Towards a Global Implementation of Named Data Networking

Niels L.M. van Adrichem

Master of Science Thesis

TU Delft Delft University of Technology

Network Architectures and Services

PVM 2012-076

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science
Network Architectures and Services Group

Towards a Global Implementation of Named Data Networking

Niels L.M. van Adrichem
1217984

Committee members:

Supervisor: Prof. dr. ir. P.F.A. Van Mieghem
Mentor: Dr. ir. F.A. Kuipers
Member: Dr. J. Vrancken

August 30, 2012
M.Sc. Thesis No: PVM 2012-076

Abstract

The host-to-host IP model currently supporting the Internet does not suffice in supporting current-day content distribution in the form of content-sharing via peer-to-peer applications, real-time media streaming and social networks. Since the design of IP, the usage of the Internet has changed from a messaging and few-to-few information sharing system to a few-to-many content distribution system where many users request large amounts of overlapping information. Running a content distribution network over a host-to-host network appears to be very inefficient since every piece of content needs to travel the complete distribution-chain from generator to consumer every time it is requested. The result is that identical pieces of information will often redundantly travel the same links and routers.

Information Centric Networking tries to solve this problem by proposing route-by-name instead of route-by-address mechanisms. This enables networks to be optimized for content-distribution instead of connections and allows routers to cache often requested pieces of content in memory. In this thesis we will attempt to solve problems that arise at the introduction of a new globally routeable network, enabling clients and networks to be a full member (both consumer and generator) on a global Information Centric Network. The topics discussed vary from dynamic end-user configuration and generating globally unique names in order to share information on the Information Centric Network, via mapping techniques to decrease routing complexity, to proposing a transition mechanism that dynamically creates IP encapsulating tunnels between disconnected Information Centric Networks. *In short, we discuss a multitude of problems which need to be addressed in order to assist the global implementation of an Information Centric Network.*

Table of Contents

Chapter 1

Introduction

The Internet as we know it today is designed as a host-to-host network. Packets efficiently travel across the network from source to destination in a packet-switched manner using the Internet Protocol (IP). At each traversed node, the packet is forwarded to a node one step closer to its destination. Routing table entries function as road signs stating in which direction packets need to be forwarded. The foundations of this network were designed in the late 1960s and early 1970s and have helped the Internet increase to its current volume and size.

The design criteria for the Internet were derived from the types of global communication known at that time, which included telephony, postal mail, telegraphs and the occasional few-to-few file exchange between universities. In the mean time the Internet has become a common good available to billions of people [36]. The increase of users, applications and new insights following the introduction of a global network have resulted in a changed usage of the Internet. Instead of the original insights of a host-to-host messaging system the Internet is consumed as a few-to-many information distribution network where many people request lots of overlapping information from a finite set of information generators. One could say we all watch the same viral videos on YouTube, news items on popular websites and want to view high-quality real-time streamed digital television at home. Even when we watch personalized pages there is much overlap in the content we see. Messages on social networks such as Twitter, LinkedIn and Facebook are also seen by common connections, while other content - such as advertisements - may overlap on a different target group.

Whenever a client requests content, a connection is set up to the generating server and the content is requested over that connection. When a large number of users request content from a source, they all set up individual connections to that source and request information over it individually, disregarding the overlap in their requests. This results in many pieces of identical information traveling the paths surrounding the source. Compared to a retail store distribution network, the buyer does not only travel to the store itself but travels the complete distribution chain to the generator of the product disregarding how many people have already traveled that path buying the same product. Taking a music store as an example, the buyer of an album needs to travel to the local music store, travel from that local music store to the wholesaler, to the distributor, to the importer, to the harbour, to the factory to pick up its

medium and travel back. The next customer desiring the same album will need to travel the same path from its own local music store to the factory, disregarding overlapping parts of the paths and parties.

Although it is clear that such a system is inefficient as a content distribution network, IP does not offer architectural possibilities to streamline content distribution. Considering the fact that client applications actually request the network to set up these connections[1] instead of requesting it to deliver content, we deliberately disallow the network to streamline on anything different from connections. There is no way for a host-to-host network, without employing Deep Packet Inspection, to employ any type of heuristics to monitor and steer the flow of content instead of the flow of connections since we explicitly ask it to set up connections. This is where the philosophy of Information Centric Networking (ICN) comes into play.

The goal of ICN is to make networks more aware of their role as information distributor in order to enable them to streamline that function. In this thesis we will discuss Named Data Networking (NDN) [64] as proposed by the Palo Alto Research Center (Xerox PARC) in Jacobson et al. [39] and conduct research on the problems one meets when globally implementing such a network. *The primary goal of this thesis is to propose solutions to enable global implementation of Named Data Networking.*

Throughout this thesis we will talk about

- Information Centric Networking (ICN) when addressing the philosophy of such a network,
- Named Data Networking (NDN) as a reference to the insights of routing-by-name as proposed by the Palo Alto Research Center and
- the Content Centric Network Architecture (CCNx) [11] as an implementation of Named Data Networking.

1-1 Thesis structure

In chapter 2 we will first discuss the history of events that brought researchers to produce the philosophy of Information Centric Networking (ICN), followed by a preliminary research of the techniques behind Named Data Networking (NDN) and its implementation Content Centric Network Architecture (CCNx). We will summarize related work previously done in the areas of information discovery and dynamic configuration which have helped us solving the problems discussed in the following chapters.

In chapter 3 we will discuss our first proposal combining dynamic auto-configuration of clients on a CCNx enabled network and the automatic generation of globally unique names. Dynamic configuration of client devices is crucial for end-user acceptation of a new protocol supporting the Internet, whilst sharing information globally in NDN requires a globally unique name. We propose a solution solving both problems and enabling dynamic formation of small office and home office networks. By offering a configuration-free mechanism for users to profit from NDN, we expect a higher willingness by users to participate.

[1] And request the content over these connections.

While chapter 3 offers a solution to easily *access and share* content in a NDN, the dynamically generated names are location based and share no context with the information being shared. In chapter 4 we propose a system for mapping registered names, such as domain names, to the location based names enabling users to share data using names matching the context. Such a mapping system is also able to solve the route decision complexity introduced by the routing-by-name name principle. The address space is much larger than IP which creates problems in maintaining global routing tables. Using a mapping system one can split registered names from aggregated routeable names. Users can use registered names for an optimal experience, while routing can use location aggregated names to keep global routing tables small and the complexity of selecting the correct forwarding rules low.

Alike to the introduction of IPv6, not all routers can be upgraded at once. Therefore, NDN needs transition mechanisms to also function in networks that are not fully NDN compatible. In chapter 5 we propose a transition mechanism and algorithm that can dynamically set up IP encapsulated tunnels between NDN islands in order to connect previously unreachable nodes. Our proposal efficiently chooses a balance of NDN dense paths and tunnels based on underlying link and path costs to keep the overhead formed by crossing NDN incompatible paths to a minimum.

Chapter 2

Preliminary Research

This chapter we will start with a brief summary of the history and findings leading to the foundation of Information Centric Networking and name based forwarding. Secondly, we will discuss the Named Data Networking implementation of CCNx, which is short for Content Centric Networking Architecture, designed by the Palo Alto Research Center (Xerox PARC) [11]. The goal of this chapter is to give more insight in the philosophy and problems faced while changing a network into an Information Centric Network. Finally, we will discuss related work on information discovery, the NDN equivalent of topology discovery, dynamic host configuration and transition mechanisms.

2-1 History

In the past, different research initiatives have worked on the idea of a content or information centric Internet. One of the first initiatives towards content centric networking is the Stanford TRIAD project [59] started in 1999. TRIAD deploys an overlay network on top of IPv4 to deliver content based routing based on HTTP URL's to move quickly to a close replica of the information. The primary goal was to elevate IPv4 and TCP to a higher content aware level by inspecting the HTTP headers and make routing decisions based on that information. The implementation heavily relies on IP routing and TCP and therefore does not represent a network independent possible OSI-layer 3 ICN replacement. The driving arguments behind the project did not include the limited scalability of a host-to-host network serving content distribution networks, but were based on the problems faced by the outrun of IPv4 addresses. The website of the project even states that a successful implementation of the technique would be able to eliminate the need to migrate to IPv6 and claims to work with current IPv4 NAT standards.

In 2006 UC Berkeley introduced DONA, a Data-Oriented (and Beyond) Network Architecture [43], which continued on the work of the TRIAD project. The project's intended implementation clearly is a route-by-name mechanism to route requests to the closest copy of the intended content. Still, the primary goals were to replace DNS and solve name resolution

from a networking perspective. After the right content is found, the content itself travels to the requesting node using a one-to-one TCP/IP connection. Again, this makes the intended end result to be an overlay application instead of an Information Centric Networking Architecture that has the ability to work independently of IP. However, in the DONA project many important subjects such as naming conventions and security issues (mostly authenticity) are already mentioned and discussed which provides a good reference for true Information Centric Networking.

The NetInf project, short for Network of Information [54], started in 2008 at the University of Paderborn. Although the goals of this project are the same as NDN's or CCNx's, their primary focus appears to lie on data modeling, naming and finding instead of routing by name. The invented technique uses Distributed Hash Tables, as with the Chimera P2P search algorithm [27], to find data quickly. Their website gives thorough tools to experiment with the technique and even has a Firefox plug-in to show the user-experience. NetInf is an EU funded project on which was also worked in the 4WARD and SAIL projects.

The apparently most active research in ICN is the route-by-name Named Data Networking technique proposed by PARC - the Palo Alto Research Center - [24] initiated and supervised by Van Jacobson, who also is accredited a great role in designing the algorithm for TCP/IP congestion control [56]. In 2009 PARC published an open source Named Data Networking implementation called the Content Centric Networking Architecture (CCNx) [11]. NDN and CCNx is based on prior and ongoing research in the field of Information Centric Networking and routing-by-name. The architecture works with so-called opaque names, which describe certain content for the end-user or application, but have no direct meaning to the networking nodes. The name in a packet replaces the destination addresses in IP and routers only forward requests based on this name to a destination providing this information. The data itself travels exactly the same way back to its requester. Caches can, and ideally will, be held by any node based on the name of the content to decrease the load on the generators of the requested content. The project has already given thought to authentication and authorization by using established signing techniques.

On the project's website one can download already functioning prototypes of the architecture. A Firefox browser plugin is currently on the list of work that is still in progress [25], as is a Layer 2 adaptation layer to provide reassembly and reordering over networks that do not support these features themselves. The project is being funded by the NSF Future Internet Architecture program under the name of Named Data Networking [24].

Table 2-1 shows a comparison between the discussed proposals of Information Centric Networking. *Since the proposal of Named Data Networking and its implementation CCNx is the most extensive and has the most resemblances to OSI Layer 3 networking, many of our work is based on, or continues, their studies.* This is also the main reason why we have chosen this particular stream of Information Centric Networking to base our research on.

2-2 Basic techniques of Named Data Networking

In this section we will present how the Named Data Networking principle and the CCNx implementation currently work. The information presented in this section is mainly a combination of the higher-level presentation of NDN in [39], which we technically filled in by extensively

Project	Main focus	Routing / Information finding	Content Delivery	Transport protocol independent
TRIAD	Prevent IPv4 address outrun	HTTP URLs	TCP/IP	No
DONA	Replace DNS and offer data oriented routing	Name Based Forwarding	TCP/IP	No
NetInf	Offer Information Centric Networking	Chimera P2P Overlay	TCP/IP or other underlying transport mechanism	No
NDN	Offer Information Centric Networking	Name Based Forwarding	Follow reverse Interest path	Yes

Table 2-1: Summary of the differences between ICN proposals concerning their main focus, difference in path or information finding, content delivery and whether they can work independently of an underlying transport mechanism.

researching through the less representative CCNx Protocol Description [19] and documentation [21]. Many conclusions of operation have been found by investigating the CCNx software documentation [22][23] and carrying out numerous of trial-and-error experiments to verify the documentation and get a good grip on the philosophy of NDN. The experiments were done using our own CCNx experiment testbed which is also used for the experiments in section 3-1. To our knowledge, no such overview both describing high level abstractions and low level functional details yet exists. Many subjects are still under active research and development, which means they might change and thus, hopefully, improve over time.

In NDN, when a node is interested in certain content or information, it can send out an Interest packet containing a name describing the content. Each receiving node forwards the Interest in the direction of the node responsible for generating the information until the Interest reaches the generator, or it reaches a node which can serve the content from its cache. When an Interest reaches such a node, a ContentObject packet is created which travels *exactly* the same way back to the requester of the information and the originating Interest message is discarded as it has been satisfied by the found content. Figure 2-1 shows a possible path over which an Interest packet and satisfying ContentObject packet may travel. Whenever a node along the path already has a cached copy of satisfying content, due to a simultaneous previous request, it may discard the Interest and return a copy of the cached ContentObject.

The goal behind the caching mechanism is to decrease the load on content generating nodes - and the links leading towards those - invoked by serving redundantly requested data. A second request for earlier delivered data can be delivered from the first coinciding node on the paths from requester to generator. The principle of NDN is that one does not care who delivers the data, as long as it is authentic While in IP networks nodes and links may overload once content becomes very popular and requested often, such as a video going viral, in NDN more requests also mean more nodes will have a copy of the popular content in cache. The probability that a node near to you on the path to the content generator has a cached copy of the content increases by its popularity. Via the caching mechanism, copies of content are automatically distributed towards parts of the network where it is requested.

