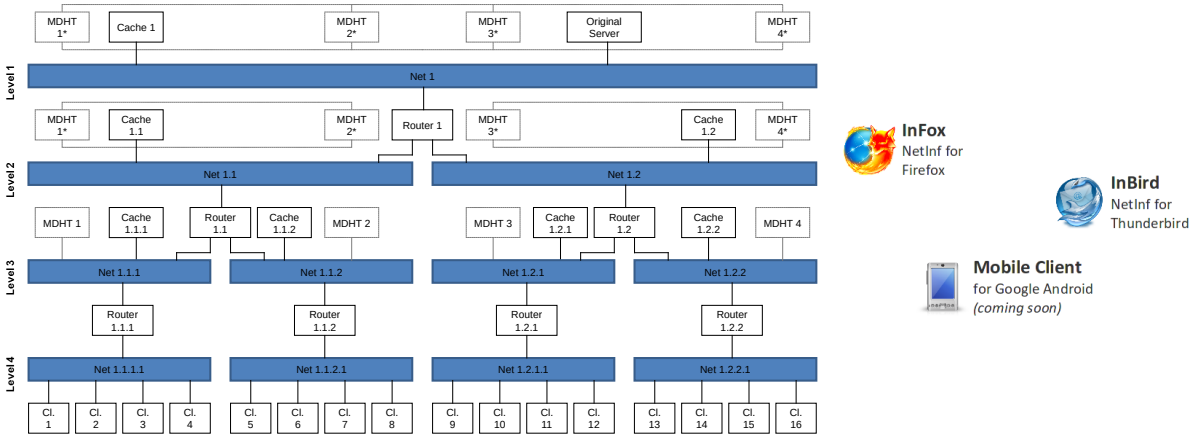


Figure 2.44: OpenNetInf software

Global Information Network (GIN) GIN is a hybrid ICN architecture able to support both dissemination and conversational communication models. It uses a stateless packet-based forwarding protocol, called GINP. GIN aims to interconnect NDOs over heterogeneous L3/L2 underlayers, in the global network, by means of an integrated name-based resolution and routing mechanism.

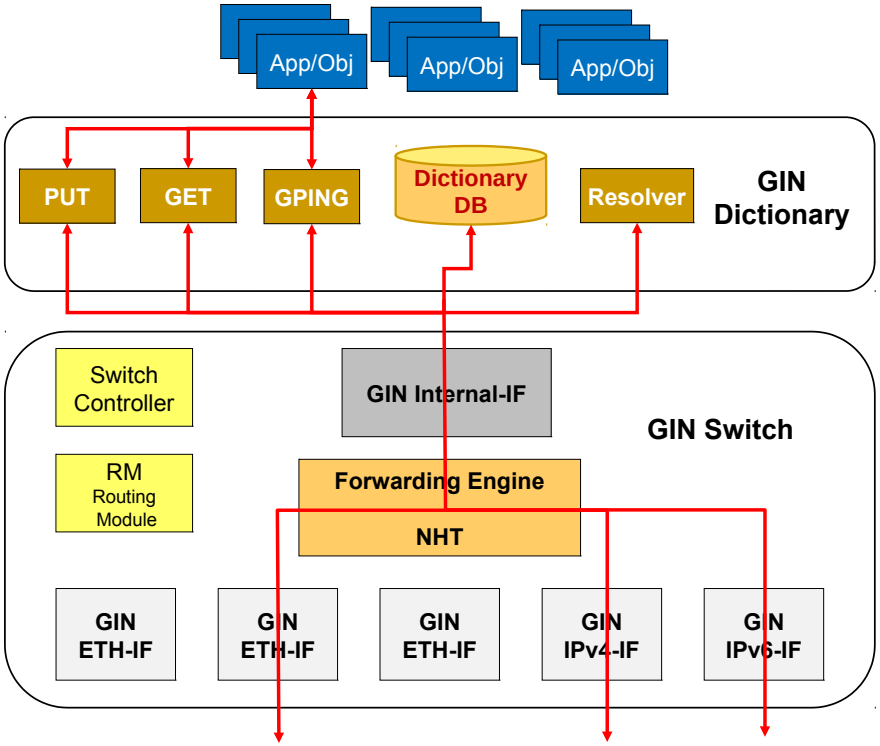


Figure 2.45: GIN software

Android Client Software SAIL has also developed an implementation of the NetInf protocol for the Android mobile OS. The implementation supports publishing or registering NDOs to NetInf infrastructure, sharing NDOs over Bluetooth, and obtaining NDOs over HTTP or Bluetooth CLs. The implementation reuses some components from OpenNetInf. It runs as an Android service and provides a local HTTP-based API to applications so that many applications can benefit from NetInf functionality.

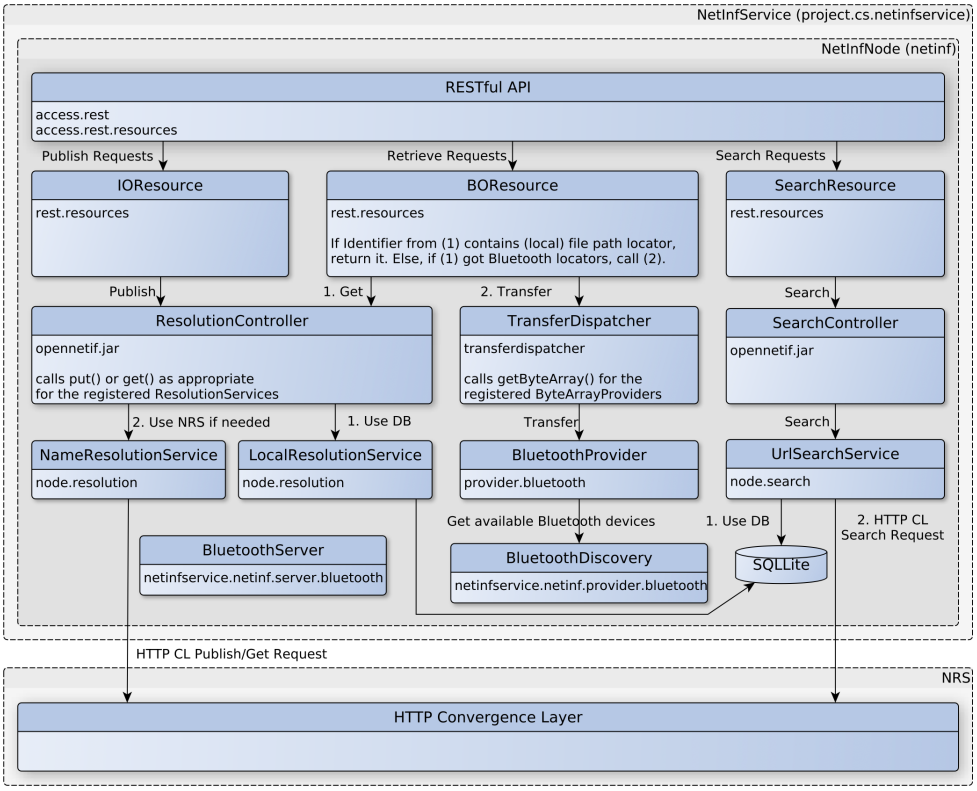


Figure 2.46: Android client software

‘

Appendices

Appendix A: Brochures

NetInf
The Network of Information

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

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

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

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

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

Key NetInf elements have been published as specifications, such as the NetInf protocol speciication [1] — 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 vari-ety 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 [2] and different Convergence Layer specifications.

In the following, we are presenting different prototypes and evaluation scenarios that had been developed by the SAIL project, illustrating different aspects of the NetInf system.

References

[1] 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.

[2] Stephen Farrell, Dirk Kutscher, Christian Dannewitz, Boerje Ohlman, and Phillip Hallam-Baker. Naming Things With Hashes. Internet Draft draft-farrell-decade-ni, Work in progress, August 2012

Contact: Dirk Kutscher
Dirk.Kutscher@neclab.eu

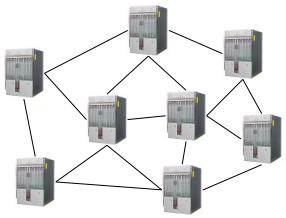
2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

NetInf
The Network of Information

Vision

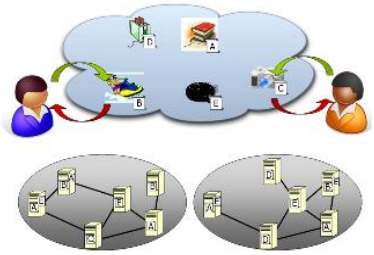
Today's Internet

Focus on
Conversations between Hosts



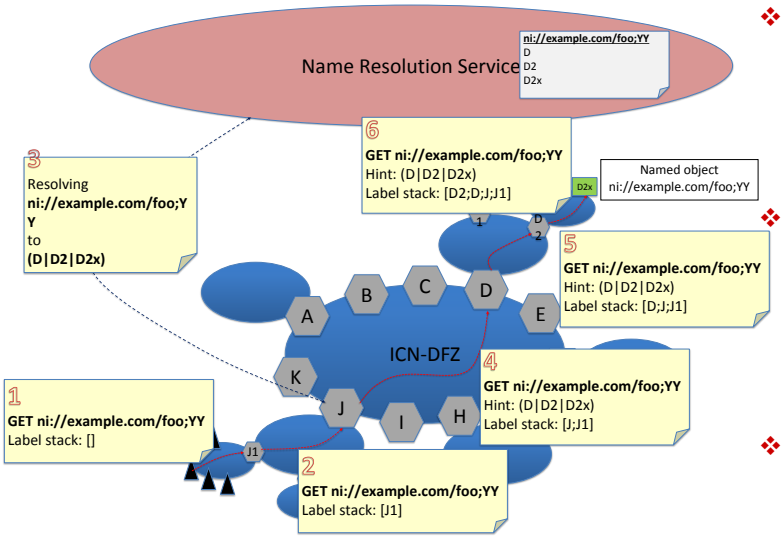
Information-centric Network

Focus on
Accessing named data objects real world objects



In today's Internet, accessing information is the dominating use case!

