Subversion Over OpenNetInf and CCNx

Bengt Ahlgren
SICS
Email: bengta@sics.se

Börje Ohlman
Ericsson Research
Email: borje.ohlman@ericsson.com

Erik Axelsson
and Lars Brown
KTH
Email: {eaxe,lbrown}@kth.se

***Abstract*—We describe experiences and insights from adapting the Subversion version control system to use the network service of two information-centric networking (ICN) prototypes: OpenNetInf and CCNx. The evaluation is done using a local collaboration scenario, common in our own project work where a group of people meet and share documents through a Subversion repository.**

The measurements show a performance benefit already with two clients in some of the studied scenarios, despite being done on un-optimised research prototypes. The conclusion is that ICN clearly is beneficial also for non mass-distribution applications.

It was straightforward to adapt Subversion to fetch updated files from the repository using the ICN network service. The adaptation however neglected access control which will need a different approach in ICN than an authenticated SSL tunnel. Another insight from the experiments is that care needs to be taken when implementing the heavy ICN hash and signature calculations. In the prototypes, these are done serially, but we see an opportunity for parallelisation, making use of current multi-core processors.

I. Introduction

In the early days of the Internet the goal of most users was to explicitly establish connectivity between two nodes in the network and then run an application which would transfer some application-specific information. Today most Internet users are interested in getting access to a certain piece of information, not connecting to a specific node in the network, or using a particular application. The principal idea of *information-centric networking* (ICN) [3] is to create a networking architecture that puts information in focus instead of the interconnection of specific nodes. By focusing on the information objects themselves we can build a networking architecture that inherently can use a multitude of alternative ways to retrieve the desired information objects. These alternative ways include, but are not limited to, locally cached copies at nearby nodes, use of multiple access networks in parallel, information retrieval from pure broadcast networks like FM or TV broadcast networks as well as satellite networks. Also, networks with intermittent connectivity and data mule networks fit into this information-centric paradigm.

It is easy to argue that the ICN approach is ideal for large-scale information distribution. It is certainly true that the largest efficiency gain is expected for this type of application. In the work presented in this paper, we are instead investigating the benefit for another type of application which can take advantage of local ICN caching rather than always interacting with a remote server, making the application less dependent on good connectivity to that server. As the example application scenario we are using our own project meetings. A group of people in the same room or building is collaborating and sharing documents using a Subversion[1] version control repository. The repository is typically located in a different country and the available access network capacity is less than the demand.

We modified a Subversion server and client to use the communication service of the two ICN research prototypes OpenNetInf[2], from the 4WARD EU project, and CCNx[3], from the CCN project at PARC. We present experiences and insights from experimenting with the modified Subversion system, including evaluating the performance in the local collaboration scenario. A master thesis report [5] provides a more comprehensive description of the experiments. A shorter version of this paper appeared at a local workshop [4].

The contributions of this paper are the experience from the adaptation of Subversion, and the initial performance measurements on the resulting prototypes. The measurements show a performance benefit already with two clients for common cases, despite the overhead of the non-optimised prototypes. One experience is that it was very easy to adapt the file fetching parts of Subversion to ICN.

The rest of the paper is organised as follows. In the next section we provide a little background on the ICN approach to networking, followed by Section III describing the application scenario in more detail. Then follows a description of how we adapted Subversion to the ICN prototypes. We present the experimental measurement results in Section V. In Sections VI and VII we discuss difficulties with evaluating the ICN prototypes in a fair manner, and the benefits of ICN more generally. Related work is covered in Section VIII and, finally, Section IX concludes the paper.

II. Background

There are a number of initiatives in the research field of information-centric networking (ICN). They all have in common a focus on the information objects (IOs) as the key abstraction of the networking architecture, not the hosts as in the traditional Internet. Some key issues are how to name the IOs, how to resolve the IO names into locators that can be used for forwarding or, alternatively, how to do name based forwarding. In this background section we present two approaches to ICN, Networking of Information (NetInf)

[1] http://subversion.apache.org/
[2] http://www.netinf.org/
[3] http://www.ccnx.org/

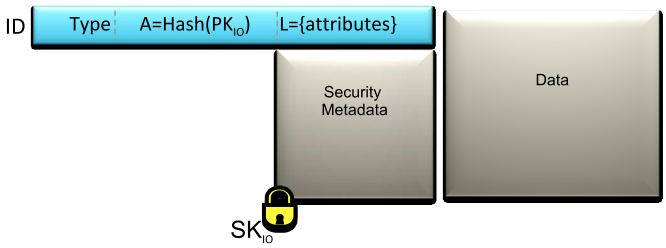

Fig. 1. NetInf IO with ID, security metadata and data (BO).