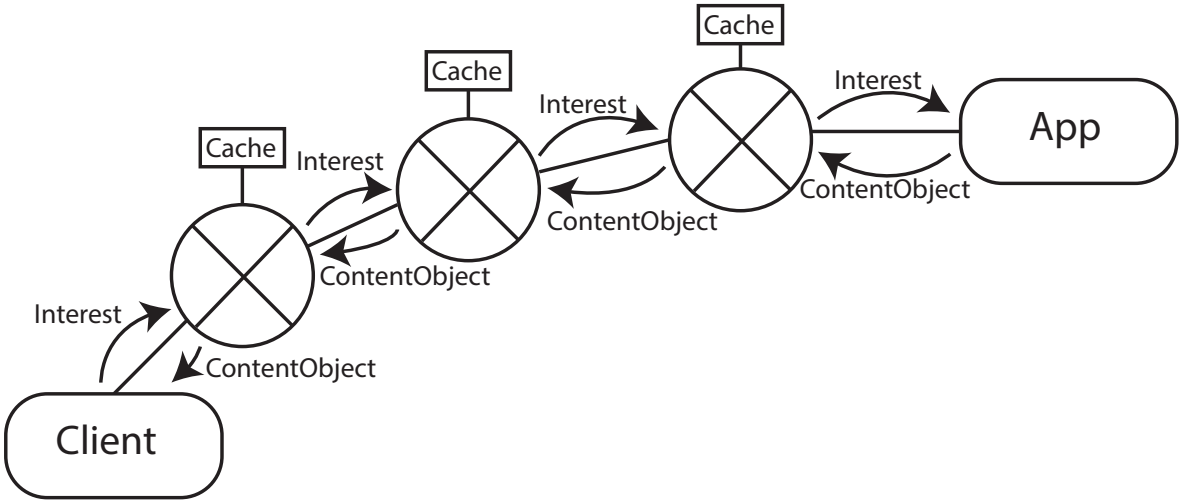


Figure 2-1: A graphical representation of an Interest requesting content traveling towards its generator. Any NDN enabled, by definition caching, node may return a cached copy satisfying the request, or forward the Interest further to the generating application.

The more content is requested in your area, the better its availability in caches near you. *One could say that the load on the network by a piece of content on average decreases when it is requested more and more often, instead of increased until the frequency of requests overloads the generating server and the paths leading to the server.* This is a very promising property for a network where a few organisations share, mainly identical, information to many.

As will be discussed in section 2-2-1, names generally take the form of `ccnx:/firstNameComponent/secondNameComponent/thirdComponent/etc/etc`. The description of the names do not have a meaning for the network itself: networking nodes merely select forwarding rules based on prefix matches on the name. The meaning of a name is only relevant for the parties requesting and generating information. Packets come in via interfaces which are called faces, due to the fact that these can be any type of connection with another process, daemon, node or link. A face can, among others, be an Ethernet-interface, an IP network, a radio link, fiber optics, a requesting or information serving application.

2-2-1 Forwarding and Routing

As is the case with all networking mechanisms, nodes or routers need a mechanism to decide how packets are forwarded across different networks. In a Content Centric Network, all nodes (routers, servers, consumers, peers) are equal in functionality, which means that they can all request but also serve, forward and cache data[1].

Every node keeps 3 tables. Based on its contents a node decides what action to perform on an incoming Interest or ContentObject. These 3 tables are:

[1]This feature actually eases the creation of Ad-Hoc Wireless Networks. Nodes can broadcast Interests to all nodes in range, who in turn can rebroadcast to even more nodes until a node satisfying the request is found. The contents of the PIT ensure the data will return to the original requester, while section 2-2-3 describes how loops in the Ad-Hoc Wireless Network are prevented. The subject of Ad-Hoc Wireless Networks however, exceeds the scope of this thesis.

- The Content Store (CS), which can cache any ContentObjects that have passed the node for a given period of time.
- The Pending Interest Table (PIT), which contains a list of Interests and incoming faces for which the node has already taken action but has not seen any response yet. The PIT entries serve as waypoints for a ContentObject to travel back through all the nodes and faces it has passed.
- The Forwarding Information Base (FIB), which contains forwarding rules for name prefixes.

The CCNx-daemon uses prefix matching of the different name components against the different tables to determine the right action it should take. Prefix matching strictly occurs on whole name components, e.g. a name of `ccnx:/alice/photo` would prefix match `ccnx:/alice` since the first name components match though it does not prefix match `ccnx:/alice/photos` since the second name components differ.

Whenever the Name of an incoming Interest *prefix matches* the name of a ContentObject in the CS, the node can discard the Interest and return a copy of the cached information. In case multiple ContentObjects match, a requested name may prefix match multiple names, at most 1 ContentObject satisfying the Interest is returned [17].

If the Pending Interest Table contains an *exact match* Interest, it means the node already has an outstanding request for the same information. The source face of the Interest is added to the outstanding list of interested faces, making sure when a satisfying response returns it is included in receiving that response. After the addition to the list the Interest is discarded as the appropriate forwarding actions were already taken.

If the Interest is still not satisfied, the Forwarding Information Base is consulted to see if any rules prefix match the name of the incoming Interest. The Interest is then forwarded to the faces denoted in the rules having a *longest prefix match* on the Interest's name. Since a face can be either a link, connection, tunnel or application it is possible for the Interest to be forwarded to a node one step closer to the generator, or to the application hosting or generating the content itself. When a node has multiple forwarding rules with a longest prefix match, Interests can be duplicated and sent to multiple faces. A rule to the Pending Interest Table is added denoting the name of the forwarded Interest and the source face it originates from. The chain of rules in the PITs on all nodes from the requester to the generator will be used for the returning ContentObject to be forwarded back to the requesters.

The process of determining how to process an incoming Interest based on its name by traversing the contents of the CS, PIT and FIB is shown in figure 2-2.

When an Interest hits an application face that is responsible for generating or hosting the information, the application creates a ContentObject containing a name that prefix matches the name of the Interest and describes the information generated. When the ContentObject enters a node, for any PIT entries whose (prefix) name matches the ContentObject, a copy of the ContentObject is sent to all its requesting faces and the entries are deleted as their requests have been satisfied. At any next node that receives a copy of the ContentObject this process continues until the chain to the requester(s) has been run down and the ContentObject is offered to the requesting application.

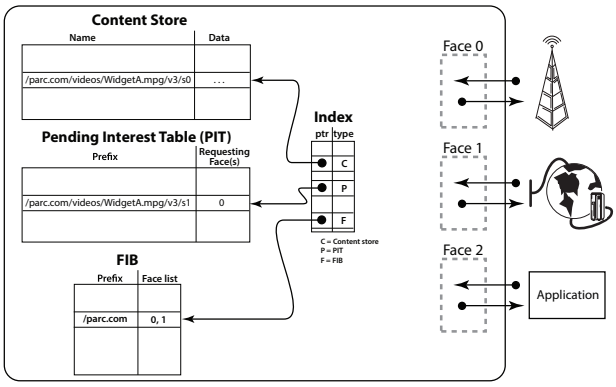


Figure 2-2: The CCN Forwarding Model, image has been taken from [39]

2-2-2 Naming

Since the routing and forwarding is done by name, we need a structured way of naming content. CCNx uses a hierarchical structure in which a name is composed out of multiple Name Components. Each subsequent component describes the characteristics of the content a little bit more precisely. As soon as the name of the Interest prefix matches the name of a ContentObject, that content qualifies for the request and can be returned to the requester. The prefix match-principle is valid in the sense that a longer name, prefix matched by your request, has a more precise description than you asked for but does satisfy your request. If we were to request pictures of animals, we could for example describe a grey horse as `/animal/horse/grey` (note the more precise description of the animal with each Name Component). In the previous case, our grey horse is a valid response to an Interest for an `/animal` or `/animal/horse` since it is in fact an animal and a horse. It is not until the user specifies a more precise description of, for example, a brown horse (`/animal/horse/brown`) that our grey horse response would not satisfy the request.

Prefix matching is also used in IPv4 and IPv6 [55][37] to select forwarding rules on routers, though the difference is that in IP matching happens on subsequent bits, whilst in NDN matching has to occur on subsequent complete tags. This means that `/animal` does not prefix match `/animals`. Therefore, bitwise, tags should be preceded stating the length of that tag. As a result of that, the forward slashes in the human readable names become obsolete in the binary encoded version of that name. Our content `/animal/horse/grey` could in that case be encoded as *6animal5horse4grey* where the numbers state the length of each tag. In this case the bit encoded name 6animal prefix matches, but the names *7animals* and *6animus* do not. A graphical representation of binary encoded name components is shown in figure 2-3.

The concluding name of information will be extended with extra components to denote versioning and segmentation (not fragmentation). For example, the name `/parc.com/videos/WidgetA.mpg/` is extended with, among others, the components `_s<segmentnumber>` and `_v<timestamp>` to denote the segment and version of the content being served. Since these components are added after the meaningful part of a prefix, these added components will not interfere with routing nor the communication between user and application. Due to the hierarchical structure of the name, the components can be interpreted as a tree, as shown in figure 2-4. In case of the versioning component, an Interest might add a preference to receive the most recent version by adding a property like RightmostChild to the Interest to ensure

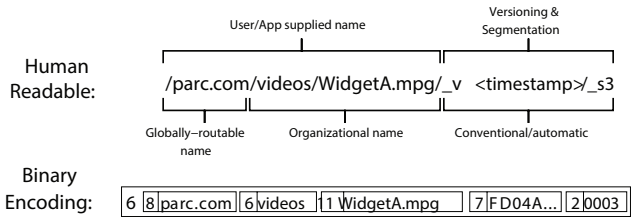


Figure 2-3: Example of a human readable and binary encoded version of a name, image taken from [39]

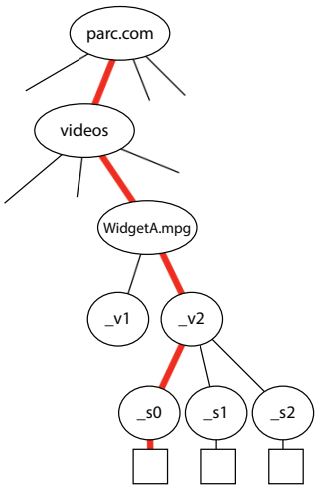


Figure 2-4: Name Tree Traversal, image taken from [39]

travel to the newest version of the information. In case the requester knows which specific version of the content it needs, it can add the appropriate Name Component `_v<timestamp>`.

CCNx basic name conventions

As discussed, CCNx uses a hierarchical naming structure in which each name component is divided by a forward slash for user readability [12]. Each name component adds a slightly stricter description of the content one needs or publishes. The CCNx basic name conventions [12] state a few guidelines for the usage of CCNx names. The meaning of a name is exclusively specified by the application and its users. Only in the case content should be globally accessible, global conventions should be followed. In local situations the use of any name is allowed and can therefore be decided upon by the organization or institution managing the network. The document does not give any further information about local conventions, even though local name conventions such as `ccnx:/organization.local` for intra-organizational networks might scale well.

The given guidelines state that for globally reachable content the first name component should be a “DNS name”. Choosing DNS names as the first name component could be useful while transitioning from IP and DNS to CCNx, since all domains have already been registered [39]. In addition, when a certain prefix is not accessible from a possible CCNx island - a set of nodes disconnected from a greater CCNx enabled network - one can use a dynamically set

up IP tunnel to a content server denoted in DNS. The most important guidelines for globally accessible content are that

1. the content name should be globally unique and
2. the content name should have a DNS name as first component.

ISP-based aggregation

One of the approaches proposed [64] for long-term routing scalability is ISP-based aggregation, which might be reused to dynamically generate names. The solution proposes a differentiation between user-selected names and provider-assigned names, where user-selected names are mapped to provider-assigned names using a mapping service analogous to DNS. The provider-assigned names are constructed hierarchically where longer names define a more precise location within a network. E.g. the content name `/att/atlanta/alice/blog` might aggregate to Alice's Internet connection in Atlanta connected by AT&T. For user-friendly access Alice might now register a shorter name `/alice` and have that name map to her provider-assigned name.

ISP-based aggregation uses the property that an entity responsible for delivering specifically named content can do so by forwarding Interests to nodes delivering even more precise specified content. In our proposal in chapter 3 we will reuse the property that an entity responsible for the primary part of a name can decide to forward Interests of subnames to other devices it is connected to.

2-2-3 Looping and TTL

In order to prevent Interest packets from looping, the originator of an Interest sends a randomly created nonce - a string of bits - along with the Interest packet. Coinciding nodes which forward a copy of the Interest packet keep a copy of the original Interest that was requested including the associated random nonce. As soon as an Interest with exactly the same name and nonce arrive on another face, the node knows that this Interest packet has made a loop and that the incoming Interest packet needs to be discarded.

All coinciding nodes keep an entry in their PIT containing names requested by the connected faces. Due to these entries, when an Interest packet hits a node who can serve the requested information the ContentObject packet travels back to the original requester using these entries. A ContentObject always follows the exact reverse order of the path the Interest traveled. Since Interest loops are already solved, and a ContentObject travels the same path as the Interest triggering its generation, ContentObjects also cannot loop.

In order to prevent packets to travel around the network for an infinite time, IP has implemented a hop limit denoting the maximum number of nodes an element may travel across before it is being dropped [55]. The hop limit is initially set by the sender, and decreased with a value of one by each node it passes until its value reaches zero. An initial value of 16 means that the IP packet can pass 16 links before the 64th node considers it *lost* and drops it.

Although the mechanism from IP is very transparent and comprehensible, CCNx uses a slightly different mechanism to avoid lost Interest packets. Every Interest packet has an

optional InterestLifeTime value which states in how many seconds, after receiving the packet, it should time out. When omitted a default value of 4 seconds is assumed. A node removes an Interest from its PIT entry as soon as it outstays its InterestLifeTime.

It is allowed (though strangely enough not recommended) for forwarding nodes to decrease the value of the InterestLifeTime to account for the time that the packet was in transmission and being processed by that specific node. If a node starts with an InterestLifeTime of 4 seconds and every forwarding node decreases the InterestLifeTime with 62.5 ms, the Interest packet is considered lost after 64 nodes. Hence, with the appropriate implementation, the InterestLifeTime mechanism can give a time out functionality which preventing lost Interest packets to travel across the network forever [17].

Note that a ContentObject travels the same way back as the Interest packet did, therefore it does not need any mechanism to prevent it from becoming lost in the network. Whenever a ContentObject can not travel any closer to the requester, which can happen when its requesting PIT entry times out, it is discarded.

2-2-4 Reliability and Flow control

In the case of audio or video streaming, or for that matter any data, another extension denoting the segment can be added. That means that a component _s0, _s1, _s2, etc. can be added to denote chunks of the stream. One of the advantages is that when the first segment has been received, the video player can already start playing while requesting the next. More importantly, it can also be used to replace the TCP acknowledgments and congestion control.