Approach



Solutions

- **NetInf Naming**
  - Identifying Named Data Object
  - Providing name-content binding validation and other security features
- **NetInf Transport**
  - Information-Centric Internetwork and Transport Protocols
  - Convergence Layer approach enabling migration and network diversity
- **Interdomain Communication**
  - Name-based Routing and Name-Resolution Services

SAIL's Network of Information evolves the Internet to directly support efficient content distribution by introducing accessing uniquely named information as a core Internet principle.

NetInf for Events with Large Crowds

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 demo will show 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.

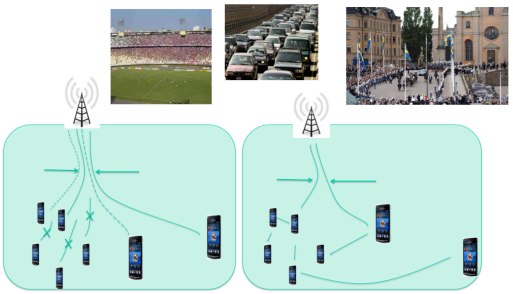
In this demo, we will show an integration of multiple partner prototypes into a workig proof-of-concept EwLC system. In addition to the required NetInf infrastructure (routing, caching, and name resolution), a NetInf system for Android devices has been implemented, and three end-user applications are shown. These are collaborative web-browsing, photo sharing with a visual content directory, and video streaming over the NetInf protocol.

In addition, there is a visualsation server that makes it easier to see what is happening in the network. The visualization server stores and analyzes notifications related to NetInf node signalling, and displays the signals in real-time. The sequence of signals can also be stepped through in a non-realtime display mode. The visualization server provides a network perspective of the signalling between the NetInf nodes, as opposed to a traditional protocol analyzer, which only provides a link local view. The visualization server is useful both for debugging and demonstration purposes.

Contact: *Anders Lindgren Börje Ohlman*
andersl@sics.se borje.ohlman@ericsson.com

Events with Large Crowds (EwLC) NetInf Demo Overview

BASIC PROBLEM TO BE SOLVED



The common problem that we illustrate with the EwLC scenario is when the wireless infrastructure can not support the communication needs of a group of people. In certain situations, when the group wants to access the same set of content items sharing them locally using NetInf can be a very powerful technique.

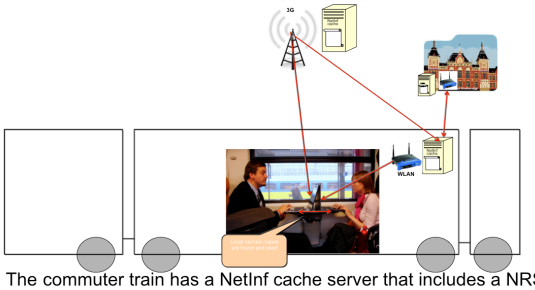
DEMO SCENARIO

Alice is on her way home from work, browsing the web, when…

… she decides to to see tonight's game of her favorite football team

After the game she likes to watch the after-game-talkshow that some of her friends are making using their Android phones and distributing via the NetInf network

COMMUTER TRAIN



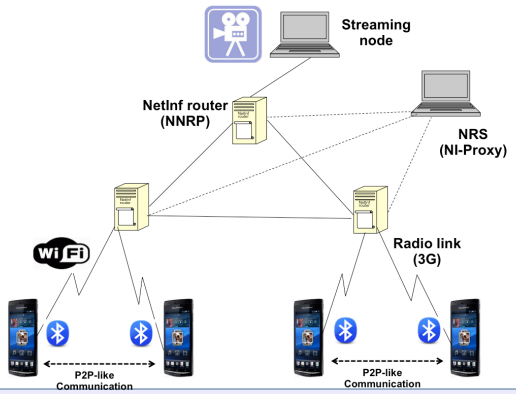
The commuter train has a NetInf cache server that includes a NRS. It's both contains pre-cached material and such that is being cached from user traffic. Devices onboard the train can download via 3G (when available), get objects from the cache and exchange objects directly via, e.g. Bluetooth, in a p2p fashion.

STADIUM SCENARIO

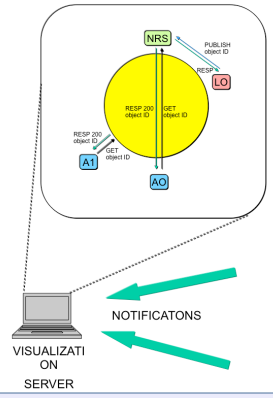
The stadium often hosts events that gather large crowds. As people come there for the same events, they have similar interests in the content they consume. They are likely to request information about the game, share locally produced media content from the event, and watch video streams from the event

The stadium has 3G coverage, and its infrastructure includes WiFi networks that are connected to a local NetInf infrastructure of caches and NRS, helping to reduce the otherwise high load on 3G and WiFi infrastructure. Network load can be further reduced by sharing content over local communication channels to people in adjacent parts of the stadium.

DEMO CONFIGURATION



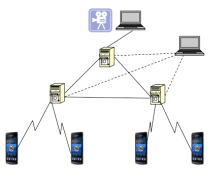
VISUALIZATION TOOL



The visualization server stores and analyzes notifications related to NetInf node signalling, and displays the signals in real-time. The sequence of signals can also be stepped through in a non-realtime display mode.

The visualization server provides a network perspective of the signalling between the NetInf nodes, as opposed to a traditional protocol analyzer, which only provides a link local view.

The visualization server is useful both for debugging and demonstration purposes.

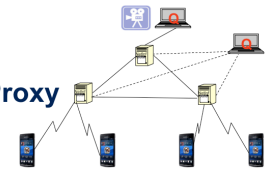


2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

NetInf for Events with Large Crowds - Physical Node Demo

Overview

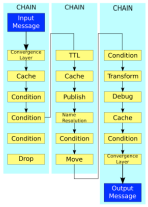
- To showcase some of the potential for NetInf in an event with large crowds, a proof-of-concept system has been implemented with components from different partners
- Routing provided by NNRP
- NRS provided by NiProxy
- NetInf for Android as mobile clients



NetInf in the Infrastructure

NNRP

- Modular NetInf router platform
- Provides routing and on-path caching



NiProxy

- Name Resolution Service
  - Including meta-data search
- Content store
- Mobile nodes register NDOs here

NetInf for Mobile Android Devices

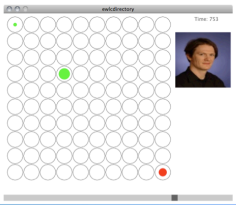
- Implementation of NetInf protocol for Android mobile OS
- Publish or register NDOs with NiProxy
- Share NDOs over Bluetooth
- Get NDOs over HTTP or Bluetooth CL
- Runs as Android service and provides HTTP-based local API to applications
- Allows anyone to create a NetInf-enabled Android app
- Three examples below

Collaborative web browsing

- When a web page is requested, the NiProxy is searched for an NDO corresponding to that page
- If found, the NDO is retrieved from appropriate locator (either NiProxy, NNRP cache, or over Bluetooth from mobile peer
- If not found, the web page is retrieved using legacy HTTP. An NDO is then created and registered with the NiProxy for future requests
- User selects if retrieved content should be shared at central repository and/or locally
- Status indicators show what source a web page is being retrieved from

Local photo sharing and visual content directory

- Problem: How does a user know what locally produced content is available?
- A user may for example be interested in photos of a goal occurring at a certain point in time
- Each user is assigned a location in a stadium environment (seat number
- Content that is shared by this user is tagged with metadata indicating the location of the user
- Users publish photos as NDOs at the NiProxy
- Visual content directory allows browsing of available content over space and time
- Use slider to show which locations published NDOs at different times
- Tap the seat location to retrieve NDO