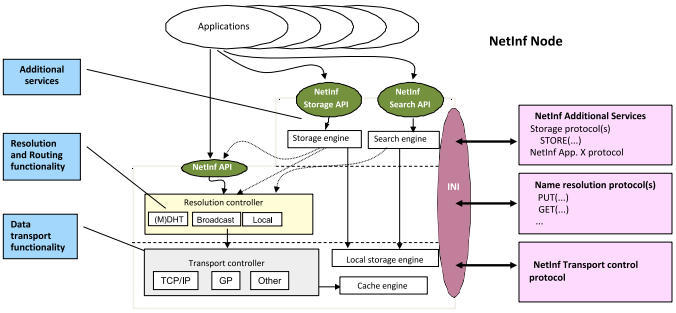

Fig. 2. The 4WARD NetInf node architecture.

and Content Centric Networking (CCN) which are used in the experiments presented later in this paper. For a more comprehensive overview of the field we refer to the paper *A survey of information-centric networking* [3].

A. NetInf

In NetInf [2] the names of the information objects, the IO IDs, are taken from a flat name space. The NetInf IO IDs are thus not arranged in any hierarchical structure. When a NetInf IO is published into the network it is registered with a Name Resolution Service (NRS) together with the network locators that can be used to retrieve copies of the published IO. When a receiver wants to retrieve an IO, the request for the IO is resolved by the NRS into a set of locators. These locators are then used to retrieve a copy of the object from the 'best' available source(s).

The NetInf information model has two basic objects:

- The NetInf Bit-level Object (BO) is the basic data object itself, or the digital representation of the object if the object is not a digital object (this is the case for real world objects for example). In other words, the BO stores the data information of the object. NetInf treats this as opaque data and does not derive any semantics from it.
- The NetInf Information Object (IO), illustrated in Figure 1 consists of three parts: the ID that gives the IO its name, the Metadata which contains semantic information associated with object, and the BO, as described above.

In addition to naming the IO, the ID has certain security properties that enable NetInf to verify the integrity of the IO (self-certification), and provide ownership information. The Metadata field consists of a set of attributes that provides semantic information about the IO. Metadata are also used for security purposes and for search.

The NetInf architecture (Figure 2) consists of a resolution service, a storage service, client applications and a number of NetInf additional services. The NetInf Name Resolution System (NRS) translates NetInf IDs into sets of locators. Depending on different requirements in different parts of the network it can be implemented in different ways. In parts of the network where local broadcast is available, a broadcast mechanism can be used. In other parts of the network, more elaborate mechanisms like Multiple DHTs (MDHT) [7] or Late Locator Construction (LLC) [9] can be used.

One central concept in the NetInf architecture is the concept of opportunistic caching of BOs to avoid unnecessary retransmissions of objects. The basic NetInf strategy for caching is to cache all BOs that reach the node; BOs are expunged when caches are full or their TTL expires.

Transport in NetInf deals with two kinds of messages. Firstly, control messages associated with NetInf services and Name Resolution that are communicated between the NetInf nodes and secondly, the messages that transport the BOs. The transmission of NetInf messages and BOs is performed by the underlying transport network with the support of transport protocols such as legacy TCP/UDP or future transport protocols custom-tailored to the needs of NetInf.

B. CCN

The main idea of CCN [11] is that a request, or *interest*, for an information object is routed towards the location in the network where that information object (IO) has been published. The routing information announcing CCN name prefixes, which is needed in the CCN nodes, are distributed in a similar way as routing prefixes are distributed in today's IP networks. At the nodes traversed on the way towards the source the caches of the nodes are checked for copies of the requested IO. As soon as an instance of the IO is found (a cached copy or the source IO) it is returned to the requester along the path the request came from. All the nodes along that path caches a copy of the IO in case they get more requests for it.

A CCN face is a generalization of a network interface. It describes a connection and how to communicate over this connection. Messages that are received and sent are always passed through a face. The operation of CCN nodes is similar to the operation of IP nodes: A packet arrives, a longest-match look-up is carried out on the name, and the corresponding action is performed.

CCN has a hierarchical naming scheme for information objects rooted in publisher prefixes, as illustrated in Figure 3. The hierarchical names makes it possible to request individual IOs or groups of IOs. It is also possible to algorithmically construct and request names for IOs that do not yet exist.

In CCN each node is equipped with three main components: a Content Store (buffer memory), a Pending Interest Table (PIT) and a Forwarding Information Base (FIB). These components are indexed in a way such that when receiving an Interest packet, a Content Store match will be preferred over

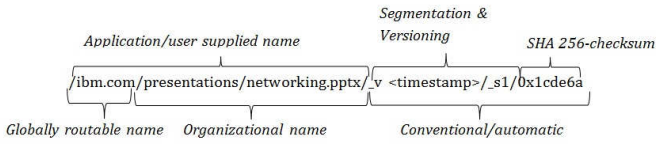

Fig. 3. CCN naming scheme.

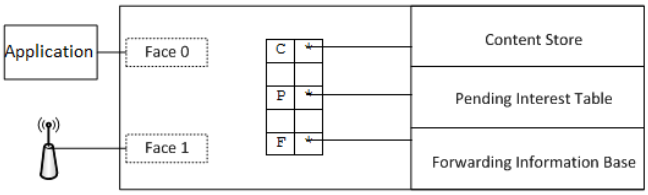

Fig. 4. CCN node architecture.

a PIT match and a PIT match will be preferred over a FIB match. Furthermore they all keep notion faces. These parts are shown in Figure 4.