As soon as a node does not receive a ContentObject for which it has sent an Interest, for example due to packet loss, it can resend that Interest and it might receive that data this time since chances are lower for the packet to be lost twice. It is possible that the Interest does not need to travel down the complete way again; a subsequent node may already have the content in its cache due to the previous Interest and retransmit that information to the user.

If tags like NextSegment are added, or we calculate the next segment we need, we can request multiple segments up front so we can influence the speed with which ContentObjects will arrive. In comparison to TCP this adds the feature of a window size.

As you might imagine, the Interests combined with sequence numbers in the name can replace the functionality of ACKs and their sequence numbers as used in TCP. The algorithm used for congestion control in CCNx is still an active research topic, though one might imagine that it might become similar to TCP. Remember, the names are meaningless to the network itself. The names are only meaningful on an application layer or even to the end-user himself. The network itself considers the added version and segmentation components as simple parts of the name based on which a prefix match does, or does not occur.

2-2-5 Encapsulation

As is the case in any network protocol, many items need to be encapsulated in the packets being exchanged between nodes. In CCNx we consider two types of packets, the Interest packets requesting information and the ContentObject packets delivering the content.

Interest packets

The Interest packet can contain the following options [17]:

```
Interest ::= Name
             MinSuffixComponents?
             MaxSuffixComponents?
             PublisherPublicKeyDigest?
             Exclude?
             ChildSelector?
             AnswerOriginKind?
             Scope?
             InterestLifetime?
             Nonce?
```

In this format, the Name is the only component that is required. As discussed earlier, in an Interest packet the name is built up of hierarchically ordered Name components describing the content we are looking for. Each Name Component consists out of 0 or more bytes, possibly but not necessarily human readable, describing the content that is asked for. The other components can add extra requirements to which the data should match so the client can force extra constraints upon the requested content.

The 4 elements (*MinSuffixComponents*, *MaxSuffixComponents*, *PublisherPublicKeyDigest* and *Exclude*) add extra requirements to which a ContentObject should obey before it is matched. The *ChildSelector* tells us which element should be retrieved when multiple elements match, e.g. the latest or newest version in case of versioning, or the first or next segment in case of segmentation. The next 2 components (*AnswerOriginKind* and *Scope*) can give extra constraints concerning the publisher of the requested information. The last 2 elements *InterestLifetime* and *Nonce* are used by the forwarding nodes to determine routing loops and the time they should continue requesting the information.

Sometimes it is possible for a requester to already know how long the name of a matching ContentObject is, or in what range it should be. In this case it can state how many extra components a prefix matching ContentObject can or should have. The *MinSuffixComponents* and *MaxSuffixComponents* state this range, by default they should be equal to 0 and infinite in order to be omitted. When a client knows the exact name (including additional segment, version and signing components) of the information it requests, it can state so by setting the MaxSuffixComponents property to 0. A MaxSuffixComponents of 1, on the other hand, means that the name in the Interest is the complete name of the desired content without the final component giving the SHA-256 digest of the ContentObject, which means the requester initially accepts any content matching the name but has no preference or knowledge of any specific versions it might prefer. We will discuss more about signing and authenticity in section 2-2-8.

A ContentObject will always contain a SHA-256 (a cryptographic hash function) digest of the information it carries, which is signed using the public key of the generator. In case a client wishes information that is published by a specific publisher, it can add the digest of the publisher its Public Key to the Interest in the form of the *PublisherPublicKeyDigest*. As soon

as the PublisherPublicKeyDigest has been added, only content published by that publisher will be prefix matched against the requested Name.

The *Exclude* parameter adds an option to specify name components to which the first next Name Component may not match. For example, in case we are looking for a horse (`/animal/horse`) but we do not wish to receive a picture of a grey horse, we can add a component *grey* to the exclude list. In this case the added component *grey* will prevent grey horses to match our request. An Interest can contain multiple Name Components that it wants to exclude, for implementation reasons these components should be denoted in alphabetically increasing order[2].

Instead of name components, it is also possible to add filters matching multiple name components a client does not wish to receive. This is done in the format of so called Bloom filters[3] which for now exceed the scope of this text.

Since the name of a ContentObject is built up hierarchically, it can be graphically or logically referred to as a tree of nodes. As shown in figure 2-4, in this tree every name component matches a node to which child nodes are attached. A name matches information as soon as there is a prefix match between the name in the Interest and the name of the ContentObject. Therefore, it is possible that multiple child nodes match the Interest name. The *ChildSelector* enables us to express a preference for one node over the other. At the moment it is possible to give preference to the leftmost child (denoted by a 0), or the rightmost child (denoted by a 1). The preference is only taken into account for the first component after the prefix, using the ChildSelector it is possible to select for example the first segment of a movie, or the newest version of a certain segment. The child components are first ordered by length, which means that a shorter component string is considered alphabetically smaller no matter the content of the string, and (if necessary) later ordered per character [10]. This is a result of the binary encoding of name components as described in section 2-2-2.

The *AnswerOriginKind* element contains a bitmask describing how the information may be fetched or generated.

- The first bit tells whether the information may be delivered from a caching ContentStore, which is the default behavior.
- The second bit tells whether the information may be generated when it does not exist yet, this is also the default behavior.
- When the third bit is set, this means that the client accepts old or stale information to be served, it is not necessarily interested in the most recent information and nodes may serve cached copies that are past their due date.
- The fourth bit is currently not used.
- The fifth bit can be used to mark information already in the ContentStore to become stale (only to be used in conjunction with a Scope of 0). This option can be used by

[2] A value in a list can be found with a work complexity of $O(log(n)$ instead of $O(n)$ if that list is ordered.

[3] `http://www.ccnx.org/wp-content/uploads/2011/08/poster-ns3-ndn.pdf` states that the support for Bloom filters is deprecated; authenticity of this source has however not been confirmed as it is part of a mailing list and the authors connection with NDN is not trivial.

applications to tell caching nodes that the information they possibly contain has become outdated. Although the current mechanism is considered to be a hack, the functionality of being able to tell geographically spread caches that information became out of date seems legitimate in a distribution network.

The property *Scope* defines whether it is allowed for the Interest to propagate across the network or not.

- When it is valued 0, the Interest may not be propagated at all which means that only information already in the ContentStore of the node may be returned.
- The value 1 means that the receiving node can only forward the Interest to directly connected application faces, not to other types of faces. This behaviour is exploited when building up caches of a nearby application. A caching node can, for example, be connected directly to a forwarding node connected to a content generating application. The caching can request all items from the application through iteration and cache them in advance to speed up content delivery.
- A scope of 2 means that the Interest can be forwarded to any face connected to the node, which is the default behaviour. This means that it will propagate into the network according to all longest prefix-matching FIB rules searching for the information requested.

The *InterestLifeTime* property defines when, measured in seconds after it is received at the forwarding node, the Interest packet should be discarded and removed from the PIT by that node. It is also possible to use this value as hop limit or Time-to-live mechanism when all forwarding nodes decrease the property with a small amount. This small amount accounts for the time the packet is in transmission and being processed by the nodes. In combination with the segment numbering discussed in section 2-2-4 and the research suggested in section 2-2-7 about possible strategies NDN nodes can deploy to forward Interests this option can help in creating mechanisms concerning Quality of Service, reliability, flow and congestion control.

As previously discussed, the *Nonce* is a randomly generated value used to detect and discard looping Interest packets. The nonce is generated once by the original requester of the information and stays the same while propagating across the nodes to the content deliverer. When a node receives an Interest on another interface whose Name and Nonce are equal to an Interest received earlier, the node knows that this Interest packet made a loop and needs to be disregarded.

ContentObject packets

ContentObjects, the messages containing the data to be delivered, contain the following properties [14]:

```
ContentObject ::= Signature
                  Name
                  SignedInfo
                  Content
```

The first element, the *Signature*, is a signature of the Name, the SignedInfo and the Content with their respective XML binary encoded start and end tags (more about this in section 2-2-6) as they appear on *the wire* in the underlying layer. The Signature is used to identify if the ContentObject delivered truly is the message as generated by the signer, it confirms whether the content is authentic and has not been damaged or corrupted (both by accident or by security infringements) on the way to the receiver. Different signing methods can be used to verify the ContentObject, we will discuss this in section 2-2-8 about Authenticity and Security, for now we can assume a SHA-256 digest [20].

The *Name* matches the same prerequisites as the name in the Interest Packet. The only thing that differs is that the last element of the name will contain an implicit SHA-256 digest of the attached Content in order to make the name of that particular information globally unique. This helps in a) linking to exactly one specific ContentObject and b) excluding specific ContentObjects using the Exclude property of the Interest message.

The *SignedInfo* element has many elements containing detailed information about the content being sent. It contains the *PublicPublisherKeyDigest*, a SHA-256 digest of the generator its public key, which may be used by an application to quickly select the right public key from its own memory to decode or check the signature of the information. It also contains a TimeStamp, though it seems unclear whether it contains the time at which the ContentObject was either created, signed or sent.

The previous two elements PublicPublisherKeyDigest and TimeStamp are always attached to the SignedInfo element; the next elements are optional and can be used to denote extra information.

The *Type* element defines the type of content carried by the ContentObject. Possible values are plain data, encrypted content, a public key, a link or a NACK stating that the generator does not have nor can it generate content for the requested Name.

The *FreshnessSeconds* element states when a ContentObject should become stale. A stale object contains possible old information and should be eliminated from the caching ContentStore within a short time to eliminate Content from the network that is not recent anymore. This element states how many seconds after receiving a ContentObject it should expire. If the element is not specified, the packet does not become stale by time. It may become stale or removed by other mechanisms, such as a cache replacement policy.

The optional element *FinalBlockId* defines which segment, if applicable, is the last segment. Knowing the last segment particularly plays a role when content - such as streaming content or other types of media - is cut up in multiple segments. This element can be used to predict the end of the content being retrieved. The element does not need to be sent along with every segment of certain data, but it may be preferable to send it with the last number of segments to let the client know the end of the data stream is coming.

The optional *KeyLocator* tells where a client may get the Public Key to verify the signing of the object and is complementary to the PublicPublisherKeyDigest. Since a public key itself can be considered as a form of information, the public key can be requested via the NDN in the same way any other content is requested. The request of a public key is safe since the public key (which is the content in this matter) will again be signed by its publisher or certificate authority giving opportunities to exploit a PKI like certificate infrastructure.

At last the *Content* itself is being served containing the, possibly 0 bytes long, requested information encapsulated as described by the Type element.

We can conclude from the previous two subsections concerning the encapsulation of Interest and ContentObject messages that NDN has initial support for many possible implementations concerning caching policies, flow control mechanisms, segmentation and cryptographic authentication and encryption. Most subjects, however, still need further research in order to select or design the right mechanisms for a trustworthy global implementation. As a result, many of the discussed options still have open ends that need implementation specific finishing.

2-2-6 XML Representation and Encoding

CCNx uses an XML representation in order to describe the content of a packet. This makes it possible to use a XML Schema Definition (XSD) to describe what a CCNx packet should look like. An XML Document Type Definition (DTD) can be used to further identify the properties of a packet. CCNx has two of those schemes [18][16] which describe the properties and their structure being used in the messages and can be used to verify the structure of a message.

While XML is a great way of structuring the syntax and format of information, using plaintext human readable XML to transfer packets will give a large. Therefore, CCNx uses a custom designed binary encoding - called *ccnb* - instead of the human readable XML representation to transfer the information. The binary encoding has been designed in such a way that binary to XML to binary conversions are processed in such a way that the input and output match exactly.

The ccnb encoding mechanism is not application specific and can be implemented by other applications in which XML needs to be transferred compactly. The mechanism employs an application specific dictionary [15] to translate common tag and attribute names to shorter numbers in order to save space in the encoded message.

Each XML encoded element starts with a number representing a tag from the dictionary. The number - for which a more detailed example is shown in figure 2-5 - is a big-endian number representation[4], in which:

- The first bit of a byte equals 1 when it is the last byte from the number or 0 if it is not.
- For all preceding bytes, the supplementary 7 bits are part of the number.
- For the last byte, bit 6 to 3 represent the most significant part of the number.
- Finally, bit 2 to 0 are used to represent the type of the current and following elements.
  - 0x1: identifies that the following amount of bytes are an UTF-8 string representation of an XML tag name.

[4] A big-endian number representation describes numbers that are described using multiple bytes in significant decreasing order. For example, in a big-endian number representation the first byte represents the biggest, most significant part while the last byte represents the smallest, least significant part (which usually describes 1 to 255). Big-endianness is similar to the numeral style of writing decimal numbers.

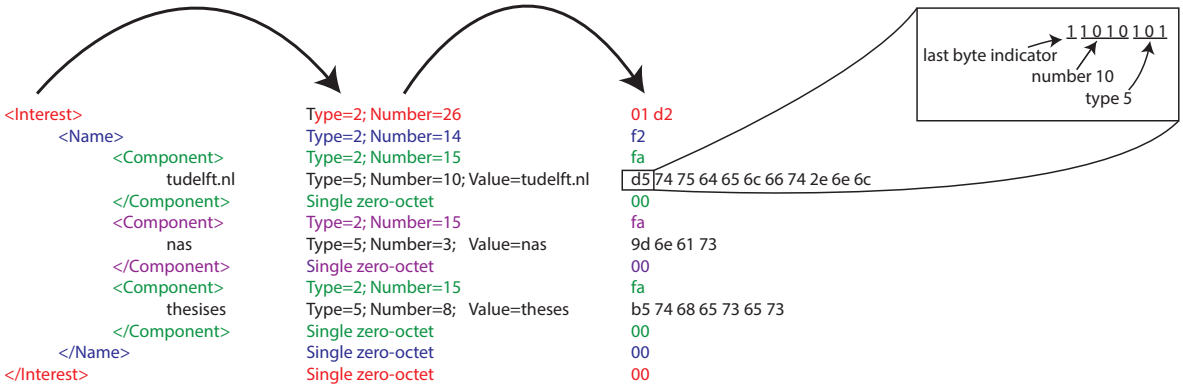


Figure 2-5: A typical ccnb-encoded Interest message for `ccnx:/tudelft.nl/nas/theses`, the box shows a typical one-byte encoding of the type and the number.