NetInf Video Streaming

- Shows that streaming can be done over NetInf
- Video recorded and published as NDOs using local NiProxy at video source
- Streaming client implemented for Android devices
- Reassembles video NDOs and launches video player
- More details in separate poster

2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

SEVENTH FRAMEWORK PROGRAMME

Events with Large Crowds – NetInf Live Streaming demo

Overview

NetInf Live Streaming demo for The Event with Large Crowd (EwLC) scenario has some key benefits of NetInf over previous networking architectures. This demo will show some of the benefits that makes NetInf a excellent platform for ad-hoc video distribution as well as an alternative infrastructure for regular media broadcast. Some of the key features include:

- Any node can be the source of a live stream
- No advance or special configuration of the network is needed as NetInf natively supports multicast functionality through caching and request aggregation. Flash crowd problems are thus avoided.
- Each viewer can independently choose to watch stream live, or from the beginning. Pausing/timeshifting the stream is also supported.
- Stream chunks can be retrieved from any node in a p2p fashion

Technical detail

Each stream is named by a stream identifier which is constructed by hashing the stream name (e.g. a human readable filename). The chunk names are constructed by appending the chunk number to stream id. The chunks are grouped into blocks that are signed. The block size is recorded in the metadata of the NDO identified by the stream id (which also contains information such latest produced chunk). The meta data of each chunk NDO contains the signed block digest and the digests of the other chunks in the block. This allows for verification of each chunk independently immediately when received. For details, see [1].

For a user to connect to a stream:

1. Hash the name of the stream to get the stream NDO ID
2. Request the stream NDO
3. Decide where to start playing the stream.
    1. Live: chunk=current
    2. Start: chunk=1
    3. Starting from minute x: chunk=x*(chunklength/min)
4. Request subsequent chunks

Responding to a stream request:

1. When responding to a GET request for the stream NDO, that NDO MUST be marked as non-cacheable.
2. When responding to a GET request for the stream-chunk NDO, that NDO MUST NOT be marked as non-cacheable.

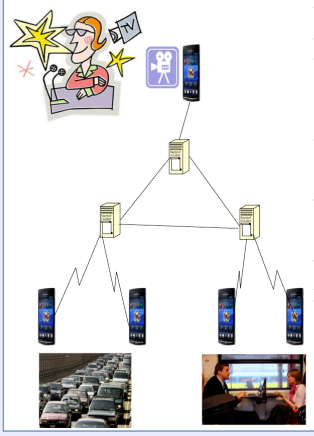
All nodes MUST understand the non-cacheable marking. There is a trade-off between the block size and delay. The larger block size the longer delay before the block can be transmitted. On the other hand the larger the block the less processing overhead (and delay) due to the signing process.

Contact: *Karl-Åke Persson* *karl-ake.persson@ericsson.com* *Börje Ohlman* *borje.ohlman@ericsson.com*

[1] C.Wong, S. Lam, *Digital Signatures for Flows and Multicasts*, IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 7, NO. 4, AUGUST 1999].

Events with Large Crowds – NetInf Live Streaming demo

STREAMING SCENARIO

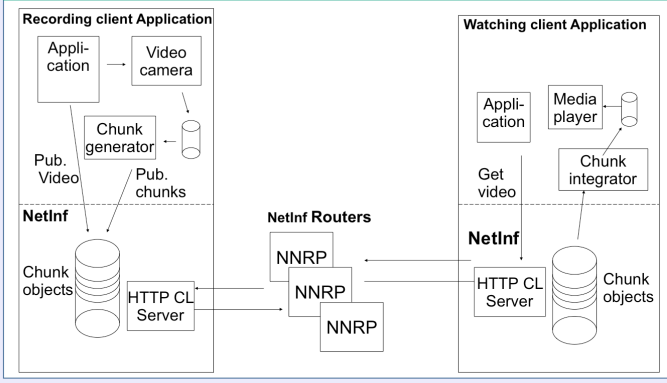


As described in the overall demo scenario Alice likes to watch the after-game-talkshow that some of her friends are making from a bar nearby the stadium. Half a year ago it was only Alice and a few friends that was watching the show, but now, after a big game there can be up to 10 000 people watching the show while commuting home or from the couch in their living rooms. Her friends are only using their Android phones to live stream the show. Thanks to the NetInf enabled network the streaming scales transparently from 10 to 10 000 viewers without putting any significant strain on the network.

FEATURES OF NETINF STREAMING

- Any node can be the source of a live stream.
- No advance or special configuration of the network is needed as NetInf natively supports multicast functionality through caching and request aggregation. Flash crowd problems are thus avoided.
- Each viewer can independently choose to watch the stream live, or from the beginning. Pausing/timeshifting the stream is also supported.
- Stream chunks can be retrieved from any node in a p2p fashion.

NETINF LIVE STREAMING IN THE DEMO



Recording client

- Video captured and stored in file
- Extract chunks from file and publish in local cache
- Update "latest chunk" info in main video file

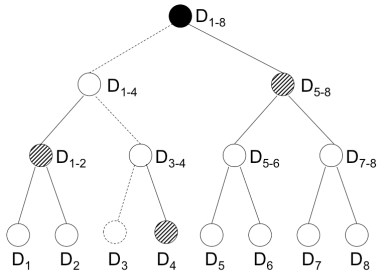
Watching client

- Application ask "local NetInf node" get video
- Local NetInf node sends get video main object
- Recording client returns main object including Latest recorded chunk
- Watching client starts retrieving chunks
- Chunk integrator reassembles chunks to media file
- Media player start reading media file and play out content

NAME-DATA-INTEGRITY FOR NETINF STREAMING

Key issue: Name-data-integrity lost with sequential chunk numbering
Resolution: Sign the chunks
New issue: Signing individual chunks is computationally heavy
Resolution alternatives:

- The chunks are grouped into blocks that are signed. The block size is recorded in the metadata of the NDO identified by the stream id. The metadata of each chunk NDO contains the signed block digest and the digests of the other chunks in the block. This allows for verification of each chunk independently immediately when received. For details, see [1].
- In the future in many scenarios signing individual chunks might be feasible (we like IETF PPSP WG stays open for both these options).
- For applications that have its own security mechanisms at higher layers signing might not be needed, e.g. like distribution of broadcast TV with dedicated set top boxes.



[1] C.Wong, S. Lam, *Digital Signatures for Flows and Multicasts*, IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 7, NO. 4, AUGUST 1999].

2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

NetInf Multi-partner Testbed Configuration and Management

The NetInf Testbed runs on machines dedicated by the partners to execute a set of NetInf prototype nodes. Targeting a large testbed running several hundreds of NetInf nodes, virtualizing nodes is the only feasible option. Thus WPB created an lxc image that allows to run different configurations of the NEC NetInf Router Platform (NNRP) along with a setup and configuration framework. Using this framework it is possible to deploy 50-150 interconnected nodes on a new machine and integrating them into the existing testbed. This should not take more than 5min. The framework allows easy updates to the installed NNRP binaries.

At a high level the testbed consists of two tiers. The inner tier connects the different partner sites over the Internet. Each sites is represented by one gateway node (GW) in this tier. The GWs primary task is to connect the different partner sites. In the outer tier the partners can deploy their preferred configuration and use the GW nodes to reach remote nodes and content. The suggested option is to connect the virtualized nodes running in a machine via a dedicated access point (AP) to the GW. That way it is easy to add both new NetInf Testbed sites and more machines to a specific site. To ensure reachability in the inner tier the TestBed relies on the Routing Hints extension to NNRP (which has been developed by SICS, see also Routing Hints poster for more info).

At the moment the Testbed is configured with the EwLC emulation scenario in mind (see also EwLC emulation Demo poster). For this all the nodes are not only connected to the AP, but also in adhoc groups with each other. To make the EwLC scenario more realistic traffic shaping is used to emulate the bandwidth and delay of 3G and Adhoc WLAN networks. The current configuration framework instantiates a configuration in which both the networks are used and the in-network caching capabilities of NetInf are leveraged. Targeting the EwLC emulation we make the following assumptions:

- Objects are published only on nodes and locators are forwarded to APs which store them
- APs and GW for caching and routing (Naming scheme reflects organizational context)
- APs fetch the object when matching a locator and serve object, instead of only returning the locator
- When a node receives a request it will first broadcast it in it Adhoc group (UDP, ½ sec timeout), and if unsuccessful forward it to the AP.
- The AP in turn check his caches and locators and if that fails uses the routing hints to determine the nexthop. This continues until the object is found.