The Content Store is a buffer memory where the nodes stores arriving ContentObjects in order to be able to share these. The ContentObjects are stored as long as possible with a LRU or LFU replacement policy. Hence if an Interest packet arrives and the corresponding ContentObject already is in the Content Store, the ContentObject will be sent out and the Interest will be satisfied and discarded. This enables each node in the network to provide caching functionality. To ensure the cache does not contain duplicates, ContentObjects will be discarded upon receipt if they match already existing entries in the store.

The Pending Interest Table (PIT) is a table which consists of Interest packets that the node has forwarded and not yet received any reply on. The PIT entries acts as 'bread crumbs', they leave a trail for the corresponding ContentObjects to follow. Therefore only the Interest packets need to be routed, since the corresponding ContentObjects will find their way back to the requester via the bread crumbs. When an Interest packet is received and there is no match in the Content Store but there is an exact match in the PIT, the face of the incoming packet will be recorded and added to the matching PIT entry, thus the interest can be satisfied by sending out the ContentObject on that face. This is exactly what is done when a matching ContentObject is received.

The Forwarding Information Base is last component, the FIB, is used to forward Interest packets towards possible data sources. If there is a FIB entry which matches an Interest packet then the Interest packet will be forwarded accordingly and a new entry will be created in the PIT. However if there is a FIB match for a ContentObject, it means that there is no PIT matching, hence there is no interest recorded for the ContentObject therefore it will be discarded. If there is no match, the Interest will simply be discarded by the receiving node since it does not know how to find the requested information.

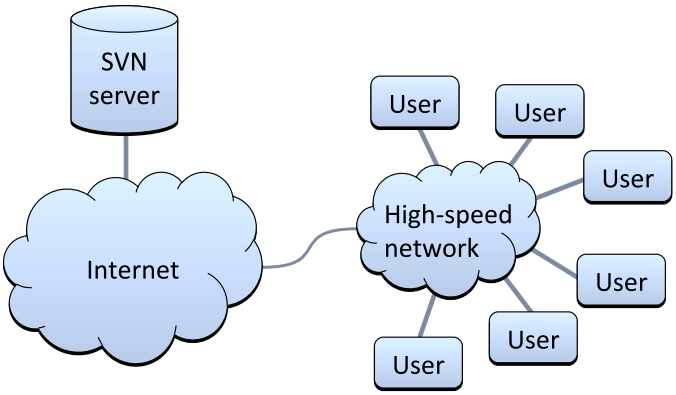

Fig. 5. Local collaboration using Subversion.

III. Local collaboration scenario

The application scenario addressed in this paper stems from our own work where a group of people is working together in a project. They find it convenient to use a collaboration environment that includes a common document repository (e.g., Subversion) and a Wiki. As is common, both these tools are located at remote network servers. The documents are mostly papers and presentations in Word, PowerPoint, OpenOffice and PDF file formats. The group often meets in a meeting room using a WiFi network connected via an uplink to the Internet as illustrated in Figure 5. They use these common resources to access the same repository. This results in multiple downloads of the same documents over the external link even though the documents are available from the computers that all are in the same room. The documents could instead thus be shared locally within the group in the room. Getting the same document from the same remote server multiple times is wasteful and often leads to performance problems, e.g., if the common uplink constitutes a bottleneck, which frequently is the case in our own work.

In an information centric network based on dissemination primitives, in contrast, the network would provide the users with access to the nearest/best copy of a requested information object rather than offering remote access only to a specific copy located at a specific host at a remote site. In ICN, a node in the room that already has a document stored can make that document available also to the other nodes in the room. The other nodes can use any type of access to retrieve the document, e.g., WiFi, Bluetooth or infrared. This has the additional benefit of making the document available also to those users that currently lack direct connectivity to a global transport network such as the Internet.

We are well aware of that this is not the typical use case for Subversion, for instance, papers and presentations are generally larger than source code and usually in some binary format that cannot benefit from sending the (small) difference between two versions. It is however what we actually use in several of our research projects. You can argue that the point is not really Subversion – any server-based file sharing tool would serve the purpose for the work in this paper.

IV. Adapting Subversion to ICN

In this section we briefly introduce Subversion and the ICN prototypes, and describe how we adapted Subversion to ICN. It should be noted that we, in our scenario, are using SVN as a document storage system for fairly large binary files, e.g. Word and PowerPoint documents. We are aware that this is defeating one of the key features of SVN, to be able to make small incremental updates to files. Nether the less, it is not an uncommon usage of SVN. Thus our results are only valid for this type of usage of SVN.