- 0x2: identifies the following number as a numerical representation of an XML tag from the dictionary. Both 0x1 and 0x2 can be followed by any type of element since XML nesting can occur.
- 0x3: idem as 0x1, though now it describes an attribute name which belongs to the previous XML tag.
- 0x4: idem as 0x2, though now it selects a name from the dictionary describing an attribute name. Both 0x3 and 0x4 must be followed by 0x6 containing the UTF-8 encoded string of data describing the value of the attribute.
- 0x5: identifies that the current number represents the length in bytes of the following data.
- 0x6: identifies that the current number represents the length of the following UTF-8 encoded information.
- 0x0: can be used for, currently non-existent, application specific extensions not coverable by the other types of information.

Basically, the last 3 bits of the big-endian number representation identify the usage of the number, and the information following the number.

Since the original XML structure has already been checked, each element is simply closed by placing a single zero-octet (0x00) avoiding the overhead of repeating each tag name when closing it.

Based on the CCNx specific dictionary [15], an Interest will be encoded as shown in figure 2-5 [19][13].

2-2-7 Strategy

One of the new options proposed in Named Data Networking is the *strategy layer* [39]. The philosophy behind the strategy layer is that a process should monitor all objects within a router and change forwarding decisions based on the events it registers. The objects to be monitored do not have to be limited to buffer usage, CPU usage and bandwidth usage (as

is usually the case with current-day QoS) but can also include average round trip times and information hits since incoming ContentObjects can be easily mapped to previously sent Interests without applying costly deep packet inspections.

NDN nodes should apply a strategy towards forwarding Interests for efficient retrieval of ContentObjects. Together with an appropriate cache replacement policy and flow control mechanism (as discussed in section 2-2-4), a proper strategy can help fine-tune or decide upon QoS parameter settings to optimize the network.

Since Interests can be duplicated and forwarded over multiple outgoing links (in case multiple longest prefix-matching FIB rules exist), a possible strategy can be to monitor which incoming links receive relating ContentObjects the fastest and to vary the amount of Interests sent to each link to the timeliness they answer with appropriate ContentObjects. If there is a link with a long route to the information and another link with a short route to the information, the strategy algorithm can decide based on the average response time (if any at all) to prefer sending Interests to the shorter and thus faster link. The algorithm can continue sending a smaller amount of Interests to the other link in order to verify existence of the path [39].

CCNx currently handles the following Strategy for all nodes. Incoming Interests are forwarded to all faces for which a longest-prefix match occur in the Forwarding Interest Base. A node will periodically retransmit Interests from active entries in the Pending Interest Table. It is recommended to retransmit at random times on different faces or use another sort of heuristic to schedule retransmission. Interests in the Pending Interest Base will timeout at a given time, if they are not timely refreshed or reactived by a new or updated Interest from their requester. This ensures that Interests for non-existent content are removed from the network over time [19].

The topology discovery mechanisms used for IP are responsible for both finding possible paths and choosing the one right (possibly most efficient) path from those paths. Forwarding an IP packet boils down to forwarding it to the single best path chosen by the topology discovery mechanism, no further intelligence is required. The strategy layer allows the forwarding daemon to employ heuristics to try out different paths found by the information discovery mechanisms and choose paths from this experience. Part of the responsibility of topology discovery, deciding on the best out of possible paths, moves to the forwarding daemon and can be further optimized by the experienced service. While propagated path costs can still play a role for the initial distribution of outgoing Interests, the strategy layer may decide to keep or change that distribution. We can conclude that the Strategy layer, combined with the fact that a returning ContentObject can be mapped to its outstanding Interest, enables the forwarding daemon to mean more to the network in forms of path selection and Quality of Service than plain forwarding of unrelated IP packets currently do[38].

Theoretically it is even possible for nodes residing in ad-hoc (wireless) configuration-less networks to gain connectivity using a specifically designed Strategy layer. If every node broadcast Interests to all devices within reach and start preferring the nodes that answer more often, while maintaining the occasional broadcast to see if other devices have found a path or are connected otherwise, paths may form in a completely topology unaware manner.

2-2-8 Authenticity and Security

One of the issues that the designers of CCNx are trying to solve up front is the ability to review authenticity of gathered information. Where IP uses encryption of transport tunnels, in NDN there is not transport tunnel from the requester to the generator. Therefore, we need to change current-day public-key infrastructures to ensure that information a) is actually authentic content generated by the intended author, b) is not seen by other people.

CCNx solves authenticity problems by obliging senders of ContentObject to sign those messages. In the current version signing is done by using self-generated self-signed keys, though a system such as our current public-key infrastructure is proposed. Although signing each ContentObject looks like a big overhead [5] we need to consider that requesting a public key or certificate is in fact also a request for content [1]. Therefore, analysis needs to be done to show whether the retrieval of public keys and certificates can be optimized to the point that its overhead accounts for the safety in return.

A SHA-256 digest algorithm is currently used to generate the last name component for any ContentObject (possibly a segment of a larger file or stream) to ensure that the complete name addresses a unique piece of content even if different authors were to generate similar data for the same name prefixes. A signature over either the complete ContentObject, or the complete file built up out of multiple segments is calculated and added up front to the first ContentObject. Since the signature is the first property of the ContentObject, even intermediate forwarding nodes can decide to check a message's signature when in doubt. Each requesting application of information is obliged to verify every message's authenticity, while an intermediate forwarding node may optionally do so [20].

2-3 Related Work

This section discusses previously related work and research concerning the topics of this thesis. It is split in 3 parts discussing Information Discovery, Dynamic End-Host Configuration and Transition Mechanisms. Within the related work we hope to find partial solutions to the problem of dynamically enabling end-users to fully participate (i.e. fetch and share data) in a Named Data Network.

2-3-1 Information Discovery

The process of Information Discovery concerns the process of finding or learning the location of information within a network. Parallel to topology discovery in IP, where generally IP prefix addresses are propagated, the process of information discovery is about propagating the availability of name prefixes across the network. In short, Information Discovery is the process of exchanging information between nodes and creating forwarding rules accordingly in the Forwarding Information Base.

[5] Every ContentObject whose authenticity is reviewed needs retrieval of the public key of its author and the keys of the instances who signed those keys up to the point that a certificate signed by a dually trusted certificate authority is found.

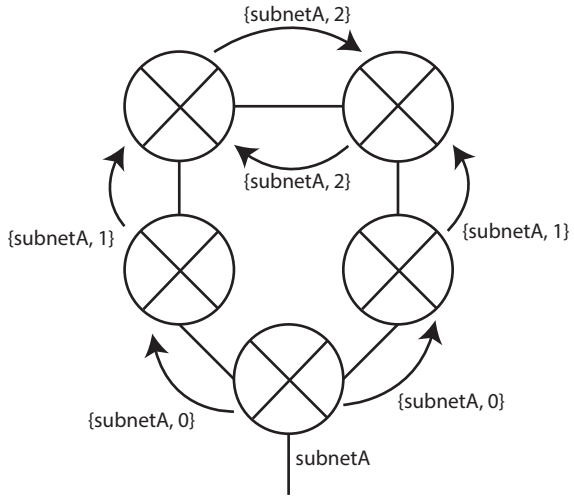


Figure 2-6: A graphic representation of the messages derived from a node with a single subnet propagating its available subnet through the network.

It is suggested by Jacobson et al. [39] to enhance IP topology discovery mechanisms [6] to support Named Data Networking. Currently 3 types of topology discovery mechanisms coexist on the Internet:

a) The less popular though very straightforward distance vector protocols such as RIP [47]. In a distance vector protocol every node broadcasts vectors containing available address prefixes and an initial cost of 0. Neighbouring nodes take over the offered availability of addresses, add a cost proportional to the link they received it from and broadcast it on all other interfaces. Whenever a node receives vectors pointing to a certain prefix from multiple interfaces, the one with the lowest cost will prevail. The lowest-cost vector for any prefix is considered the lowest-cost route to that prefix and is added to the local forwarding table and propagated on all other interfaces. The vectors being exchanged for every known prefix contain the summed cost and the direction or interface (possibly an IP address) the vector is sent from. This behaviour results in a distributed shortest path calculation.

Distance vector protocols are not very popular due to count-to-infinity problems, the fact that routing loops can occur and the fact that the diameter of the network is bound to the highest possible cost number. RIP, for example, does not support networks whose largest path is longer than 16 links. Figure 2-6 shows a graphic representation of the messages derived from a node with a single subnet propagating its available subnet through the network.

b) The more popular link-state routing algorithms are very often used by ISPs and companies for intra-AS [7] topology discovery [51]. Instead of using a distributed shortest path calculation, all routers exchange all link-states they have possible heard of defined by a connection cost between two routers. Routers are identified by a unique IP-like address, possibly its real IP-address, which is used to identify links between two routers.

Every router collects and broadcasts all updates it receives and uses this information to create

[6] Such as OSPF [53] and IS-IS [8] for intra-AS topology discovery and BGP [49] for intra-AS topology discovery

[7] an Autonomous System refers to a finite network under administration of a single authority, intra-AS topology discovery therefore refers to a topology discovery mechanism within such a network

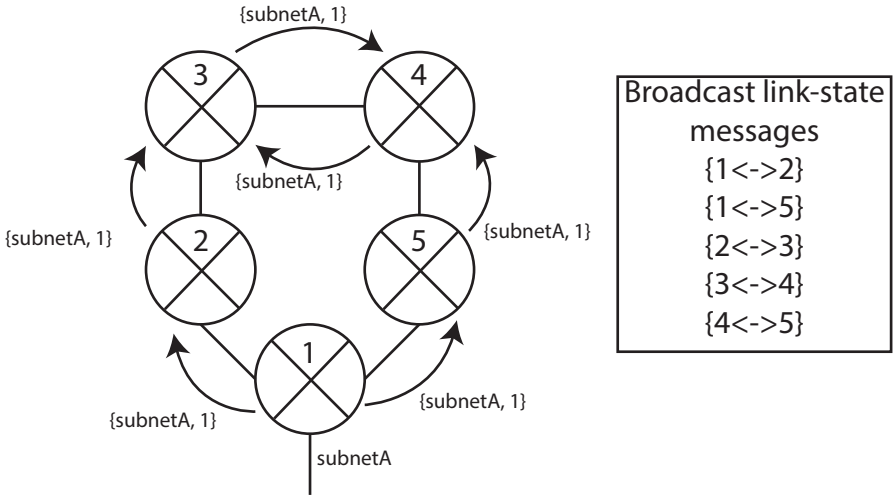


Figure 2-7: A graphic representation of the messages derived from a node with a single subnet propagating its available subnet through the network. The box named "Broadcast link-state messages" indicates which messages are shared between all nodes in order to build a graph.

a graph representing the network. The graph can then be used to calculate shortest paths to all routers and their subnets, after which forwarding rules are created. Whenever a link changes, is added or removed, this knowledge is sent through the network, the graph is changed accordingly and recalculation can occur. Since shortest-paths are strictly calculated using the created graph, the mechanism does not suffer from the problems distance vector algorithms have. Figure 2-7 shows an overview of all the link-state messages broadcast between all nodes, and the propagation of a message indicating a certain node serves a specific subnet.

The link-state algorithm protocols OSPF [53] and IS-IS [8] are often used by ISPs for intra-AS topology discovery. Due to the fact that a link-state algorithm gathers a complete view of the associated network it is unsuitable for a network as large as the Internet. Next to the fact that creating the graph is as complex as the number of links within the network, a shortest path algorithm generally has a time- and work-complexity of $O(N^2)$, where N is the number of nodes in the network.[51].

c) For topology discovery among Autonomous Systems (inter-AS) ISPs use the Border Gateway Protocol [49], which is a path-vector routing algorithm [51]. In a path-vector algorithm, nodes broadcast messages containing their unique identifier and the prefixes they serve. Every node receiving such messages adds its own identifier to the message, and broadcasts it on all other available links. This results in vectors containing address prefixes and paths to the address prefix described by the list of identifiers. When a node receives multiple paths to an endpoint it can choose between paths based on

- path length,
- if applicable, path cost,
- trustworthiness of the administrative organizations maintaining the paths,
- particular agreements it has made with other nodes

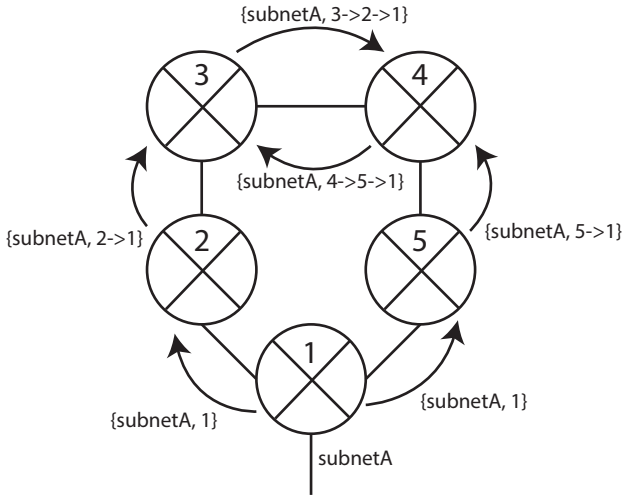


Figure 2-8: A graphic representation of the messages derived from a node with a single subnet propagating its available subnet through the network using a path-vector algorithm.

- and all other constraints which can be led back to the path.

After which, the chosen path is propagated to all other nodes with the identifier of the propagating node attached to the path. Path-vector algorithms have the positive property that they can give insight in the path messages will travel without the need to generate a complete graph of the network. Routing loops are easily determined: whenever a node receives a message already containing its own id (or 2 other equal identifiers) in the path-vector that specific message needs to be discarded. Figure 2-8 shows the process of subnet propagation through a simple network using a path-vector algorithm. The number of exchanged messages is equal to a distance vector protocol, though has less of the distance vector specific disadvantages and gives more insight into the path a chosen forwarding rule would travel.