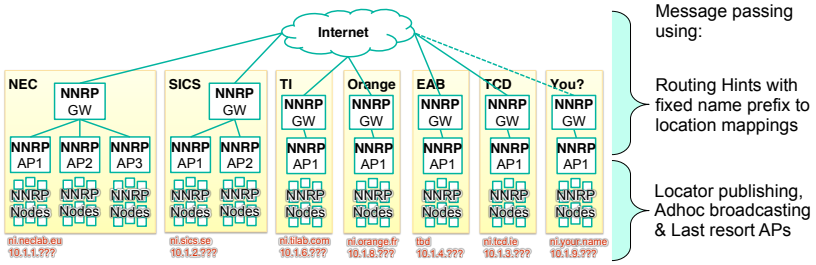
To allow for centralized object publishing and inserting requests at dedicated nodes, the configuration framework also features a control network that can be extended across different machines and sites using L2 tunnels (in our case ssh).

Contact: *Fabian Schneider. Andreas Ripke, Dirk Kutscher* *Bengt Ahlgren*
first.last@neclab.eu *bengta@sics.se*

NetInf Multi-partner Testbed Configuration and Management

Testbed Topology & Message passing mechanism

- Each site running virtual network and virtual nodes (lxc)
- NEC NetInf Router Platform (NNRP)
- SICS Router Module for NNRP
- Orange Transport Module for NNRP

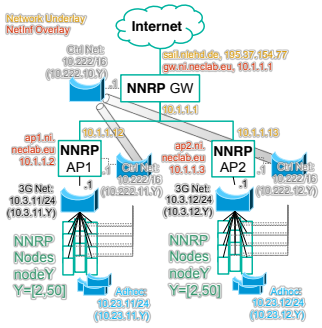


Testbed operation assumptions

Current testbed was configured with EwLC use case in mind. Configuration can be changed for different use cases

- Objects are published on nodes only; Locators are forwarded to APs
- APs and GW for caching and routing (Naming scheme reflects organizational context)
- APs fetch the object when matching a locator and serve object, instead of only returning the locator

Single Location Setup



- Only GW needs Internet access
- GW can reside on same machine as AP+Nodes
- One large L2 control network connected via tunnels (ssh)

NetInf GET Message Processing

Stop if any of the step yields the object:

1. Node checks local caches
2. Node broadcast request to adhoc network
3. Forward request to AP, await response
   a) AP checks locator database
      - On hit GET the object from locator, put in cache and serve back
   b) AP checks local caches
   c) AP uses routing hint to locate next hop
      - Can be another local AP or the GW

Routing example

GET for ni://ap3.ni.neclab.eu/sha-256;4….

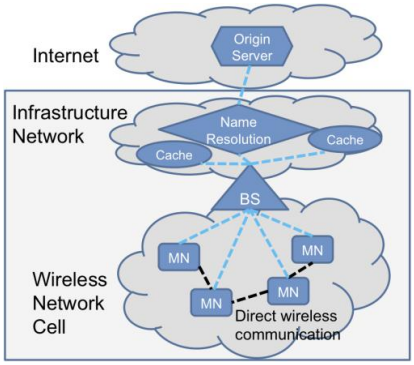
1. Prefix resolves to 10.1.1.4
2. TI AP → nexthop is TI GW
3. TI GW → nexthop is NEC GW
4. NEC GW → nexthop is NEC AP3
5. NEC AP3 knows where to get the object local

Management Tools

- Scripts to start and stop virtual machines from single lxc image → many nodes possible
- Scripts to configure lxc, NNRPs and network on each machine
- Framework to publish and request objects and query cache status

NetInf EwLC Emulation Demo Caching & Adhoc networking Benefits

Idea



- Emulating specific network setups to evaluate NetInf protocol performance in different load scenarios
- Motivation: running real code in controlled environment for more meaningful and accurate evaluation
- Event with Large Crowd: Many mobile NetInf nodes connected to wireless infrastructure network and enabled to communicate locally
- Configuring different mobility patterns, publish/requests patterns, popularity distribution etc.

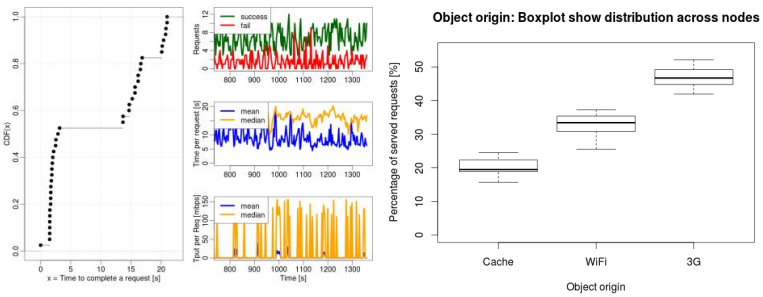
Approach

- Use the NetInf Testbed as execution platform
  - NNRPs in LXC virtual machines, 50+ nodes per machine
  - Multiple physical machines to scale up network
  - Testbed APs correspond to BS, nodes to MN, GW for connectivity, Remote location nodes serve as origin servers
- Emulate two networks per node + Control to issue request
  - 3G: connecting all nodes of a BS, traffic shaped to 7.2Mbps/150ms
  - Adhoc: connecting 8-12 nodes directly, traffic shaped to 54Mbps/2ms
- Script node behavior, request generation, communication link and storage constraints to ensure reproducible experiments
- Collect performance measurements for real-time and offline evaluation
- (Not in demo) Mobility through changing Adhoc group memberships

Lessons learned:

- NS3 is too slow to emulate Adhoc WLAN connectivity and/or mobility for 20+ nodes
- NNRP can process (relay) a request in less than 20ms
- LXC very light-weight and scalable

Results



- Significant offload potential through ICN in-network storage and NetInf local communication
- Mobility can be a feature: disseminating locally generated or cached data objects
- Live demo at SAIL NetInf event

Contact: Fabian Schneider, Andreas Ripke, Dirk Kutscher
first.last@neclab.eu

2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

NetInf Global Routing Using Hints

The global routing mechanism for the NetInf protocol makes use of two levels of aggregation in order to achieve a high level of scalability. The mechanism is an adaptation of the Global Information Network Architecture (GIN) [1] and Narayanan and Oran's ideas [2] to the NetInf protocol.

The mechanism is mainly concerned with the global aspects of routing requests for Named Data Objects (NDOs) towards the corresponding publisher, i.e., routing in the NetInf default-free zone, comparable to the current Internet's BGP-routed infrastructure. Just like in the current Internet, edge domains can utilise different routing schemes that are adapted to particular needs of the respective domain.

NDO aggregation

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

We thus introduce the notion of NDO aggregation, meaning that a set of NDOs are grouped together. For routing and forwarding purposes, the NDOs in an aggregate are then treated the same. Such NDO aggregates, with the same origin, occur naturally in reality, for instance, chunks of a video, photos in a collection, individual objects on a web page and/or site, and so on. NDO aggregation increases performance in that a name resolution cost need only be taken for the first NDO of the aggregate. It likewise increases scalability in that routing information is only needed for the aggregate as a whole. We use the authority part of the ni: URI [3] to name NDO aggregates.

Components

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.

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: URI and/or routing hints into the address of the next hop to forward the request to.

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

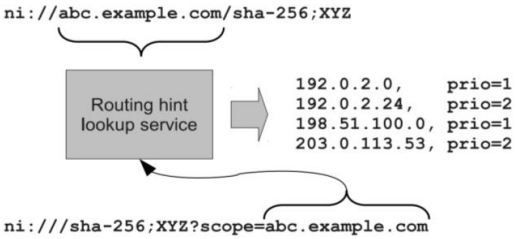
[2] A. Narayanan and D. Oran. Ndn and ip routing – can it scale? Presentation at ICN side meeting at 82nd IETF, November 2011. http://trac.tools.ietf.org/group/irtf/trac/attachment/wiki/icnrg/IRTF

[3] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", draft-farrell-decade-ni-10 (work in progress), August 2012.