A. Subversion

The Subversion [6] version control system uses a centralised approach for sharing information. At the core of the system is the *repository*. It is a data storage which is structured in the same way as a file system tree, that is, a hierarchy of directories and files. Clients can add content to the repository so it becomes available to other clients, and clients can check out files from the repository in order to read them. The above description would fit to describe any regular file server. What makes a version control system different from a file server is that it keeps information about its own state and the changes to that state. This makes it possible for clients to request data from any given state, current or in the past. Providing this functionality in an efficient way is a prime objective of every version control system.

The basic client operations in Subversion are:

- **checkout:** Create a local copy of a certain version of a repository.
- **commit:** Upload changes in the local copy to the repository, creating a new version in the repository.
- **add:** Schedule to add a new local file or directory to the repository at the next commit.
- **update:** Update the local copy to the latest version in the repository.

B. Subversion over ICN implementation

This section describes the components used and the modifications made in order to adapt Subversion to run over the OpenNetInf and CCNx prototypes. In short, the Subversion commands which include downloading of information from the repository were adapted to be able to use OpenNetInf and CCNx for the content transfer. All other commands were left unmodified. Two Subversion protocol types, `icn+netinf://` and `icn+ccn://`, were added to be able to choose the ICN transports. Note that the ICN transports were unaware of versions. A new version of a file in Subversion is simply a different file.

1) Server side: SVNJ: SVNJ[4] is a Java-EE servlet implementing server-side access to the Subversion repository, similar to `mod_dav_svn` for the Apache web server. SVNJ was run in a Jetty web server.

SVNJ was modified in two ways. First, the data transfer commands, for example the above mentioned `checkout` and `update` commands, were modified to return the OpenNetInf or CCNx object names to the client instead of the files. Second, functionality was added to make the files available through OpenNetInf and CCNx. Which ICN that SVNJ should use is determined by the request it receives from the client. The non-data-transfer commands were handled by an unmodified Apache-based Subversion server.

2) Client side: SVNKit: SVNKit[5] is an open-source Java toolkit that implements all Subversion client functionality, including a command line client program.

We extended SVNKit to support file transfer with OpenNetInf and CCNx. The user decides which transport to use by specifying one of the new protocol types described above. For the `checkout` and `update` commands, SVNKit first communicates with the SVNJ server to get the OpenNetInf or CCNx object names for the the desired versions of the files, and then retrieves the files using OpenNetInf or CCNx. This means that the Subversion server always supplies the client with the object name list for the desired version, guaranteeing that the client can get the latest (HEAD) version. All other commands are sent to the Apache-based Subversion server.

C. OpenNetInf prototype

OpenNetInf uses a flat namespace for information objects [8]. The names have three fields: type, authenticator and label, where the authenticator is the hash of a public publisher key, and the label is chosen to be unique by the publisher, similar to names in DONA [13]. The names and locations of the information objects are registered in the NetInf name resolution service (NRS). To retrieve an object, a client first resolves its name using the NRS, and then retrieves a copy from one or more of the registered locations.

We implemented two extensions to the OpenNetInf prototype to support our scenario well. A parallel name resolution controller and a multicast-based resolution service have been implemented for the client side. The controller makes it possible to query multiple resolution services in parallel and return the answer from the fastest. The multicast-based service makes it possible to query nearby OpenNetInf nodes without any previous configuration or service discovery. When a node receives a multicast message the request is run through its local resolution service to see if the node itself has a copy of the object. If it is available, the node responds with an answer including itself in the list of locators.

The developed system uses NetInf names as follows. The name consists of three parts. The first part is the hash type where a SHA1 hash is used. The second part is the SHA1 hash of the Subversion server's public key and the last part is the repository and directory location of the file that this name belongs to.

D. CCNx prototype

CCNx uses hierarchical names, somewhat similar to URLs, that are rooted in publisher prefixes. A client request, or

[4] `http://code.google.com/p/svnj/`

[5] `http://svnkit.com/`

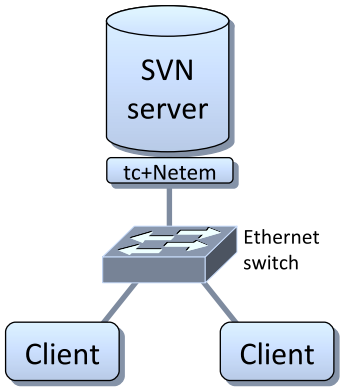

Fig. 6. Experimental setup.

interest, for an object is routed by the network towards the publisher using the object name. Each node on the path towards the publisher checks its cache for copies of the requested object. As soon as a match is found, the object is returned on the reverse path of the request. All the nodes along that path caches a copy of the object in case they get more requests for it.

We made no modifications to the CCNx prototype. CCNx names are used as follows. The Subversion URL for a file is directly mapped to a corresponding CCNx name with the revision number added as a suffix. Every client node had two remote routes in its routing table, one for multicast communication and one for communication with the Subversion server.