BGP is the globally used path-vector protocol to interconnect ISPs and other ASes. Every AS is considered to be a single node with its own unique Autonomous System Number (ASN) administered by the Internet Assigned Numbers Authority (IANA). Since the ASNs map to organizations, whenever a party does not trust an organization (for example for political reasons) it can easily discard paths containing ASNs from that organization. Paths containing ASes an organization might have special agreements with, can be handled accordingly. Since the complete AS is considered to be a single node, ASes still need internal discovery mechanisms such as OSPF, IS-IS or run an internal instance of BGP which does take all nodes into account.

Since IS-IS is protocol-independent it can very easily be converted to other OSI layer 3 protocols by introducing a new protocol identifier without altering the protocol itself. OSPF and BGP, however, both are IPv4 specific routing protocols and therefore need more adaptations to serve other types of networks. Luckily, both mechanisms have the possibility to add extra options to the process of IP topology discovery. Although the primary focus of the processes is to exchange information about reachable IP subnets, it can also be reused to also exchange information about available NDN networks and their name prefixes [6][5].

An NDN implementation based on top of OSPF, called OSPFN, already exists. It consists

of an extra daemon connecting to the OSPF daemon implemented by Quagga [41]. Through the connection it can read received information about NDN enabled nodes and inject prefix names hosted by the system into the IP discovery mechanism. The experiments in chapter 3 will show the results of OSPFN in practice. Unfortunately, neither IS-IS nor BGP currently have a NDN enabled version.

2-3-2 Dynamic End-Host Configuration

In order to enable end-users to quickly use a network we need ways to dynamically configure devices with appropriate gateways and settings. Parallel to the Dynamic Host Configuration Protocol [28] in IP, a NDN enabled DHCP service called CCNx-DHCP [46] is distributed.

CCNx-DHCP relies on server daemons listening on multicast-faces, the NDN equivalence of an IP multicast address, responding to the name `ccnx:/local/dhcp`. CCNx clients entering the network multicast an Interest to the name `ccnx:/local/dhcp/content` indicating that they need network settings. The server daemon responds by sending a ContentObject, satisfying the Interest, containing a set of forwarding rules the client can use. The set of forwarding rules can contain a gateway route prefix matching all possible names (`ccnx:/`).

The CCNx-DHCP daemons can be used to easily set up a multitude of CCNx clients. Unfortunately there is no user or device specific parametrization of settings possible, any configurable settings apply to all clients. The protocol can only inform clients which forwarding rules they should use to access content. It is impossible to configure clients with dynamically generated names they can use to globally share content. There is no registration of devices currently relying on information offered by the server either, while this might be preferred for administrative purposes.

In chapter 3 we will test the functionality of CCNx-DHCP by a series of experiments and further exploit the possibilities of dynamic end-user configuration in topology creation and name generation.

2-3-3 Transition Mechanisms

Although we have considered ways to spread the knowledge of available information through different domains of networks using information discovery, there is still the case that a user requests data that is served in a piece of network that is unreachable via CCN or unknown to the requester.

In such a case we have CCN islands that we want to interconnect. The interconnection of disconnected CCN islands can happen through manual configuration of IP or UDP tunnels between the islands. Though this is a good solution for a network where a limited amount of islands coexist, when the usage of CCN grows manual configuration becomes infeasible.

What we need are automated ways to configure connectivity between CCN islands. Luckily the same research question has already been solved in the scenario of IPv6 islands trying to connect through IPv4 networks. In the next paragraphs we will discuss a few IPv6 transmission mechanisms and then discuss how these could be used in CCN.

One of the mechanisms IPv6 uses to establish a connection between end points is called 6in4. In 6in4 a user or administrator manually registers two endpoints in the form of IPv6 prefixes

or routes and 2 IPv4 addresses. IPv6 traffic between the hosts is directly encapsulated in an IPv4 header and transmitted to the other IPv4 host. The receiving party then unpacks the IPv4 packet, resulting in the IPv6 packet, and then treats that IPv6 packet as if it just arrived on an interface [34]. The endpoint connected to the native IPv6 network propagates an agreed IPv6 subnet for the other client and encapsulates and tunnels received packets for it. This mechanism is similar to the earlier described manual configuration of IP and UDP tunnels between CCN islands.

Since manual configuration is not feasible on large networks, a dynamic variation on 6in4 has been engineered. 6to4 [9] functionally uses the same techniques to encapsulate and tunnel IPv6 packets over an IPv4-only network and has additional mechanisms to dynamically set up tunnels. Clients on a non-native IPv6 network get an IPv6 address block mapped to their IPv4 address starting with *2002::/16*, this means that an IPv6 address starting with 2002 can be translated back to an IPv4 endpoint address and vice versa. If a non-native peer needs to send out an IPv6 packet it can use a host address from the IPv4-mapped IPv6 address space and encapsulate the message in a IPv4 packet directed to either

- the IPv4-calculated address if the packet is also addressed to an address from the 2002::1/16 range,
- or the anycast IPv4 address 192.88.99.1 for which any node that can unpack encapsulated packets may answer.

Any nodes with the capability to encapsulate and unpack IPv6 packets from these ranges propagate the respective IPv6 and IPv4 address(es) by means of topology discovery mechanisms.

In practice this mechanism works quite well, unfortunately it is not completely fault free. The IPv4 anycast IP address guarantees connecting to *a* IPv6 network, not a possible main or all public IPv6 networks. Even when it is possible to sent IPv6 packets to an IPv6 network using encapsulation, it is not guaranteed that packets can also travel back. The receiving IPv6 network needs to have access to an encapsulating tunnel endpoint publishing the IP range 2002::1/16, if this address space is not available in its network returning packets will not find their way across the network.

In the case of NDN, a third possibility proposed by [39] to dynamically create tunnels is to use DNS lookups to find tunnel end-points serving networks. For example, when a network does not have forwarding rules to the NDN domain `ccnx:/tudelft.net`, a border gateway node may request the tunnel end-point by requesting a DNS A or AAAA lookup to `ccn.tudelft.nl` or a SRV or newly added type of request to `tudelft.net`. The border gateway node encapsulates Interests destined to the NDN domain and sends it to the other network over IP. The receiving node creates a reverse tunnel based on the source-address, forwards the Interest according to its FIB and returns satisfying ContentObjects into the tunnel. We will use the idea of DNS lookups to find entities serving content in chapter 4.

Chapter 3

Dynamic Configuration and Sharing of Information

One of the problems that needs to be solved in NDN is the process of dynamic configuration and information discovery (analogous to topology discovery in IP networking). End-users need to be made aware of the available information and devices need to be configured accordingly. Although a few information discovery mechanisms already exist for CCNx [63][46], dynamically sharing content for end-users without participating in difficult topology discovery mechanisms is not yet possible in NDN. Instead, one needs to propagate an individually registered name prefix using information discovery processes, such as OSPFN [63], comparable to full-size topology discovery mechanisms in order to globally share content. Whilst the power of current networks is that after devices are auto-configured, given the user has the right credentials for closed networks, users can use the full potential of the network.

In the next section we will set up experiments to test different combinations of the available auto-configuration and information discovery mechanisms in NDN in order to find their functional boundaries. Subsequently, we will give a summary of the problems found and the implications of those problems. Based on the problems found and the properties of the network we will propose a solution that both dynamically configures devices and enables end-users to globally share information using dynamically generated names. Appendixes A, B and refAppendixCombined contain log-files and status reports from the different daemons used in the experiments. These files and reports are used to support the conclusions of the experiments.

3-1 Experiments: Information Discovery

In order to verify the functionality and the limits of the currently existing dynamic configuration and information discovery mechanisms CCNx-DHCP and OSPFN, which are both

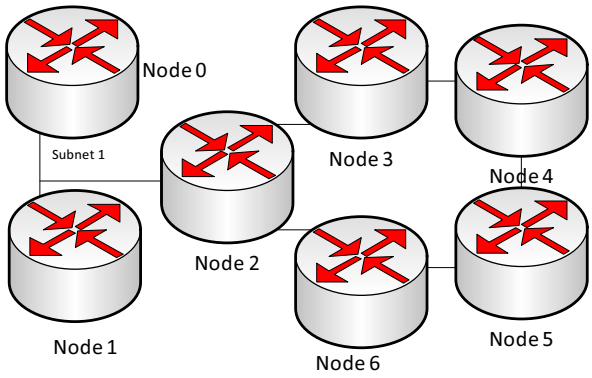


Figure 3-1: Experiment network topology

discussed in sections 2-3-1 and 2-3-2, we have set up several experiments. *The main goal of the experiments is to find out to what extent the combination of these two mechanisms offers dynamic configuration in order to enable end-users to dynamically* fetch and share *content.*

First, we will explain the setup of the experiment environment to enable readers to verify the results of these experiments. After that we will use two lab setups to test both CCNx-DHCP and OSPFN independently. Finally, we will use a lab setup in which we use CCNx-DHCP to dynamically connect to a network which is configured by OSPFN to simulate a situation with the most similarities to current-day Internet connections.

3-1-1 Experiment Environment

We use 7 different virtual machines based on VMware connected using IP subnets. Every subnet has its own IP range in the form of *10.12.X.0/24*, where *X* is unique for the subnet within the experiment environment. Each node also has a management interface with one IP address taken from the range *172.19.5.0/24*. This range is solely used for management purposes and is excluded from usage within the experiments. The nodes are numbered 0 to 6 and are designated as follows:

- Node 0 is used occasionally when the experiment requires more than two nodes within the same subnet.
- Node 1 fulfills the role of a SOHO (small office or home office) network that needs to be connected via an ISP.
- Nodes 2 to 6 simulate the role of an Autonomous System of a single ISP and are connected in a circle in order to quickly simulate multiple paths to a single destination.

Node 1 connects to node 2 as if it were connected to an ISP, nodes 1 to 6 will play the role of an ISP administered Autonomous System which needs to be auto configured using information discovery mechanisms. Figure 3-1 presents the network topology used during the experiments, the interfaces of the networks are connected to each other by means of VLANs.

Not all experiments will use all nodes, though the basic set up remains unchanged throughout most experiments. All nodes are running Ubuntu 10.04 LTS with CCNx version 0.4.2, OSPFN

commit 0251aab [63] and ccnx-dhcp commit 6df96d849f [46]. End-to-end connectivity and information dispatch is established by the utility called ccnping [61] which is reconfigured to add the hostname of the generator of the reply to the body of the ContentObject. The ccnping client is altered slightly[1] to print this information upon arrival at the requesting node. These alterations enable us to verify where a reply came from in case multiple nodes are allowed to generate content for a certain name prefix.

3-1-2 CCNx-DHCP Experiments

During the first set of experiments, we will verify the functionality of the CCNx-DHCP daemon by dynamically configuring end-users. First we will verify basic functionality by using a single server against multiple clients. Later we will enhance the experiment environment by introducing several complexities into the network such as multiple gateway servers and multiple links. Configuration of the client and servers is done by altering the files *ccn_dhcp_server.conf* or *ccn_dhcp_client.conf* and either running the server daemon *ccndhcpserver* or the client daemon *ccndhcpnode*.

Single server

In this experiment we will verify the basic functionality of the CCNx-DHCP daemon. For this experiment we solely use nodes 0, 1 and 2 from figure 3-1 in a single subnet and start by designating node 2 as a server and node 1 as a client. Node 2, the server, serves as a gateway forwarding rule (`ccnx:/`) to any client request. Theoretically, the client or end-user, node 1, multicasts an Interest for the configuration information on the CCNx Multicast face and receives the gateway forwarding rules accordingly. In the FIB of the client, the gateway route is added to the lists of forwarding rules and the dynamic host configuration is successfully terminated.

As can be seen in appendix A-1-1 and A-1-2 both node 1 and node 2 have a multicast NDN face registered on the IP multicast adress 224.0.23.170 (which is reserved for the usage by CCNx in general) with a forwarding entry for the name `ccnx:/local/dhcp` pointing into the multicast face. Node 2 also has a gateway forwarding rule, which has been dynamically added by the client, pointing towards the multicast face.

The client can now access information generated or forwardable by node 2. If node 2 was part of an ISP and had access to a greater part of the Internet, one could argue that node 1 is now dynamically configured to access the Internet since node 2 can forward the Interests of node 1. Unfortunately, there is no way for the end-user to dynamically share content since there are no forwarding rules added leading from other nodes to the client.

In order to make this experiment a little bit more concrete, we have reconfigured the server to create a gateway rule pointing to its IP unicast address and added a second client (node 0 from figure 3-1) to the subnet. Figure 3-2 shows a graphical representation of the two client applications requesting configuration from the server application via the CCNx routing daemons on the aforementioned nodes. As shown in Appendices A-1-3, A-1-4 and A-1-5 all three nodes have created the required multicast face and set up appropriate routing. The

[1] By adding a simple *printf()* statement at the right line.

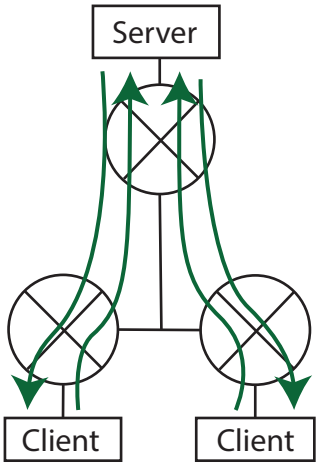


Figure 3-2: The messaging process in which client CCNx-DHCP applications request dynamic configuration from a server. The green lines show the path of the Interest encapsulated request and ContentObject encapsulated response messages across the different CCNx forwarding daemons of the nodes.

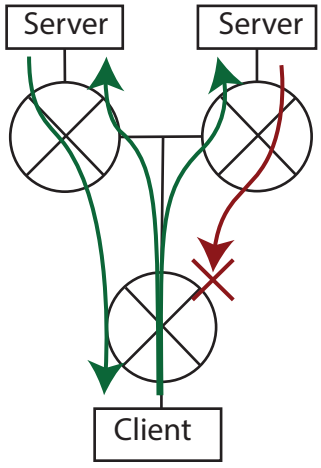


Figure 3-3: The messaging process in which a client CCNx-DHCP request is forwarded to two servers, though only the ContentObject encapsulated response of one server is forwarded by the client's CCNx daemon. The CCNx daemon considers the second ContentObject to be superfluous, since the first ContentObject has already satisfied the initial Interest.

clients (node 0 and 1) both have generated a new face pointing to the unicast address of node 2 and set up a gateway forwarding rule accordingly.