NetInf Global Routing Using Hints

NDO aggregation and routing hints

- Adaptation of the Global Information Network Architecture (GIN) and Narayanan and Oran's ideas to the NetInf protocol
- Aggregation of routing information key to scalability
- Named Data Objects (NDOs) are grouped into aggregates
- NDO aggregates are mapped to routing hints thought a lookup service (can be DNS)
- Multiple hints with priorities
- Global routing only needed for lowest priority hints



Components

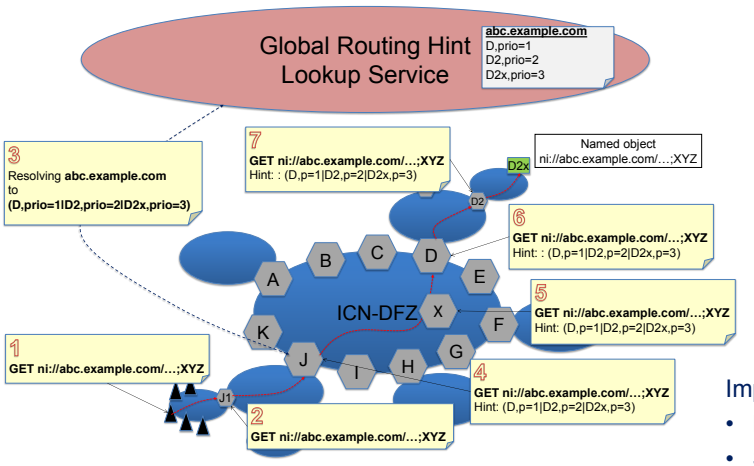
- Routing hint lookup service
- NetInf BGP routing system for hints
- Forwarding tables at NetInf nodes – one or both of ni: name table and routing hint table

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

Forwarding process

- Check the object table (cached and permanently served NDOs)
- Check ni: name forwarding table; if match, forward to that next-hop
- If needed, perform lookup of routing hints
- Lookup all hints in routing hint forwarding table; if match, forward to next-hop of hint with highest priority

Example



Forwarding tables:

J1	
Routing hint	next-hop
Default	J

D	
Routing hint	next-hop
D2	D2

J	
Routing hint	next-hop
D	X

D2	
Routing hint	next-hop
D	D
D2x	D2x
Default	D

X	
Routing hint	next-hop
D	D

Implementation:

- NEC NetInf Router Platform (NNRP)
- SICS Router Module for NNRP

Contact:
Bengt Ahlgren,
bengta@sics.se

2013-02-13

www.sail-project.eu

SCALABLE & ADAPTIVE INTERNET SOLUTIONS

SEVENTH FRAMEWORK PROGRAMME

A NetInf Congestion Control Protocol

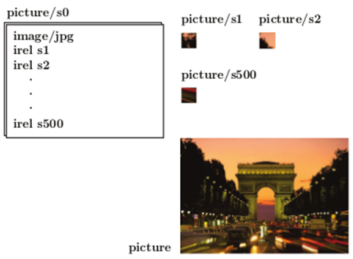
Objectives:

- Congestion control
- Error recovery
- Reassembly

Integration in SAIL:

- Developed as a NNRP module
- Integrated to the Event with Large Crowd emulation scenario

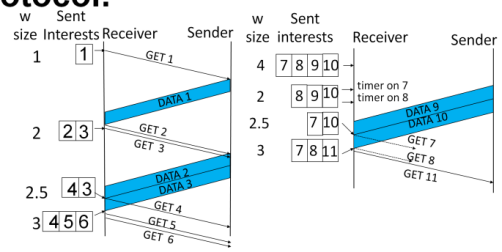
Chunking:



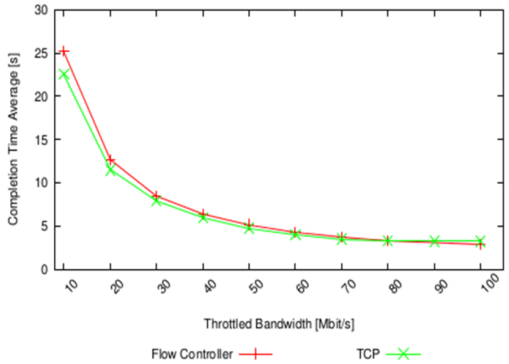
Key design choices:

- Receiver-based
- Object published as a set of small chunks
- Chunks are regular NDOs
- AIMD window controls the rate of GET messages
- Re-requests unanswered GET

Protocol:



Single source:

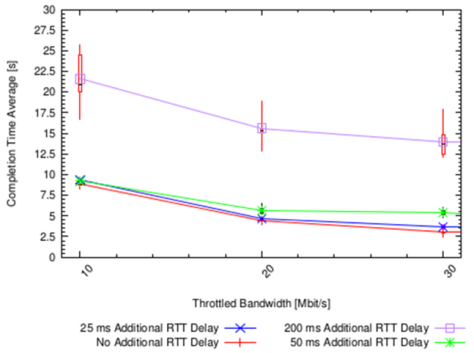


Testbed:

POC for multipath:

- Developed as a NNRP module
- Round-robin forwarding based on a set of next hops

Three sources:



References:

• G. Carofiglio, M. Gallo, L. Muscariello, **ICP: Design and Evaluation of an Interest Control Protocol for Content-Centric Networking**, INFOCOM NOMEN 2012.
•G. Carofiglio, M. Gallo, L. Muscariello, **Multipath Congestion Control in Content-Centric Networks**, INFOCOM NOMEN 2013.

Caching in a Network of Informations (with visualisation)

Network caching of data retrieved from a server has been investigated both from a research and industry perspective. From work in the early 90's on web server caching, to more recent work (VoD file distribution and 3G access networks) caching has become a viable solution within the network to save resources. In this demonstration, we show how to cache objects in an Information Centric Network. The significant difference between an ICN model and today's networks is that each web object has a unique identifier that is used to create, locate, distribute and verify the object. The overwhelming advantage is that the traditional client-server model can be split allowing objects to moved, replicated, cached without an end-to- end connection. Additionally metadata can be used to store attributes about the object. We leverage this architectural to design a new type of cache management (or eviction) policy. We use the metadata field to store the hitrate of each object and compare this field to the other cached items when deciding which object to replace as caches fill. We compare our approach, called FMD, to LRU, LFU and random replacement policies.

We consider a hierarchy of caches in a tree structure with a source server at the top and clients requesting Nmaed data Objects (NDO) at the leaves. The basic concept is to decouple the client-server approach by allowing 'self-certifying' objects to be be transported rather than the TCP bitstream for reliable traffic or the UDP datagrams in unreliable transfers. Obviously, it is desirable to store highly popular content in caches close to the clients and less popular NDOs higher up in the cache hierarchy in order to reduce overall latency and load on the network. We assume that the caches have limited storage space and an eviction policy is applied.

We simulate a hierarchical infrastructure to determine which NDO's to evict from a full cache when a new NDO is visible to the cache. Two of the simplest and also most widely used policies are Least Frequently Used (LFU) and Least Recently Used (LRU). The LFU selects the IO with the lowest hit rate while LRU evicts the IO that has not been requested for the longest time.

A problem with LFU and LRU is that these policies tend to evict potentially popular content before they have an opportunity to grow popular when storage space is small compared to the amount of IOs visible to the cache. To mitigate this effect caches higher up in the hierarchy are provided with larger storage space. This, however, works to a limited extent since the cache memory becomes to large and latency for searching the cache becomes prohibitive. Ideally, the most popular content should be stored in the cache closest to the client while the next popular content is stored at the next level in the cache hierarchy etc. Furthermore, one could argue that once popular content has been distributed out to the caches close to the edge of the network this content could be expunged from caches higher up the hierarchy to provide storage space for new content. To achieve this we propose to attach meta-information about the popularity of the IO when it is sent from a source towards the requesting client. The meta-information is updated at each cache and the updated meta-information is used by the re-placement function. We call our proposal Forward Meta Data (FMD). At each cache on the path to the client the meta data is examined and if the popularity of the IO is higher than the least popular content in the cache the lois replaced with the newly arrived IO.

We will show this dynamic behavior through a poster explanation and a demonstration based on a Java simulator.

Contact: Ian Marsh
Email: ianm@sics.se

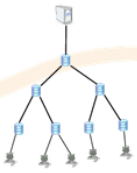
Caching in a Network of Information (with visualisation)

Caching in a Network of Information (with visualisation)

SWEDISH ICT SICS

Basic problem

- Investigate caching policies in a NetInf
- In a system with many edge nodes & few storage servers
  - Form a tree-like structure with requesters at the leaves
  - All items are stored in the sever
- Distribute copies of the items within the tree
  - To minimize the access time of the data items
  - Reduce the load on the (few) servers
  - And on the bottleneck link
- Classical problem is which items to evict as the caches fill
- Other works investigate the policies and system requirements



Network of Information caching

- The NetInf idea is to decouple self-contained objects from servers
- Popular items should be cached as above
- Important mechanism is that popular items should be **replicated**
- Since objects are self-contained the access to each needs to be kept
- The metadata is one place to hold the number of accesses
- We call this Forward Meta Data (FMD)

FMD algorithm

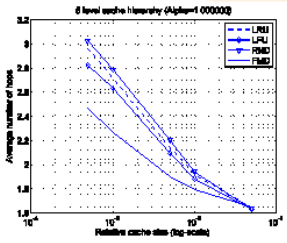
- Requestors at leaves request an object
- A search is conducted up the tree
- If the item is found its access count is updated by one
- On delivery to the client its access count is compared to the lowest hit rate in the cache
- If its hit rate is higher than the lowest hit rate, it replaces the cached item
- Repeated for each cache down the tree

Simulation/visualisation environment

- A custom C++/Processing environment
- Implemented Least Recently Used (LRU), Least Frequently Used (LFU) , Random replacement policy (RND) and Forward Meta Data (FMD)
- Assume items follow a popularity distribution given by Zipf's law z(k; α) = 1/k^α

Results

- Show the average number of hops for cache sizes (relative to document size)
- Show the effect of the distribution of the popularity (α)
- Table av. rank of document in memory at each level (100,000 docs, 100 cache size)