E. Security

We have neglected access control in the just described experimental adaptation of Subversion over ICN. A legacy Subversion system uses an authenticated SSL tunnel to the server which enables per-user access control checks by the server in a secure fashion. With ICN, where any node can deliver the desired data files, another approach is needed. For the server to be able to control access, the distributed data files need to be encrypted. Access control is then achieved by sharing the decryption key with the authorised clients.

V. Experimental results

In this section we present some measurements from the experiments we made with the two prototypes. The results show that the prototypes work as intended, most importantly, that local copies of data are retrieved automatically by the ICN network service without application involvement. The absolute performance numbers should however not be taken too seriously – these research prototypes are far from optimised.

The experimental setup is shown in Figure 6. The Subversion server and the two client nodes are connected to the same gigabit Ethernet switch. The WAN link to the server is simulated using the Linux kernel traffic control (tc) mechanism together with the Netem queuing discipline. Three settings were used: 100 Mbit/s with 5 ms delay, 5 Mbit/s with 50 ms delay, and 1 Mbit/s with 100 ms delay. For the second case, the commands were:

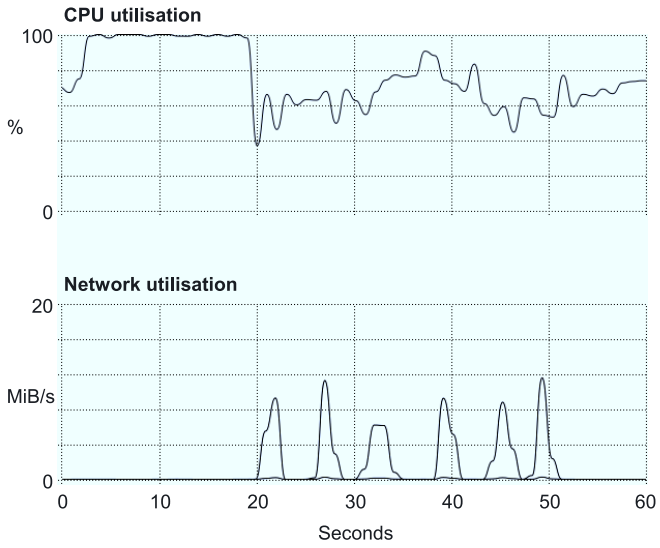

Fig. 7. OpenNetInf CPU and network utilisation.

```
tc qdisc add dev eth0 root handle 1: tbf rate 5mbit \
   buffer 16000 limit 3000
tc qdisc add dev eth0 parent 1: handle 10: netem \
   delay 50ms
```

All nodes are Lenovo Thinkpad X100e with 1.6 GHz AMD Athlon NEO MV-40 CPUs running the Ubuntu 10.10 Linux distribution. The Subversion repository contained six 10 MB large files, representing the fairly large documents in the studied scenario. We furthermore used CCNx version 0.3.0 and OpenNetInf from ca September 2010.

Figure 7 shows the CPU and network utilisation at the server when the first client checks a repository out using the OpenNetInf communication service and unlimited WAN link. During the first 20 seconds, the server makes the requested files available through the OpenNetInf service, resulting in close to 100% CPU utilisation. This includes calculation of cryptographic hashes and signatures of each file, and registration in the OpenNetInf name resolution system. This is a one-time cost per version of a file that in the prototype is taken at the first request, but should of course be done in advance. During seconds 20-50 six files are transferred with good throughput to the client with short delays in between each. These delays are mostly due to signature verification at the client. The graphs clearly show the serial nature of the implementation.

Similarly, Figure 8 shows the CPU and network utilisation for CCNx. We only see the first two files being transferred in the 60 second interval, because the transfer takes much longer compared to OpenNetInf due to lower speed, about 1.5-2 MiB/s, whereas OpenNetInf gets around 10 MiB/s. CCNx thus seems to have a higher overhead overall. More investigation is however needed to find the reason.

Table I shows the measured performance of a checkout operation using Subversion over HTTP (legacy), over OpenNetInf, and over CCNx for the three setting of the WAN link.

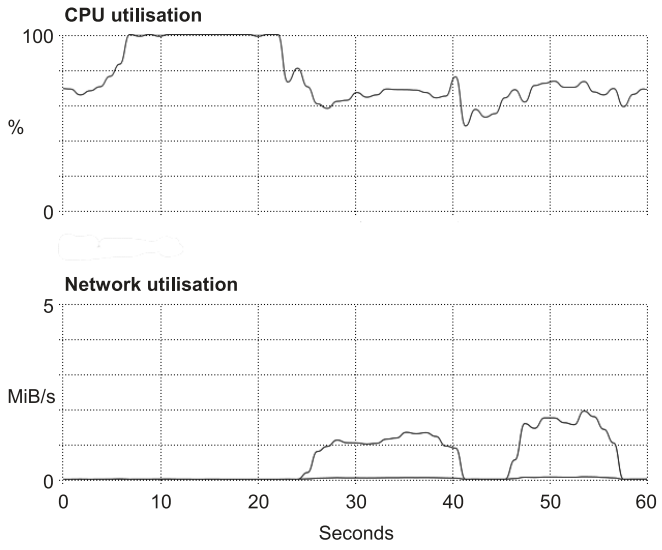

Fig. 8. CCNx CPU and network utilisation.