From this small experiment we can conclude that CCNx-DHCP can indeed dynamically configure multiple clients without the need of manual configuration. In the next experiments we will try more complex situations to determine whether the daemon can handle a higher complexity of networks.

Multiple servers

One advantage of NDN over IP networking is the possibility that a single prefix might be accessible via multiple routes. In such a case the Interest packet for the prefix is duplicated and sent out over all links that have a matching forwarding rule in the FIB [19]. The NDN transport layer, which as discussed in section 2-2-7 is still under construction, will be responsible for employing heuristics to potentially load-balance or prefer better performing faces over the other. The NDN architecture supports multi-homing where access to a greater network can be achieved via more than one ISP. This means that clients may have to deal with multiple CCNx-DHCP servers whose configuration parameters may need to be combined in order to gain advantage from multiple gateways to access content[2].

In order to verify the behavior of CCNx-DHCP in such a multi-homed environment, we have reconfigured node 1 from figure 3-1 to act as a server providing its own configuration

[2]Regular DHCP also supports the availability of multiple servers, though this is often implemented for means of robustness by redundancy

parameters. Where the client sends out an Interest looking for configuration parameters, the response is encapsulated within a ContentObject. Since an Interest is satisfied by a single ContentObject matching all requirements [19] we expect the client to only receive and digest the first response instead of receiving all possible configuration parameters and make a final decision based on those parameters.

Unfortunately, the experiment reveals this expectation to be true. Where Appendices A-2-1 and A-2-2 show the usual faces and forwarding rules for a server, Appendices A-2-3 and A-2-4 show the client accepting either the settings[3] from either node 1 or node 2. As discussed the configuration parameters are encapsulated within a ContentObject, thus the client will receive at most one set of configuration parameters since the CCNx daemon considers the original Interest to be fulfilled and will not return subsequent content without explicit request. A graphical representation of this is shown in figure 3-3 where the first response of a server is accepted and forwarded, though the second response is discarded. This means that a client cannot gain advantage of the availability of multiple gateways or paths and the decision of the path taken will be based on probability instead of availability, preference or the strategy layer.

Possible solutions to this problem, which will be deployed in our proposal, include either

- encapsulating the response in a new Interest packet which would result in all responses to be multicast to all joined nodes[4],
- or having the client re-express the Interest after each response and use the exclude options as discussed in section 2-2-5 to disregard sources from where earlier results have been received.

From this experiment we can conclude that although CCNx-DHCP can dynamically configure multiple clients, it does not yet

- have a solution to conform to the NDN multipath philosophy,
- give the client the option to compare and decide on[5] or merge configuration settings.

We will use the lessons learned from this experiment and incorporate them in our proposal.

Multiple interfaces and subnets

One of the goals of generic DHCP in IP is to dynamically configure clients on local networks beyond the scope of the topology discovery process of the ISP. Even when the local network consists of multiple links and subnets, which happens regularly in larger companies or institutions, the dynamic configuration process has to be solid to function properly. In this experiment we have connected multiple links to different CCNx-DHCP clients, as well as to servers in order to evaluate its behavior in more complicated networks.

[3]The distinction can be made by looking at the dynamically configured unicast gateway address, both servers try to configure their own unicast IP address as gateway

[4]including other clients, as is also the case with regular DHCP

[5]as is usual for DHCP [28]

We already know that the client will accept the response of at most one server, but keep in mind this was tested on a network with solely one subnet.

In this experiment we have configured one node as either client or server and 2 nodes, connected via different subnets as opposite server or client, in order to create a scenario where a client is connected to multiple links and a scenario where a server is connected to multiple links. This setup facilitates to see how both server and client behave when multiple links to clients and servers exist.

Unfortunately, the experiment reveals that both the clients and the servers are unable to manage multiple interfaces. The CCNx multicast face is, both by client and server, registered at one IP interface at most, which means not all possible servers can be found by the clients and servers cannot serve more than one subnet at a time. Though acceptable for a client (they usually only connect using a single interface), a situation where a DHCP server or DHCP relay server needs to control multiple subnets is quite common.

After more thorough investigation into the source of CCNx-DHCP [46] and that of CCNx itself two problems appear:

1. The CCNx-daemon tries to set up exactly 1 multicast face without specifications on which IP interface this should occur. Therefore, there will be only 1 multicast face connected to the IP interface selected by the kernel's IP forwarding table. The IP interface selected being either the default gateway, a specific forwarding rule for the multicast IP range or none at all. This behaviour is shown in the logs of attachment A-3-1 where at first the server can not create a multicast face at all due to missing forwarding rules for the multicast IP range. After adding IP forwarding rules for the multicast IP range pointing to a specific IP interface the multicast face can be created.

2. A bug in CCNx [2] prevents clients to create multiple multicast faces to the same multicast address on different IP interfaces. In the implementation of our proposal we have implemented a patch for CCNx [3] regarding this bug which has been submitted to PARC[6].

In short, we can state that the CCNx-DHCP currently has no support for nodes connected to multiple interfaces.

Conclusion drawn from CCNx-DHCP experiments

The CCNx-DHCP server and client daemon show potential to dynamically configure many clients at a time. However, more work needs to be done in the fields of link- and node-specific configuration and for networks in which multiple servers and Internet access gateways exist. Stateful registration of clients would enable to create node-specific configuration and would be a step closer towards our proposal in which names are dynamically reserved for content shared by clients.

[6]The patch works good for our NDN testbed. Michael Plass, a researcher and core developer at PARC, confirmed that the patch was a proper one though they need to extend it with support for IPv6-backed networks before it can be released.

3-1-3 OSPFN Experiments

OSPFN is an NDN addition to the OSPF daemon from the Quagga-routing suite [41]. As discussed in section 2-3-1 OSPFN uses an API to connect to an OSPF daemon regulating an already configured IP topology. Names are inserted into the network using so-called Opaque Link-State Advertisement options which can basically add application specification information to the information shared about links and routers [6]. OSPFN daemons can subsequently read the propagated Opaque LSAs containing the name-to-router tuples.

In this section we will verify the functionality of OSPFN in several situations common to both regular and NDN enabled networks.

OSPFN across a multipath NDN network

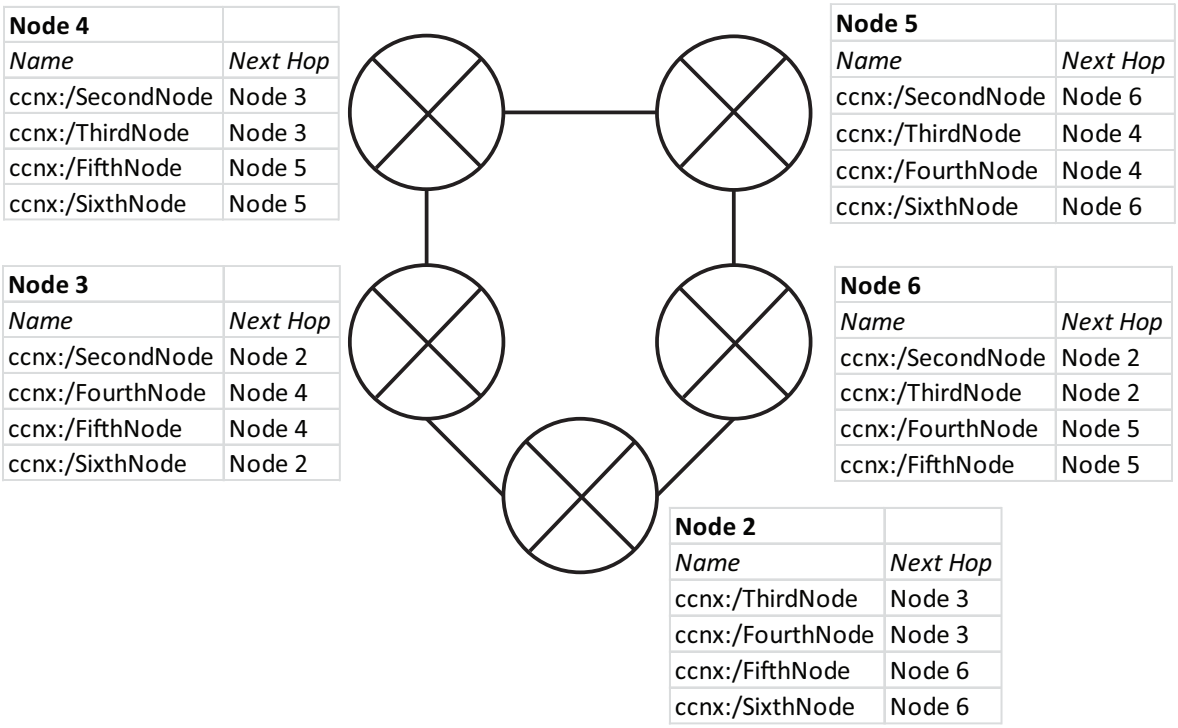
During this experiment we will verify the functionality of OSPFN in a network where content for a certain prefix is generated by at most one node. Nodes 2 to 6 from figure 3-1 are placed in a circle to play the role of an AS in which multiple paths to content generators coexist. Every node is given a name in the form of `ccnx:/SecondNode` up till `ccnx:/SixthNode` in order to have unique names within the experiment. After starting, the OSPF and OSPFN daemons fill the IP routing table and CCNx FIB quickly with forwarding rules to the different IP subnets and NDN names. As can be seen from the CCNx status reports - Appendices B-1-1 up to B-1-5 - the forwarding rules always point in the direction of the shortest path to their destination. For example, node 2 has a forward for the name `ccnx:/FourthNode` to node 3, who then has a forward to node 4. Figure 3-4 shows an overview of all NDN forwarding rules created by OSPFN. When testing end-to-end connectivity and information dispatch with the ccnping application, network connectivity and therefore the information discovery process work seamlessly.

This experiment shows that the OSPFN daemon enables network operators to automatically set up NDN forwarding rules in an environment where multiple paths to a destination coexist. Whenever a node propagates a certain namespace, which in this case was a location-dependent name, other nodes set up the right routes to these destinations.

Information generation by multiple nodes

One of the main differences between NDN and host-to-host networking is that information is requested independent of the location where it is generated. This enables scenarios where content can be generated by multiple nodes governed by the same organization. Since information can be generated independent from its location, it is possible that authentic information can be generated by multiple geographic locations, therefore it is important for an information discovery process to support situations where a single prefix has forwarding rules pointing to different geographical locations.

In order to simulate such an environment we have two nodes, 4 and 5, propagate that they serve content for the same name (for convenience we took `ccnx:/ourRedundantName`) and repeat the previous experiment. As shown in the CCNx status reports in Appendices B-2-1 up to B-2-5, nodes for which nodes 4 and 5 have shortest paths pointing in the same directions (i.e. nodes 3 and 6) add a single forwarding rule pointing in the direction of both shortest



Node 4	
Name	*Next Hop*
ccnx:/SecondNode	Node 3
ccnx:/ThirdNode	Node 3
ccnx:/FifthNode	Node 5
ccnx:/SixthNode	Node 5

Node 5	
Name	*Next Hop*
ccnx:/SecondNode	Node 6
ccnx:/ThirdNode	Node 4
ccnx:/FourthNode	Node 4
ccnx:/SixthNode	Node 6

Node 3	
Name	*Next Hop*
ccnx:/SecondNode	Node 2
ccnx:/FourthNode	Node 4
ccnx:/FifthNode	Node 4
ccnx:/SixthNode	Node 2

Node 6	
Name	*Next Hop*
ccnx:/SecondNode	Node 2
ccnx:/ThirdNode	Node 2
ccnx:/FourthNode	Node 5
ccnx:/FifthNode	Node 5

Node 2	
Name	*Next Hop*
ccnx:/ThirdNode	Node 3
ccnx:/FourthNode	Node 3
ccnx:/FifthNode	Node 6
ccnx:/SixthNode	Node 6

Figure 3-4: An overview of all NDN forwarding rules created by OSPFN. The forwarding rules of all nodes form, similar to forwarding rules in IP, a chain of forwarding rules from each source to each destination.

paths. Nodes for which the direction of the shortest paths to nodes 4 and 5 differ (which is the case for node 2) add two forwarding rules each pointing in the direction of both shortest paths (either node 4 or node 5).

This experiment shows that OSPFN can support complex situations where data is generated by multiple nodes within a multipath network using location independent names. In the future, the strategy layer of CCNx could employ heuristics on top of the created forwarding rules in order to further balance or choose between the available content generators.

3-1-4 Combined OSPFN and CCNx-DHCP

Analogous to IP networking, an end-user or Internet connection might not be part of the intra-AS information discovery process. A scenario in which end-users are configured dynamically using CCNx-DHCP by a node that is part of the intra-AS information discovery process appears plausible and scalable. This experiment consists of connecting node 1 as an end-user to the multi-path, multi-destination networks from the previous OSPFN experiments. The end-user does not run any information discovery process, but instead runs a CCNx-DHCP client. The node it is connected by to the rest of the AS, node 2, runs a CCNx-DHCP server next to the intra-AS information discovery process offered by OSPFN. A gateway route (`ccnx:/`) pointing towards the virtual Autonomous System of the ISP is offered to the end-user in order to gain full connectivity to the network as shown in Appendix C-1. End-to-end connectivity is, like in the previous experiments, verified using the altered ccnping

application.

The results in Appendix C-2 show that both answers varying from nodes 4 and 5 (who are both responsible for generating content for the name ccnx:/ourRedundantName) are received by the client configured via CCNx-DHCP. This experiment shows that it is already possible to dynamically access information without any end-user network configuration or running information discovery processes.

3-1-5 Experiment conclusions

In the conducted experiments both OSPFN and CCNx-DHCP work well in the sense that they provide ways for dynamically connected end-users to learn about available information and how to retrieve that information. When using OSPFN it is not only possible to construct routes that enable a node to request content, but also to distribute the availability of content at the node itself. Unfortunately, neither mechanism allows for end-users to dynamically share content without registering their own first name component and configuring the available routing protocols.

3-2 Proposal: Recursive Name Aggregation

In this section, we will propose to combine a naming convention, based on the idea of ISP-based name aggregation and a local configuration and information discovery process based on DHCP resulting in a path-vector protocol.