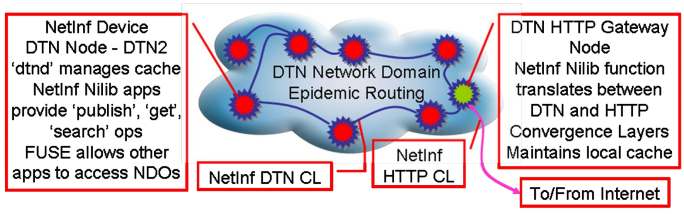
	LFU	LRU	RND	FMD
Level1	43439	66729	72171	183
Level2	63705	66357	65619	164
Level3	72756	64124	63547	145
Level4	64159	66345	66459	66

2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

SEVENTH FRAMEWORK PROGRAMME

Netinf: Using DTN and Nilib

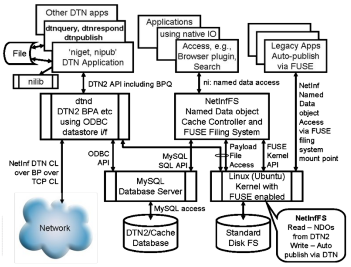
NetInf DTN Convergence Layer and DTN↔HTTP Gateway



- NetInf DTN Convergence Layer (CL) integrates with HTTP (and UDP) CLs
  - draft-kutscher-icnrg-netinf-protocol
- Nilib code works with DTN2 OSS reference implementation
  - Uses BPQ and Metadata blocks
  - C and Python interfaces

http://sourceforge.net/projects/netinf/ and **...projects/dtn/**

NetInf ICN Device

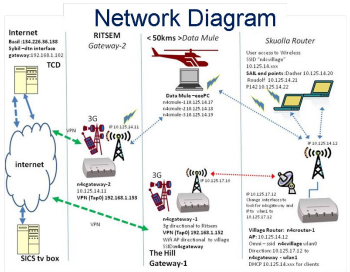


- Overall architecture
- OSS Components
  - DTN2
  - NetInf Nilib
  - FUSE userspace file system
- Aims at ICN/DTN as sole communication mechanism

NiLib Python

- NetInf HTTP CL with GET forwarding
- PUBLISH, SEARCH direct
- Filesystem for NDO storage
- Filesystem or REDIS "NOSQL" DB for NDO meta-data
- Apache module install (via WSGI) or Python HTTP server
- Running on tcd.netinf.eu SAIL testbed

DTN in use: SAIL Summer 2011 Trial – Padjelanta in Arctic Sweden – TCD and SICS



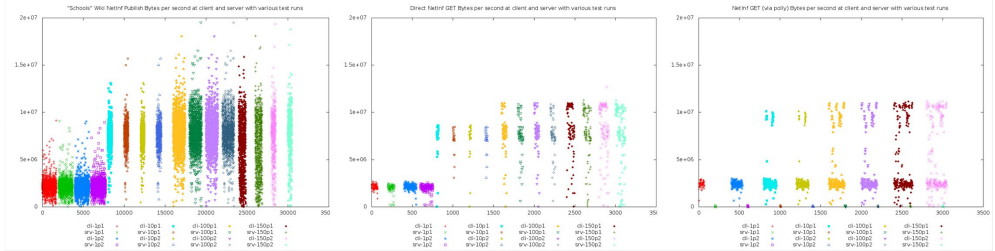
- Goals:
  - Trial improved h/w & s/w after N4C
  - Demos of Twitter, facebook, email over DTN for local reindeer herders
  - First experiments with additions to DTN bundle protocol for ICN
    - draft-irtf-dtnrg-bpq

Work in progress

NDO (bundle) transport

http://down.dsg.cs.tcd.ie/s11inf

NetInf NiLib Python Performance Test Harness



3.8GB (125,000 files) wikipedia snapshot at tcd.netinf.eu; PUBLISH, direct and in-direct GETs

Contacts:
Stephen Farrell, Elwyn Davies
Stephen.Farrell@cs.tcd.ie
davieseb@scss.tcd.ie

2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

Netinf: Using DTN and Nilib

The NetInf DTN Convergence Layer

The NetInf Protocol [1] adopts a Convergence Layer (CL) architecture that allows Named Data Objects (NDOs) to be transported between pairs of nodes using a CL protocol that is chosen for the nodes' network domain. The NetInf protocol is well-suited for moving NDOs across disparate domains using gateways at the domain boundaries where they choose a CL protocol for the next transport hop: an incoming request using (say) the HTTP convergence layer in the well-connected Internet could be forwarded into a domain where Delay- and Disruption Tolerant Networking (DTN) is appropriate.

The work represented by this demonstration shows how a NetInf CL can be implemented on top of the DTN Bundle Protocol (BP) using the Bundle Protocol Query Extension Block (BPQ block) developed during the SAIL project to carry the NetInf specific information [3] in conjunction with Metadata Extension blocks.

The DTN architecture supported by the BP [4] is described as a 'store, carry and forward' so that all DTN nodes already support a cache for bundles that are in process of forwarding by the BP. Unlike conventional routers, these copies are not necessarily transient but may be held for extended periods of time to allow for the delays and disruptions foreseen in a DTN environment. During the SAIL project we have added functionality to the DTN2 bundle protocol agent (BPA) daemon ('dtnd') that uses the information in the BPQ block of bundles carrying NetInf requests to either 'publish' an NDO by retaining the content in its cache or create a response to a 'get' request from this cache when the NDO name matches with the NetInf request parameter.

HTTP↔DTN CL Gateway

In addition to the HTTP CL components in the NetInf Nilib OSS, the repository now includes extensions to the standalone NetInf server that accept NetInf protocol messages carried over the DTN CL together with extended applications that can select the CL to use according to the format of locator hints provided.

NetInf DTN Device and FUSE NetInf Cache Access Component

Computers used in the demonstration as DTN nodes are setup to manage external data access as far as possible using only NetInf access over the DTN BP. NDOs cached by the DTN BPA (dtnd) on these nodes can be read using their ni URIs as file names through a FUSE filesystem. Local applications can write files into this filing system which will be automatically published as they are completed ('closed') and will then be accessible either via the correct ni URI name or the local name used when they were created.

NetInf NiLib Python Performance Test Harness

Nilib [7] is a set of open-source (Apache licensed) implementations of the NetInf Protocol developed as part of the SAIL project with various language bindings including C, PHP, Python, Ruby and Java. We use the Python client and Apache server implementation for this, and are developing a set of tools that will enable comparisons between NetInf protocol implementations and network designs, but can also be used to compare different ICN approaches. The basic approach is to have a standard corpus [8] and to develop tooling and measurements to enable the above comparisons to be made and to be independently replicated. We plan to continue this work within the IRTF ICN research group. We will demonstrate and discuss this early-stage work.

References

[1] D. Kutscher, S. Farrell, and E. Davies. The NetInf Protocol. Internet-Draft draft-kutscher-icnrg-netinf-proto, Oct 2012. Work in progress.

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

[3] S. Farrell, A. Lynch, D. Kutscher and A. Lindgren, Bundle Protocol Query Extension Block, draft-irtf-dtnrg-bpq, IETF, May 2012. Work in progress.

[4] V. Cerf, et al. Delay-Tolerant Networking Architecture, RFC 4838, IETF, April 2007.

[6] http://sourceforge.net/projects/dtn

[6] FUSE Filesystem in Userspace: http://fuse.sourceforge.net/

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

[8] http://www.soschildrensvillages.org.uk/about-our-charity/archive/2008/10/2008-wikipedia-for-schools

GIN: A Global Information Network for NetInf

GIN (Global Information Network) is a hybrid architecture able to support both dissemination and conversational communication models. It uses a stateless packet-based forwarding protocol, called GINP (GIN Protocol). GIN aims to interconnect Named Data Objects (NDOs) over heterogeneous L3/L2 underlayers, in the global network, by means of an integrated name-based resolution and routing mechanism. Data are routed by names into the GIN network on a double path: the resolution path is used to route 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 the Multilevel Distributed Hash Tables (MDHT) architecture [5]. 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 Table (NHT).

Telecom Italia has developed a **proof-of-concept prototype** of a GIN infrastructure node. The current GIN demo prototype provides the following features and services:

- name based routing and forwarding over heterogeneous networks (IPv4, IPv6, ETH);
- integrated name resolution by means of a distributed MDHT dictionary;
- in-network registration and storage of named objects;
- multicast ping of named objects (called gping);
- end-to-end receiver-based multipath retrieval of named objects;
- support for search of data objects by keywords and names by means of a simple search engine.

The GIN node, developed on **FreeBSD**, consists of two subsystems: the **GIN Switch** and the **GIN Dictionary**.

The GIN Switch is implemented with a multithreaded program in C language. 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 enable GINP over Ethernet, IPv4 and IPv6; to add, delete, print GINP static routes; to shape GIN interfaces, to print or reset GIN interface counters; etc.