	Legacy		OpenNetInf		CCN	
	mean	stdev	mean	stdev	mean	stdev
100 Mbit/s, 5 ms delay						
1st client	19.8	0.53	49.6	0.81	103	1.14
2nd client	=	=	53.0	2.08	267	9.8
5 Mbit/s, 50 ms delay						
1st client	147	0.24	191	2.15	366	7.69
2nd client	=	=	53.8	1.11	262	3.72
1 Mbit/s, 100 ms delay						
1st client	726	0.45	770	2.61	871	1.54
2nd client	=	=	56.1	2.69	269	3.96

TABLE I
MEASURED PERFORMANCE OF CHECKOUT IN SECONDS.

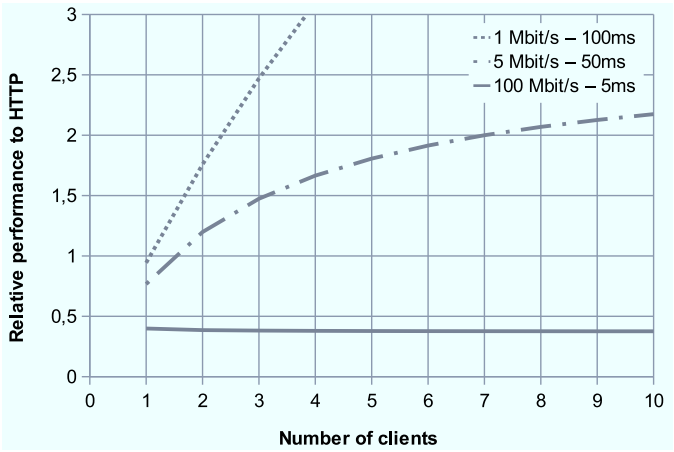

Fig. 9. Subversion/OpenNetInf performance relative Subversion/HTTP.

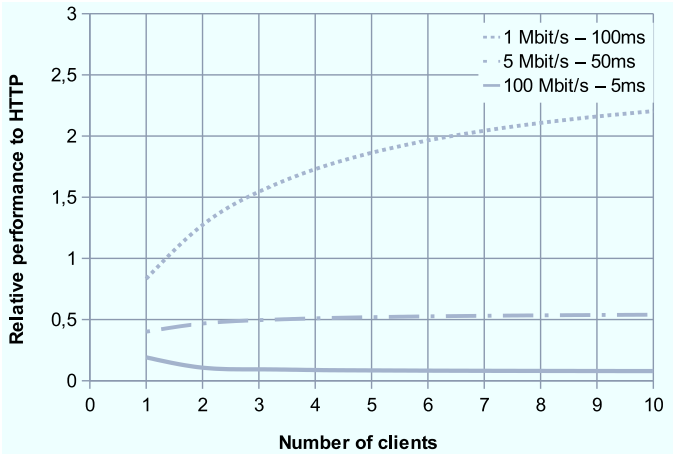

Fig. 10. Subversion/CCNx performance relative Subversion/HTTP.

The first client fetches from the server, and, for the two ICN prototypes, the second client fetches from the first client. The values are the means for five runs. As expected, the second client is in practise unaffected by the setting of the WAN link. Unexpected, however, is the low performance of the second client for CCNx, lower than the first client for the best WAN link setting. One reason is that requests are multicast resulting in that both the server and the other client replies.

The next two figures, 9 and 10, show the same data as Table I for Subversion over OpenNetInf, and over CCNx, respectively, relative to the performance of legacy Subversion over HTTP. The data for the third client and above are extrapolated as explained in the next section. Values below one (1) on the y-axis mean lower performance than legacy Subversion, and above mean higher. As the 100 Mbit/s 5 ms curves shows, there is no benefit to run SVN over ICN in a high (unlimited) bandwidth environment as the ICN overhead then is greater than the transmission savings. The three curves show the performance for different connection bit-rate and delay to the server, simulating different WAN characteristics for the path to the server.

For Subversion/OpenNetInf, there is a clear performance gain for the 1 and 5 Mbit/s cases already when there are two local clients. For Subversion/CCNx there is only a gain for the 1 Mbit/s case due to the higher base overhead of the prototype.

VI. Evaluation Issues

In this section we discuss some issues with the experimental evaluation. Especially the performance measurements are severely affected by a number of factors, both positively and negatively, making it difficult to draw clear general conclusions.

A. Extrapolation to many clients

The experimental setup only includes two clients. The measurements can thus only be done for the first and second client. The results for three and more clients are instead extrapolated using a simple formula for the time t it takes for n clients, where t_1 and t_2 are the measured values for the first and second client, respectively:

$$t_n = t_1 + (n-1) \times t_2 \quad (1)$$

This formula for instance neglects parallelism. For the legacy (non-ICN) case with a single server there is a limited amount of parallelism that can be exploited before the server or the communication path becomes saturated. For the ICN case, on the other hand, there is much more parallelism possible, since the more clients, the more sources that can provide the data. So the conclusion is that the formula is conservative, that is, is underestimating the expected performance of the ICN cases.