Our philosophy is that a node within an AS responsible for connecting end-users, such as home offices, is given a certain prefix for which routes are set up. This node is called the Entrypoint. Entities connected to the ISP via that node are given subnames of that prefix. Repeating this process recursively for devices who are again connected to the previous device generates prefixes that become even longer when descending into the dynamically configured network. When every node sets up forwarding rules for the subnames they have proposed to their child nodes, the dynamically generated names become globally available since:

1. the name is globally unique - which is one of the demands stated by the CCNx name conventions[12],
2. the presence of the primary prefix used by the first node to start the process is globally known due to information discovery within the AS and to the Internet,
3. all intermediate nodes forward subnames to the appropriate descendant.

Forwarding tables outside of the local dynamic network and AS remain unchanged, which helps keeping global forwarding tables scalable.

Let us consider a short example: An ISP receives Interests for ccnx:/isp.net, the availability of the information by this ISP is made globally aware by means of information discovery. The ISP forwards Interests to Bob's home router for ccnx:/isp.net/bob. Bob's router again

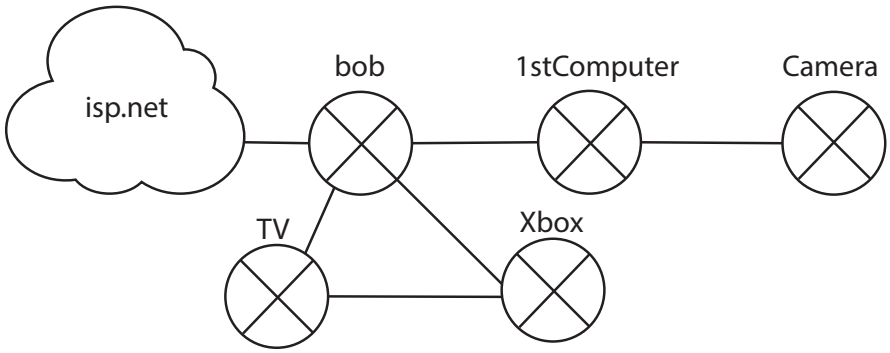


Figure 3-5: A graphical representation of a possible end-user home network. Bob's ISP forwards Interests for `ccnx:/isp.net/bob` to his home router named *bob*. The router generates subnames of this name to forward Interests to connected devices, these devices can in return use their given name to recursively generate the name `ccnx:/isp.net/bob/1stComputer/Camera` for the camera. This name can be used to share photos from the camera to the Internet or view them from other devices within the house.

generates and forwards Interests for the prefix `ccnx:/isp.net/bob/1stComputer` to Bob's computer. When Bob plugs in a device, such as a photo camera, into his computer, the computer will again generate a prefix for that particular device (e.g., `ccnx:/isp.net/bob/1stComputer/camera`). The camera can now share the photos it has stored using the name `ccnx:/isp.net/bob/1stComputer/camera/photos`. The deeper one repeats this process into the network the longer the names will become. Due to the aggregation of the names back to the names of the intermediate and entry nodes the names will stay globally unique and accessible.

Although we have introduced location dependency within the dynamically generated names (as shown in figure 3-5 the names represent physical nodes and structure within a network) the benefits of NDN over a host-to-host network are maintained. The mechanism still allows for caching of content on subsequent nodes which results in the ability to serve more users than one would with a regular IP connection. A mapping service, possibly an NDN equivalent of DNS, could translate user-friendly names to one or more dynamically generated names. Both can be requested efficiently using NDN in a distributed fashion. If the NDN node responsible for processing the Interests to the registered user-friendly name is able to

- duplicate and translate the Interests to all possible location dependent names and
- set up flow control between the stream of Interests and returning ContentObjects to and from the different locations,

we have regained all benefits of NDN as described in [39]. *In Chapter 4 we will discuss a proposal for such a mapping system to solve this problem.*

3-3 Basic Dynamic Host Configuration and Name Generation Description

Using our publicly available prototype [3], we have verified the following protocol description. Newly connected nodes first act as clients to gain access to the network. As shown in figure 3-6, the client queries the local subnets by multicasting a *Discovery* message containing its host identifier and preferred hostname. All available servers respond with zero, one or more *Offer* messages containing the following properties:

- The original entry-point, in order to identify unique entry-points and their possible multitude of paths to that Entrypoint.
- The aggregated name of that entry-point the server is willing to forward to the client, made up from its own generated name extended with the hostname preferred by the client.
- The cost and path-vector of the route.

The responses are currently encapsulated within Interests in order to enable the client to receive multiple responses and make a decision based on the cost and path which Offers, possibly more than one, it wishes to use. This is done to prevent the behaviour experienced with the CCNx-DHCP daemon in the experiment of section 3-1-2 where only the reply of one server could be digested due to the fact that the CCNx daemon only returned the first received response. Packing all messages in Interests results in the responses being multicast to all clients and servers within the subnets, even when they have not requested that response. This behaviour complies to the behaviour of DHCP with IP, where all acknowledgments are sent to all servers as a means to say a client has chosen another server's configuration [28].

Since it is not strictly necessary to multicast all Offers and Acknowledgments[7] to all nodes, it is also possible to encapsulate those in ContentObjects. Clients then need to recurse through all possible ContentObjects by re-expressing their Interests and excluding previously received results using the Exclude option (as discussed in section 3-1-2) to prevent receiving solely the first returned Offer. Since we were able to verify behaviour with our implementation using the encapsulation by Interests (the client applications receive all Interest encapsulated responses without re-expressing Interest) and the fact that the recursion by repeated Interests poses a greater effort of the network, we have chosen to keep responses Interest encapsulated. If necessary or preferred, future releases can be altered to encapsulate responses in ContentObjects and have clients recurse through them.

At label 1 in figure 3-6, the new peer has received all offers and calculates a shortest path to each entry-point. For each shortest path the client asks the chosen server whether it is allowed (see label 2) to use that route and name by sending a *Request* message[8]. A client can request the complete set, a subset, or none of the configuration parameters offered by a peer. The client can decide to choose rules based on the cost or path offered by the rule. When the

[7] Which both are server-to-client replies.

[8] This behaviour is actually very similar to DHCP [28], due to the long history of dynamic configuration by DHCP we decided to reuse common parts of its protocol

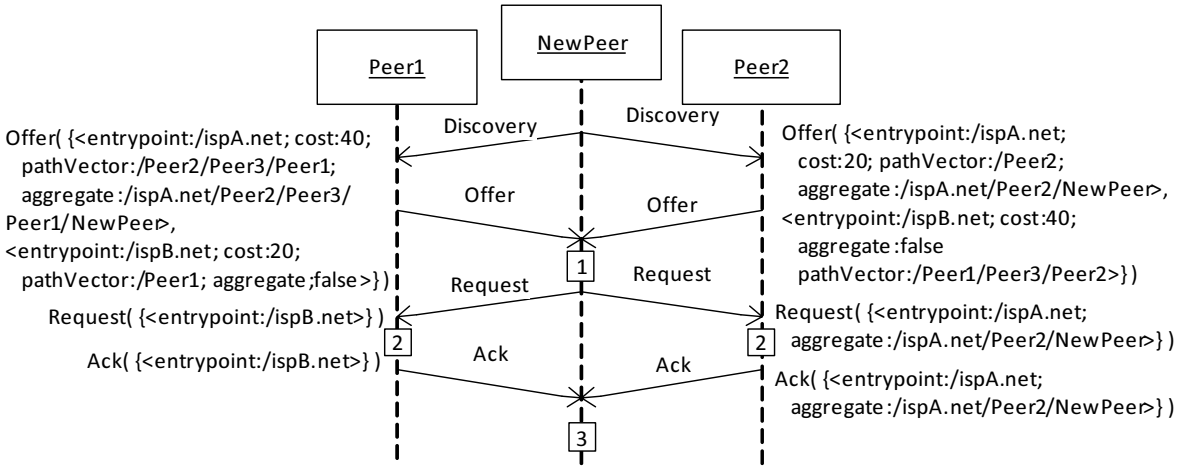


Figure 3-6: Example communication protocol

request is allowed, the server adds forwarding rules for the generated name towards the new peer and acknowledges the usage of the rules (at 2).

As soon as the client receives the *Acknowledgment* message it also adds a forwarding rule for the name of the entry-point pointing towards the chosen path. This results in a chain of forwarding rules on all nodes towards the original entry-point which guarantees the new client can reach all other nodes who have derived a name from this entry-point. An Interest to another node will simply travel back into the direction of the entry-point until it reaches a node that has a more precise[9] forwarding rule pointing towards the desired node or one of its aggregators.

From this moment on, the client also starts acting as a potential server to other interfaces using its own name as aggregation base, though maintaining the original entry-point in order to distinguish from other entry-points. The server can offer newly generated aggregated names to new clients when requested to do so and set up the appropriate forwarding rules to ensure global reachability.

This protocol enables end-users to start sharing content globally using the unique names generated by the proposed configuration-less topology discovery and configuration protocol.

3-4 Formal Dynamic Host Configuration and Name Generation Protocol Description

This section discusses the communication between nodes necessary to initialize and configure new nodes. First, we will shortly summarize the communication between client and already existing peers, after which we will discuss the different messages in more detail.

As discussed in section 3-3 and shown in figure 3-6, the initialization of a new node in the network consists out of 4 types of messages:

[9]Remember that CCNx Interests are forwarded according to the longest prefix matching forwarding rule

- The client sends out a *Discovery* message claiming its existence and requesting for existing paths to entry-points to all nodes within reach.
- Peers that can already act as servers, return *Offer* messages describing the paths and names they can serve against which cost.
- The client calculates which paths and names it will use from the offers and requests those by sending *Request* messages to the appropriate servers.
- If the request is a subset from the original offer, the server will add the appropriate forwarding rules and return with an *Acknowledgment*.

As soon as the client receives the acknowledgment it will add a forwarding rule for the entry-point pointing towards the chosen server in order to enable it to access data from other nodes using names from the same entry-point. The client is now initialized and can also start acting as a server to newly added clients.

The Discovery message has the following two properties:

```
Discovery ::= clientID
              clientHostName?
```

The clientID is a unique identifier or name within its subnet such as a public or private IP address, Ethernet-MAC address or other type of unique hardware address, as long as it is unique within its own broadcast reach. The clientID is used by the server to link offers, requests and acknowledgments to the right client and keep an administration of active clients. A client can optionally set the property clientHostName indicating a preferred hostname with which the server can extend the recursive name, the server may but does not need to conform to this request as the formed name may already be handed to another host.

The server replies with an Offer message stating the possible paths it can serve:

```
Offer ::= clientID
          serverID
          Rules*

Rules ::= Entrypoint
          Cost
          Path
          AggregatedName?
```

The clientID and serverID serve to identify to which client-server conversation the message belongs. The Offer contains zero or more Rules the server can offer. Each rule consists out of 3 parts:

- A name describing the Entrypoint; usually an Internet-routeable name.
- The cost of the path to that Entrypoint.

- A path-vector of clientIDs to that Entrypoint; which is used to avoid routing loops and it enables clients to take complexer considerations than solely cost-based decisions.
- The aggregated name the client can use, possibly extended with the previously described preferred hostname; although the description of the protocol so far mentions this property to be obligatory, section 3-5 further enhances the protocol due to which it will become optional.

From all received Offers and their Rules the client daemon choses the most optimal rules. The client requests servers to use (parts of) their offers by sending a Request message:

```
Request ::= clientID
            serverID
            Rules*
```

Since a client can decide to request all, none or a subset of the offered Rules, the Rules in the Request message are a subset of the Offer from that Server. A client can request non-coinciding subsets from multiple servers. If the request is valid (i.e. a subset from the original proposal), the server acknowledges the Request by returning an identical Acknowledgment message:

```
Acknowledgment ::= clientID
                   serverID
                   Rules*
```

The messages are encapsulated within Interests by using a base name of `ccnx:/local/dhcngp` appended with:

1. a name component containing the clientID; this name component is redundant to the clientID in the request, though appeared to be very useful during debugging
2. a name component containing the serverID; idem to the clientID though filled with the clientID, the serverID can be filled with a value of `_null`[10] to indicate a broadcast to any server for the Discovery message
3. a name component with an encoded representation of the message.

Since the daemon is written in Java, the messages are encoded from memory into a byte-string by using the Java Serializable interface [35]. The Serializable interface allows programs to exchange objects across different Java virtual machines by offering functions to serialize and deserialize an object to and from a byte-string containing the values of the object's properties. As long as Java is used as a basis for the Dynamic Host Configuration and Name Generation Protocol, the Java Serializable interface is a fast and efficient way of encoding the objects into transferable byte-strings. Whenever the need to support other languages arises

[10]The CCNx daemon was very reluctant to forward Interests in which a name component whose string values equaled "null" or contained "broadcast", possibly we were hitting on another bug or discovering some undocumented future features.

we need to implement a programming language independent encoding scheme such as the ccnb encoding discussed in section 2-2-6. Since the goal of this implementation is to verify its functional behaviour we have chosen to encode the messages using the Java Serializable interface. For a productional, possibly further enhanced, version of the protocol we propose to use the ccnb encoding scheme since it is both programming language independent and designed to efficiently encode objects into byte-strings.

3-5 Enabling data access

The protocol description above allows for clients to dynamically generate names and share information using those names. However, the client still needs other ways of information discovery or dynamic configuration to access content. Therefore, we have enhanced the protocol to make information discovery without name generation possible. In order to do so the aggregated name is made optional and when nullified indicates it is unusable for name aggregation (which means the name cannot be used to share data) but can be used as a forwarding rule towards the described name.

Clients choose appropriate forwarding rules based on route cost and path and propagate these decisions to further attached nodes in the same way any other path-vector routing protocol would. This enables end-users to dynamically access content without the need of applying network topology discovery mechanisms themselves, or manually configuring network settings. The unique point in this proposal is that it works configuration-less towards end-users.

3-6 Implementation

The implementation of the protocol is named CCNx-DHCNG which is short for CCNx Dynamic Host Configuration and Name Generation Protocol. The source of the prototype can be downloaded from [3] in order to enable others to investigate the subject of configuration-free topology discovery in NDN.