The GIN Dictionary is implemented in PHP language and runs over an Apache HTTP server providing proxy services. The GIN Dictionary is composed of a Dictionary DB, holding bindings for registered object IDs, and three main modules:

- the "Resolver" module handles resolutions for GINP packets and multicast GIN echo requests (called gpings).
- the "PUT" module implements the registration protocol (client and server side).
- the "GET" module implements reliable end-to-end multisource retrieval protocol (client and server side).

A network testbed of virtual GIN prototype nodes has been setup on a VMware platform. The testbed 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 underlayers (IPv4, IPv6 and Ethernet) and no IP connectivity is provided end-to-end. GIN Protocol (GINP) packets flow seamlessly over different underlayers. GINP provides the common network communication level. Client data objects can be stored in GIN access nodes, and near object copies can be retrieved and *gpinged* from the GIN access nodes, showing locality and anycast behavior.

Current prototype **software and documentation have been publicly released** and are available on GIN web site http://gin.ngnet.it . 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, 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).

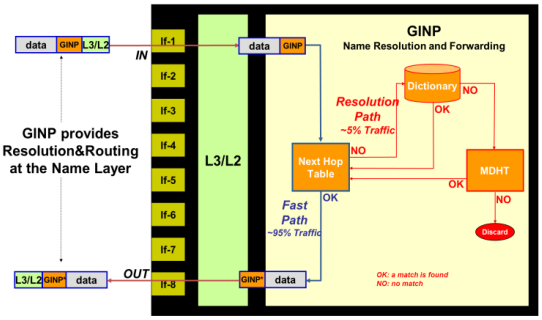
Contact: Matteo D'Ambrosio
matteo.dambrosio@telecomitalia.it

GIN: A Global Information Network for NetInf

THE GIN VIEW

- **GIN is a hybrid networking architecture for ICN**
  - GIN supports both host-centric and information-centric communications
- **GIN interconnects Information Objects, addressed by user-level names**
  - GIN supports any user-level naming scheme (Internet URIs, NetInf, DONA, CCN, etc.)
- **GIN implements a packet-based Name Networking Layer**
  - GIN forwards data by name in the global Internet, in packets over L2/L3 heterogeneous sublayers
  - GIN Protocol packets are encapsulated directly in the L3/L2 frames
- **GIN adopts end-to-end transport services**
  - Any nearby copy of a named Information Object can be used for direct download
  - Object copies are registered in a Network-distributed Dictionary
- **GIN integrates innovation as it appears in the overlays**
  - Services can be added to an open and flexible SDN network platform equipped with storage and processing

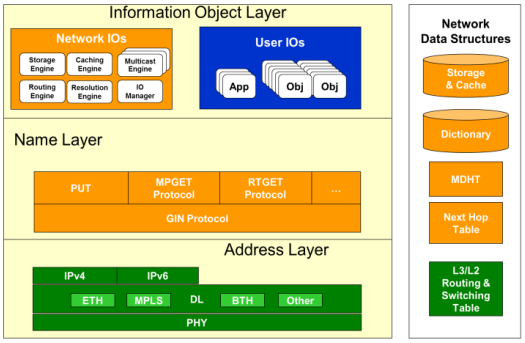
GIN NODE



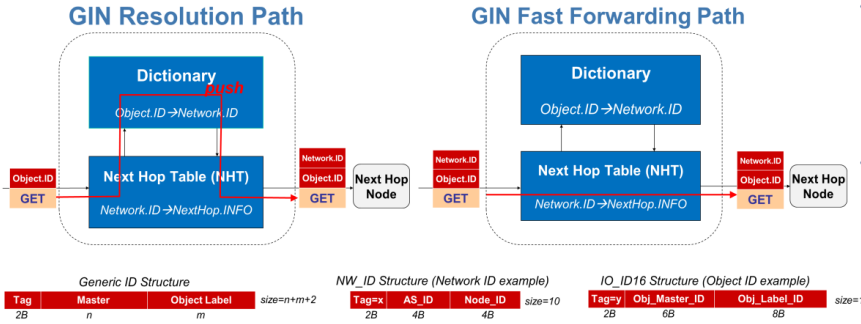
- A Control Plane provides local and global resolution and routing and support for network services (caching, storage, multicast, mobility, etc.)
- A Data Plane provides fast forwarding on shortest paths with ID switching

PROTOCOL STACK

- **GIN Architecture Overview**
  - At the core of the GIN network architecture is a Name-based Networking Layer
- **The foundation of the Name Layer is the GIN Protocol (GINP)**
  - GINP is the core protocol used to connect named objects over GIN
  - GINP is a stateless delivery protocol similar to IP
  - Information Objects (GINP endpoints) are identified by means of ID stacks
  - GINP packets are routed by object IDs though the network
  - GINP is the unifying name networking layer over heterogeneous L3 or L2 networks
- **PUT Protocol**
  - The PUT protocol is used as a signaling protocol between clients and nodes in order to register/update/delete binding information into the GIN Dictionary (i.e. the GIN distributed Name Resolution Database).
- **GET Protocol**
  - The GET protocols are request/response protocols, used by the clients to ask objects by name to the network
  - The MPGET (MultiPath GET) protocol is a multisource retrieval protocol



NAME RESOLUTION & FORWARDING



- **GIN Resolution Path**
  - In the Dictionary, Object IDs are mapped to network IDs
  - Object IDs are registered by clients with the GIN registration protocol (PUT)
  - MDHT (Multilevel DHT) maps ranges of Objetct Ids to next hop Dictionaries if no resolution is locally available
- **GIN Fast Path**
  - In the NHT, Network IDs are mapped to L3/L2 next hop information
  - Network IDs are advertised in traditional intra/inter-domain routing protocols (e.g. ISIS, BGP)
  - Most of the GIN traffic is routed over the Fast Path by means of ID switching

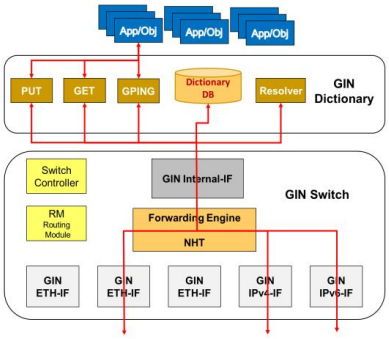
2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

SEVENTH FRAMEWORK PROGRAMME

GIN Prototype

A NetInf Proof-of-Concept of the Global Information Network

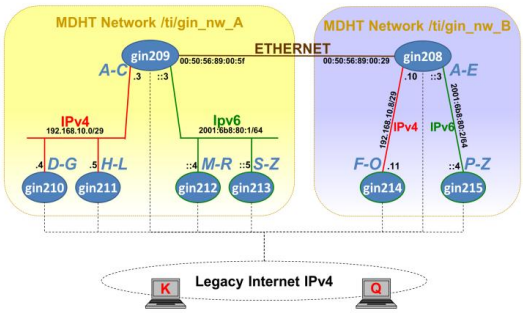
PROTOTYPE ARCHITECTURE



The GIN prototype is licensed as open software, for demo and testing purposes under Apache License 2.0, over FreeBSD 8.2.

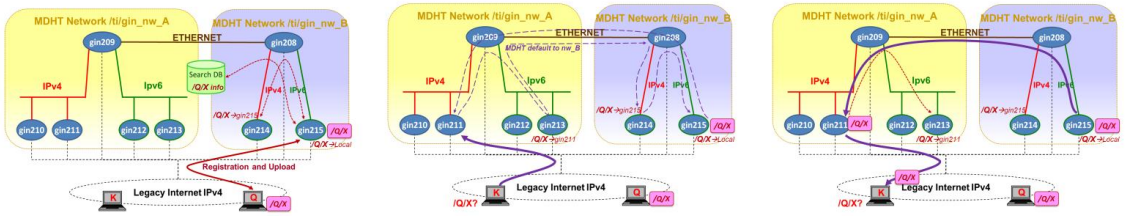
The Forwarding system has been written in C language; the Dictionary system has been developed in PHP language.

A TESTBED SETUP



An example GIN testbed has been set up: GIN nodes are interconnected by means of heterogeneous sublayers: IPv4, IPv6, Ethernet. Two double-level Multilevel DHT Domains (A and B) are statically configured and provide a distributed Name Resolution System in each network domain.