B. Signature and hash value calculation

A good part of the overhead of both ICN prototypes come from the calculation and verification of cryptographic hashes and signatures. We have some indication of that overhead from the CPU utilization measurements, but the exact extent and differences between OpenNetInf and CCNx remains to be investigated in more detail.

For both Subversion over ICN prototypes, the first client requesting a particular file from the repository pays the cost of entering that file in OpenNetInf and CCNx respectively, including signature calculation. In a real implementation, this should be done already when that file is entered into the repository.

There are more opportunities for increasing the performance. On the client side, the prototypes perform reception, hash calculation and signature verification serially for each file retrieved. The hashes can be calculated while receiving, in an integrated layer fashion [1], and the signatures computed in parallel with the reception of the next file, making good use of today's multi-core CPUs.

C. Automatic selection of 'best' source

As caching the same information in multiple locations in the network is one of the key ICN features, source selection becomes a critical issue. In ICN, source selection can either be done by the network or by the receiver. If done by the network, a strategy function is needed that can perform the selection procedure. Alternatively, the network can deliver a list of possible source alternatives, including characteristics that can be used for selection, that the receiver makes its choice(s) from.

In OpenNetInf, this translates to selecting a locator (IP address), and in CCNx it translates to routing the interest packet. As both prototypes only have rudimentary such functions at best, the experiments were made with fixed priorities between the known sources.

D. Comparing apples with oranges

We found that the experimental evaluation of ICN compared with legacy host-centric networking is difficult for several reasons. Some of the overhead that ICN introduces only applies to when objects are stored or accessed the first time. How shall that performance loss be taken into account in a fair way when comparing with traditional host centric networking?

The result is very dependent on the scenario and its network topology. The traffic pattern is different for the ICN and non-ICN case. For our local collaboration scenario, few clients and a well connected server means no gain with ICN. But with many clients and a badly connected server the gains can be huge. So it is hard to make general statements about the technologies as such, it all depends on what assumptions you make on how they will be used. It is often not possible to make a direct comparison – we end up comparing apples with oranges.

VII. Discussion of ICN benefits

A. Performance/efficiency benefits with ICN

Traditional client server applications are connected via a point-to-point connection over which downloads and/or transactions are performed serially. When an application is ICN enabled one thing that comes with it is parallelization. An ICN application is not making requests towards a specific server in the network but it is making its requests to the 'network'. One way to model these 'network requests' are as anycasts.

After that an ICN application has sent a series of requests to the network the application is out of control. To which extent requests are handled in parallel is determined by the name resolution and routing mechanisms in the network. ICN requests are about retrieving or connecting to information objects. From the application's perspective it is irrelevant if the requests are satisfied by one or many hosts/caches/routers.

B. Advantages with the information-centric security model

In ICN security is based on cryptographically binding the name of an information object to the very content of that information object. Once you have an ICN identifier that you know represents the information you want you can retrieve a copy of that object from any source without having any trust relation to that source.

This is very different from traditional host-centric networking where you need to trust the source host that is sending you the information object. In addition to trusting and verifying the authenticity of the source you need to secure the transmission channel between the source and the sink. This has several drawbacks. One is that it limits the number of possible sources to the set of hosts that you have built a trust relationship with. Another is that if someone, e.g., a trojan in the source host, have manipulated the information that the source host is sending to the sink it is difficult for the sink to detect this.

C. DTN/adhoc features

ICN has good potential for supporting disruption-tolerant networking (DTN) and adhoc scenarios, since caching of information objects is a basic ICN function. For DTN, local caches means that a disconnected part of a network can still function and deliver the data that are present in those caches (and servers) still available in the partitioned network. It is necessary that some local NRS is present in the partitioned part of the network. Simple broadcast resolution can be used in small partitions. For larger network partitions reconfiguration of NRSes and potentially election mechanisms are needed.

Adhoc scenarios are well supported as ICN hosts can support P2P delivery. The minimum network configuration is

two nodes. Each host can have a local NRS which it uses to keep track of locally cached copies as well for caching results of previous remote NRS requests.

D. ICN and multihoming

Hosts can have multiple interfaces (e.g., WiFi, Infrared, Bluetooth). Requests for information objects can be sent over all interfaces in parallel or in sequence in accordance with policies. How to make requests is a strategy decision that depends on user/network policies. ICN technology as such puts few limitations on how multiple interfaces can be used. Registrations of information objects and responses to NRS requests might differ for different objects as not all objects might be possible to deliver over all interfaces (e.g., live streaming HD video can not be supported by a low bitrate interface).

VIII. Related work

ICN is a new area of research, related work has been/is being done in EU projects 4WARD[6], SAIL[7], PSIRP[8], PURSUIT[9], COMET[10] and in the US projects DONA [13], CCN [11] and NDN[11]. A general overview of the ICN area initiatives is available in the ICN overview paper [3].

Some applications have already been developed that implements ICN architectures. On the NetInf side Extensions to Mozilla Firefox and Mozilla Thunderbird called InFox and InBird have been developed with the OpenNetInf platform as their base [2]. On the CCN side applications include a simple chat (CCNChat), a file proxy (CCNFileProxy) application and voice-over-ccn (VoCCN) [10]. The PSIRP/PURSUIT prototyping have primarily focused on evaluating performance of a pure publish/subscribe architecture [12].

To our knowledge this work is the first where the same application has been implemented on two ICN platforms and a comparative study between the two approaches has been made.

IX. Conclusions

We have adapted a Subversion server and client to use the network service of the ICN prototypes OpenNetInf and CCNx. Experiments were done in a local collaboration scenario to evaluate the benefit of ICN for applications that can take advantage of local caching to lessen the need for good connectivity to a server.

We described experiences and insights from the experiments. It was straightforward to adapt the file transfer parts of Subversion to use ICN. It is clear that ICN has a higher base overhead than current network protocols, but the experiments revealed that the gain quickly offsets the costs. The measurements show a performance advantage for ICN already with two clients for realistic simulations of the WAN connection to the server, despite the systems being non-optimised research prototypes. The experiments also revealed that there are good opportunities for optimising the prototypes by parallelising the computation of the hashes and signatures, lessening the impact of these overheads. Automatic selection of the 'best' source is crucial for good ICN performance, a function the prototypes currently lack.

While ICN can make communication more efficient for the same content distribution scenarios that motivate the use of CDNs, the local collaboration scenario of this paper and similar scenarios that approach disruption-tolerant networking (DTN) provide additional motivation for ICN.

[6] http://www.4ward-project.eu/
[7] http://www.sail-project.eu/
[8] http://www.psirp.org/
[9] http://www.fp7-pursuit.eu
[10] http://www.comet-project.org
[11] http://www.named-data.net/

References

[1] B. Ahlgren, M. Björkman, and P. Gunningberg, "The applicability of integrated layer processing," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 3, pp. 317–331, Apr. 1998. [Online]. Available: http://www.sics.se/ bengta/ilpapplic.ps.gz

[2] B. Ahlgren, M. D'Ambrosio, C. Dannewitz, A. Eriksson, J. Golić, B. Grönvall, D. Horne, A. Lindgren, O. Mämmelä, M. Marchisio, J. Mäkelä, S. Nechifor, B. Ohlman, S. Randriamasy, T. Rautio, E. Renault, P. Seittenranta, O. Strandberg, B. Tarnauca, V. Vercellone, and D. Zeghlache, "Netinf evaluation," 4WARD EU FP7 Project, Deliverable D-6.3, Jun. 2010, fP7-ICT-2007-1-216041-4WARD / D-6.3, http://www.4ward-project.eu/.

[3] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking (draft)," in *Information-Centric Networking*, ser. Dagstuhl Seminar Proceedings, no. 10492. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2011. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2011/2941

[4] B. Ahlgren, B. Ohlman, E. Axelsson, and L. Brown, "Experiments with subversion over opennetinf and ccnx," in *7th Swedish National Computer Networking Workshop (SNCNW)*, Linköping, Sweden, Jun. 13-14, 2011.

[5] L. Brown and E. Axelsson, "Use of information-centric networks in revision control systems," Master's thesis, Royal Institute of Technology (KTH), Jan. 2011.

[6] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion*. O'Reilly Media, 2004.

[7] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone, "MDHT: A hierarchical name resolution service for information-centric networks," in *ACM SIGCOMM Workshop on Information-Centric Networking (ICN-2011)*, Toronto, Canada, Aug. 19, 2011, in conjunction with ACM SIGCOMM 2011.

[8] C. Dannewitz, J. Golić, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *13th IEEE Global Internet Symposium*, San Diego, CA, USA, Mar. 19, 2010, in conjunction with IEEE Infocom 2010.

[9] A. Eriksson and B. Ohlman, "Scalable object-to-object communication over a dynamic global network," in *Future Network and MobileSummit*, Florence, Italy, Jun. 16-18, 2010.

[10] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "VoCCN: Voice-over content-centric networks," in *Re-Architecting the Internet (ReArch'09)*, Rome, Italy, Dec. 1, 2009, a CoNEXT 2009 workshop.

[11] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th ACM International Conference on emerging Networking EXperiments and Technologies (ACM CoNEXT 2009)*, Rome, Italy, Dec. 1-4, 2009.

[12] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line speed publish/subscribe inter-networking," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. Barcelona, Spain: ACM, Aug. 17-21, 2009, pp. 195–206. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1592568.1592592

[13] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proc. ACM SIGCOMM*, Kyoto, Japan, Aug. 27-31, 2007.