ICN SERVICES ON THE GIN TESTBED



Registration and Upload

- Client Q uploads object /Q/X to Access Node gin215
- Object /Q/X is registered in MDHT (locally and on another upper level node) and in the Search Engine DB

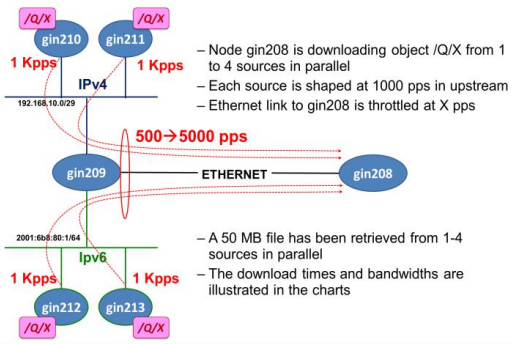
Resolution

- Client K sends a GET request for /Q/X to access node gin211
- Node gin211 sends a GPING request for /Q/X and receives a GPING Reply from gin215

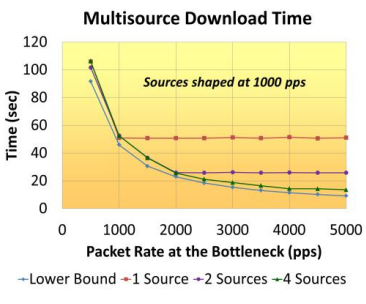
Retrieval

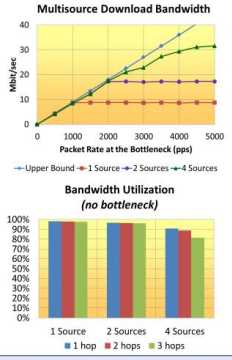
- Object /Q/X is downloaded from node gin215, cached on gin211 and registered in network A and delivered to the requester
- Following requests for the same object in network A will be satisfied locally

AN EXPERIMENT: MULTISOURCE RETRIEVAL PERFORMANCE GAIN



- Node gin208 is downloading object /Q/X from 1 to 4 sources in parallel
- Each source is shaped at 1000 pps in upstream
- Ethernet link to gin208 is throttled at X pps
- A 50 MB file has been retrieved from 1-4 sources in parallel
- The download times and bandwidths are illustrated in the charts





2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

SEVENTH FRAMEWORK PROGRAMME

NetInf
Open Source Software

SAIL has developed a rich set of prototype implementations of the NetInf protocol and corresponding applications. A significant fraction of these implementations have been released under Open Source Software licenses and are used by the ICN community for experiments and new research activities.

NetInf Software on SourceForge

The SAIL project has released an open-source (subject to the Apache v.2 license) set of tools for NetInf. These implement various aspects of the NetInf protocol in different languages. At the time of writing, there are C, Python, PHP: Ruby, Clojure and Java implementations with the Python, Puby and PHP code having seen the most development so far.

OpenNetInf

The OpenNetInf prototype [36] building on and extending earlier work from the 4WARD project.. The OpenNetInf implementation is a proof-of-concept implementation of the major NetInf elements, including the NetInf API, inter-NetInf-node interface, information model, naming concepts, security concepts, name resolution, caching, and data transfer. The goal is to evaluate the major NetInf design decisions and the overall NetInf architecture in practice. OpenNetInf contains a hierarchical name resolution system (MDHT-based) and ntegrated caching functionality. Another focus is on the NetInf API and on the inter-NetInf-node interface. The software contain browser plugins and video streaming software.

Global Information Network (GIN)

GIN is a hybrid ICN architecture able to support both dissemination and conversational communication models. It uses a stateless packet-based forwarding protocol, called GINP (GIN Protocol). GIN aims to interconnect Named Data Objects (NDOs) over heterogeneous L3/L2 underlayers, in the global network, by means of an integrated name-based resolution and routing mechanism.

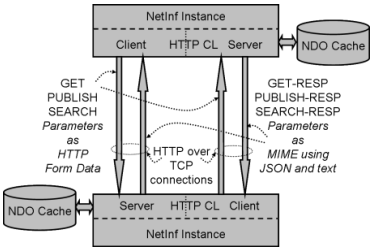
Android Client Software

SAIL has also developed an implementatio of the NetInf protocol for the Android mobile OS. The implementation supports publishing or registering Named Data Objects (NDOs) NDOs to NetInf infrastructure, sharing NDOs over Bluetooth, and obtaining NDOs over HTTP or Bluetooth Convergence Layers (CLs). The implementation reuses some components from OpenNetInf. It runs as an Android service and provides a local HTTP-based API to applications so that many applications can benefit from NetInf functionality,

Contact: Dirk Kutscher
Dirk.Kutscher@neclab.eu

NetInf
Open Source Software

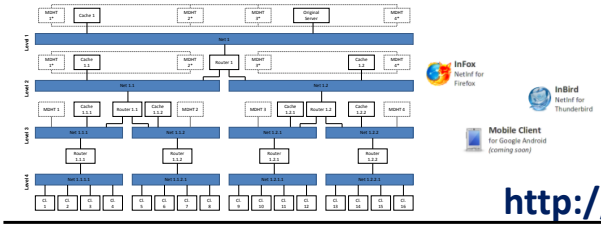
NetInf Software



- NetInf protocol stacks, routers, and applications
- Implements *Naming Things with Hashes*
  - draft-farrell-decade-ni
- Implements *NetInf protocol* and HTTP + UDP Convergence Layers
  - draft-kutscher-icnrg-netinf-protocol
- C, Clojure, Java, Python, Ruby

http://sourceforge.net/projects/netinf/

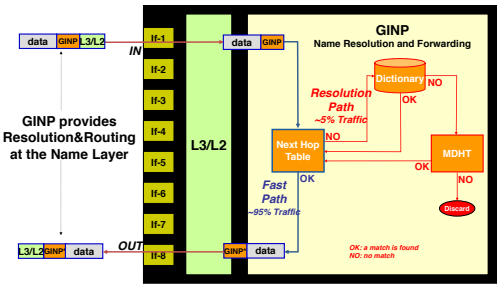
OpenNetInf



- NetInf Name Resolution
  - Multi-Level DHT (MDHT)
  - Hierachical SkipNet (HSkip)
- Applications and Browser-Plugins for web access and video streaming
  - InFox, InBird, Android client

http://code.google.com/p/opennetinf/

Global Information Network

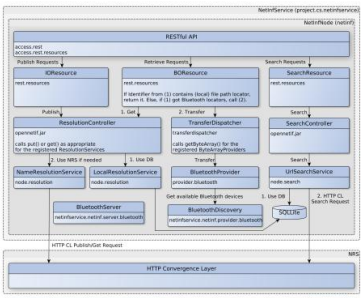


- Hybrid networking architecture for ICN
  - Host-centric and information-centric communications
- Interconnecting Information Objects, addressed by user-level names
  - Supporting any user-level naming scheme
- Packet-based Name Networking Layer
  - Forwarding data by name in the global Internet, in packets over L2/L3 heterogeneous sublayers

http://gin.ngnet.it

Android Client Software

- Implementation of NetInf protocol for Android mobile OS
- Publish or register NDOs with NiProxy
- Share NDOs over Bluetooth
- Get NDOs over HTTP or Bluetooth CL
- Reuses some components from OpenNetInf
- Runs as Android service and provides HTTP-based local API to applications
- Allows anyone to create a NetInf-enabled Android app



List of Abbreviations, Acronyms, and Definitions

ADU	Application Data Unit
AP	Access Point
BPQ	Bundle Protocol Query
BP	Bundle Protocol
BPA	Bundle Protocol Agent
BS	Base Station
CDN	Content Delivery Network
CL	Convergence Layer
DHT	Distributed Hash Table
DTN	Delay- and Disruption-Tolerant Networking
EwLC	Event with Large Crowds
FMD	Forward Meta Data
GIN	Global Information Network
GINP	Global Information Network Protocol
GW	Gateway
HTTP	Hypertext Transfer Protocol
ICN	Information-Centric Networking
IO	Information Object
IP	Internet Protocol
LFU	Least Frequently Used
LRR	Least Requested Rate
LRU	Least Recently Used
LXC	Linux Container
MDHT	Multilevel DHT
MN	Mobile Node

NDO	Named Data Object
NHT	Next Hop Table
NNRP	NEC NetInf Router Platform
NRS	Name Resolution System
NetInf	Network of Information
P2P	Peer-to-Peer
RND	Random
SAIL	Scalable and Adaptive Internet Solutions
SDN	Software Defined Networking
SSH	Secure SHell
URI	Uniform Resource Identifier
VoD	Video on Demand

List of Figures

Bibliography

[1] 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.

[2] 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.

[3] SAIL Project. Final NetInf architecture (D.B.3). Deliverable D-3.3, SAIL EU FP7 Project, 2013. FP7-ICT-2009-5-257448/D-3.3.

[4] 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.

[5] S. Farrell, D. Kutscher, C. Dannewitz, 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.

[6] C.Wong and S. Lam. Digital Signatures for Flows and Multicasts. In *IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 7, NO. 4,*, Toronto, Canada, 1999.

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

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

[9] 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.

[10] G. Carofiglio, M. Gallo, and L. Muscariello. Multipath congestion control in content-centric networks. In *IEEE INFOCOM NOMEN 2013 (to appear)*.

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

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

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

[14] 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.

[15] 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.

[16] 4WARD Consort. 4WARD – Architecture and design for the future Internet, 2008.