For general end-users the daemon is completely configuration-free. The daemon iterates through all possible interfaces and multicasts Interests to find nodes it can connect to. For nodes that are entry-points or default gateways, which are generally provided and preconfigured or maintained by ISPs, there is a configuration file *config.properties* (see listing 3.1) that needs to be configured.

The file its syntax conforms to the common Java's Properties [62] file syntax. Each entry is given a subsequent number starting from 0 in order to differentiate the properties that are given to each entry. The entries contain 3 properties:

- The CCNx name of the entry-point. Which can either be the name of your ISP-connection (`ccnx:/isp.net/alice` in the case of dynamic aggregation for information sharing), or the name of a forwarding rule that needs to be propagated (`ccnx:/` for a default gateway rule).
- The initial cost of the connection which defaults to 0. Similar to a distance vector protocol, at every link deeper into the network the cost of the rule increases. In the

case of ISP-multihoming multiple gateways can propagate a gateway rule. The client chooses the closest router for each entry-point based on the cost and path[11]. Being able to set the initial cost of a gateway enables administrators to express preference for a primary or secondary link.

- A boolean indicating whether name aggregation by clients is allowed, to prevent misconfiguration this property defaults to be turned off.

The following example shows a configuration file in which the publicly accessible name `ccnx:/isp.net/alice` may be used for name aggregation and a gateway route is proposed by the border router between the local network and the ISP.

Listing 3.1: Example configuration file with arbitrarily chosen costs

```
1 entry.0 = ccnx:/isp.net/alice
2 entry.0.cost = 10
3 entry.0.aggregate = true
4
5 entry.1 = ccnx:/
6 entry.1.cost = 20
7 entry.1.aggregate = false
```

Appendix D-1 shows an additional screen capture of a DHCNGP client starting up and negotiating network configuration parameters with an already configured peer.

3-7 Future work

One of the architectural options that have to be considered while further enhancing the daemons, is the fact that links may break or be added dynamically. Therefore, a mechanism has to be included that monitors link states and detects changes in links. At the detection of a new or broken link, a change in entry-point and at timely schedules, recomputation of the shortest paths have to occur. Computation of new shortest paths can lead to a different set of dynamically generated names which also need to recurse into the connected clients.

When using a mapping service to assign a registered user-based name to your dynamically generated location-dependent names, this mapping service needs to be notified of the update. We will examine the problems of mapping in the following chapter.

3-8 Conclusion

Where previously designed information discovery mechanisms already allow end-users to access content, this chapter proposes a mechanism which enables end-users to dynamically *access and share* content on a NDN. Using the restrictions invoked by naming conventions

[11]The possibility to choose both and employ a strategy as discussed in section 2-2-7 over the multipath routes might need further research.

and the lessons learned from earlier developed topology discovery, information discovery and dynamic configuration mechanisms, we have been able to specify a mechanism and protocol for dynamic configuration of end-users. With our publicly available prototype [3] end-users are able to access the network and are given a dynamically generated globally unique name under which they can share their content. The dynamic configuration and information discovery occurs within the boundaries of local networks possibly connected by multiple ISPs, without compromising the information discovery within the ASes of those ISPs. The key point is that there is no need for end-user manual network configuration; all local configuration can be dynamically generated from the settings of the ISP-maintained or preconfigured entry-nodes giving access to a greater network.

Chapter 4

Mapping

One of the topics opened by the idea of location-dependent CCNx names is the fine line between names describing information and names describing locations. Where in chapter 3 mapping is only applied by users who wish to use a registered name but only have a dynamically generated location-based name, the principle of mapping can be applied by all users. Until now, CCNx has left the decision between routing on information describing names or using a possible mapping system to first translate information describing names to location aggregated names open for research [64].

One of the major advantages of using location-dependent names mapped to content names is the decreased size of the routing complexity since each AS can reserve a limited number of first name components and use a geographical distribution of the subnames of these name prefixes to define regions and connections within that region. As an example, an NDN connection to the Delft University of Technology supplied by SURFnet could be named `/surf/netherlands/zh/tudelft`. Again, the name has no meaning to the network as long as it aggregates back to the first name component describing the ISP who can forward the subnames to the right nodes. Since the complexity of the routing problem is at most upper bounded to the size of the routing table, the routing problem in a location-dependent naming scheme is upper bounded by, or proportional to the number of ISPs or ASes connected to the network.

The number of ASes on June 4th 2012 equaled 41.313 announcing 414.483 IPv4 prefixes [26], while halfway 2010 already 196.3 million domain names (excluding NDN routeable subdomain names and subdirectories) [60] are reported. Given the fact that the global routing table size is already growing into problematic sizes [57], an increase with a factor of 500 if we were to base routing directly on registered domain names[1,2] bounds complexity problems to occur. A mechanism mapping domain-like names to a location to a path[3] may solve the complexity problem by splitting it in multiple, less complex, steps.

[1] Ignoring subdomains and subnames, which make the level of complexity even higher.

[2] Consider that prefix-matching on human readable encoded strings by itself is already more complex than a static 4-byte or 16-byte value as in IPv4 and IPv6.

[3] As proposed by the Location Identifier Separation Protocol [30] for IP.

In the following paragraphs we will discuss related work concerning mapping of identities, either in the form of content or entities, to locations.

4-1 Related Work

4-1-1 DNS

One of the most frequently used mapping systems in today's Internet is the Domain Name System [52]. It is used to translate domain names, which regularly refer to an organization or person one wants to connect to or needs a service from, to locations (mostly IP addresses[4] or other host names). DNS can, among others, be used to request the following for a given domain, host name or authority:

- Locations of web servers (both the IPv4 A-record, the IPv6 AAAA-record as well as canonical names denoted by CNAME records).
- Locations of mail servers (MX records).
- Organization preferred spam settings (described in the SPF records).
- As well provide generic information using the less specific SRV- and TXT-records[5].

The service works in a distributed fashion which means that any DNS server responsible for a top-level domain, domain or subdomain can refer to delegates when a client requests information about a hostname subzoning the (top-level/sub) domain it is responsible for. This has resulted in a system in which (1) 13 *root-servers* have root lists referring to (2) servers responsible for sets of top-level domains (.com, .net, .org, etc.) which again refer the requester to a (3) DNS server responsible for the domain. This server can either give the answer itself, or refer to a delegate which is responsible for the described subdomain. The last process repeats itself until a server answers with the requested information. Figure 4-1 shows an example of such a recursive lookup for the domain-name `www.nas.ewi.tudelft.nl`. When a server gets a request it is not authoritative for, it will refer the client to the first set of root servers, in order to have it recurse from there. This system in which DNS hierarchically serves requests or refers to servers serving more fine grained domain names leads to a distributed system in which many servers have little specific knowledge though always can refer to a server containing more fine grained information. Since the upper root- and top-level domain servers have such a great set of child nodes, as shown in the tree in figure 4-2, a regular domain lookup can be done within a nearly constant work and time complexity.

The authenticity of a received mapping is important due to the fact that a mapping containing false information may lead to security breaches. For example, when the mapping system's security is infringed, the entity gaining access over the mapping system may redirect traffic

[4] Which, to be more precise, refer to interfaces of hosts, not to locations or hosts themselves.

[5] Being the most generic types of records, we might use these to support our proposal in section 4-2. SRV-records are commonly used by, among others, Microsoft Windows Active Directory to distribute the knowledge of available Active Directory servers. TXT-records are, among others, used by Google to identify website administrators and have them configure website specific search engine preferences. Due to the generic ground of the records, they can be used for application specific services without fundamentally changing DNS.

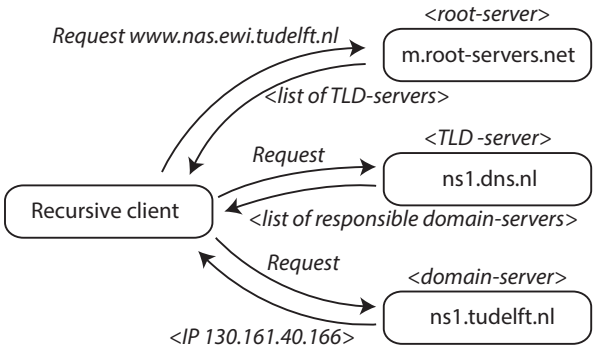


Figure 4-1: A graphical representation of a recursive DNS lookup for the name `www.nas.ewi.tudelft.nl`. First, the client queries any root DNS server, in this case `M.ROOT-SERVERS.NET`, for the domain which returns a list of DNS server responsible for the Top Level Domain (TLD) `.nl`. Next, the client queries one of the TLD servers for the name. Again, the TLD servers will not have the answer but will recurse to the name server responsible for `tudelft.nl`. Finally, the client queries the domain server (although it could have been possible that it had to recurse even further) and receives a response stating an IP address mapped to the name.

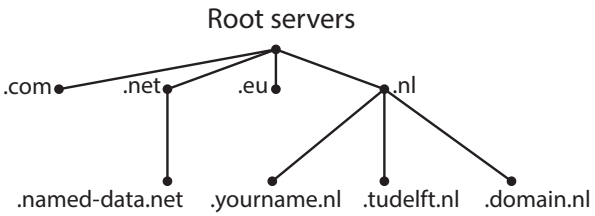


Figure 4-2: A graphical representation of a part of the DNS tree build by the referrals from root-servers to the TLD-servers of the different top-level domain extension to the domain servers of the domains themselves.

by returning incorrect mappings and setting up man-in-the-middle attacks. Added security can be delivered by using DNSsec [4], which adds a set of security extensions to DNS enabling clients to check origin authenticity of received mappings.

4-1-2 Location Identifier Separation Protocol

A recent protocol in which many research has been put concerning the separation between identifiers regarding to persons or institutions opposed to identifiers strictly referring to interfaces is called the Location Identifier Separation Protocol (LISP) [30]. The protocol offers a mechanism to identify sites (as opposed to organizations in NDN) independent from the ISPs or IP addresses by which they are connected. When a user in a network tries to connect to a site it will address the site by its *Endpoint Identifier* (EID) and rely on the Ingress Tunnel Router (ITR, basically the border router between the local LISP domain and an IP only domain) of its own site to:

1. Lookup the EID of the receiving site to its IPv4 or IPv6 *Routing Locator* (RLOC, the IP address of a border router).

2. Encapsulate the packet in an IPv4 or IPv6 packet destined for the RLOC of the receivers *Egress Tunnel Router* (EGR, which is the receiver its border router), fill the source address with its own RLOC and transmit the packet.

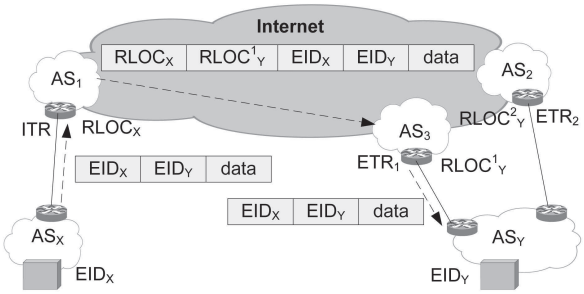


Figure 4-3: Graphical representation of a packet sent from site EIDx to site EIDy. EIDx's ITR with IP address RLOCx encapsulates the packet in a regular IP packet and sends it to one of the ETRs of site EIDy by setting the destination to one of the IP address RLOCy. Image has been taken from [45].

The ETR at the receiving site unpacks the encapsulated packet and forwards it to the appropriate site or node identified by the EID. The outer header of the encapsulated packet contains the RLOCs of the encapsulating ITR and decapsulating ETR, while the inner header contains the source address (possibly a LISP EID) of the requesting client and the EID of the receiving site [50]. Figure 4-3 shows an overview of how packets sent from within the LISP address space can travel to another LISP network by tunneling over regular IP using encapsulating and unpacking ITRs and ETRs.

Supplementary to the previously described data plane, LISP relies on a mapping system called the control plane to translate the EIDs to RLOCs, which primarily is *the same type of process we will need to map user-defined names to location-based names*. Due to the open implementation of mapping for LISP, many different mapping techniques have been researched and engineered. We will discuss the mapping techniques that currently are the most active under research and available as implementations [40][32][48][33]. In the following paragraphs we will discuss these implementations.

LISP+ALT

The *LISP+ALT* technology is the currently most employed implementation of mapping in LISP [32]. It consists of an overlay network of GRE tunnels connecting peer LISP networks together. Over the overlay network runs an eBGP instance propagating EID prefixes. The overlay (alternative) network consists strictly out of EID addressing and is therefore fully LISP routeable. The alternative network is then used to send Map-Requests to EIDs to request their EID-to-RLOC mappings and the requests are routed accordingly over the network. When an ETR receives such a Map-Request it responds with a Map-Reply containing the mapping and further communication between ITR and ETR can occur over the regular IP network. Advantages of the overlay networks are that one uses existing, compatible technologies such as GRE tunnels and BGP omitting expensive router software upgrades to reach the goal of global mapping. Disadvantages, however, are that the alternative overlay network probably is not using shortest paths for the spanning tree and tunnels may intersect frequently leading to inefficiencies. The overlay network also poses a great effort in configuration by multiple parties when an institution wants to join the overlay network, since it takes at least two parties to set up a tunnel.

LISP-TREE

LISP-TREE [40] is a mapping technique based on DNS. The proposers have chosen to trust the decade-long experience that industry has built up maintaining the DNS system. Each *LISP-TREE Server* (LTS), a DNS server equipped for serving EID to RLOC mappings, is responsible for a set of prefixes of EID addresses. LTSes can either serve the mapping for an EID themselves, recurse to known servers responsible for subsets or refer to these servers in the same way DNS usually offers mappings for domains and subdomains. An LTS only answers EIDs for whom no other known servers have a longer matching prefix. Therefore, it is possible that a client might need to recurse multiple LTSes before receiving an authoritative answer.

Recently, another approach similar to LISP-TREE called *LISP-DDT* [33] has been proposed. It replaces the LISP-TREE DNS system with application specific protocol modifications to address problems with caching, encoding the IP-like EID prefixes and negative answers. This indicates that the DNS implementation of LISP-TREE contained strong building principles, though needs LISP specific adjustments.

LISP-DHT

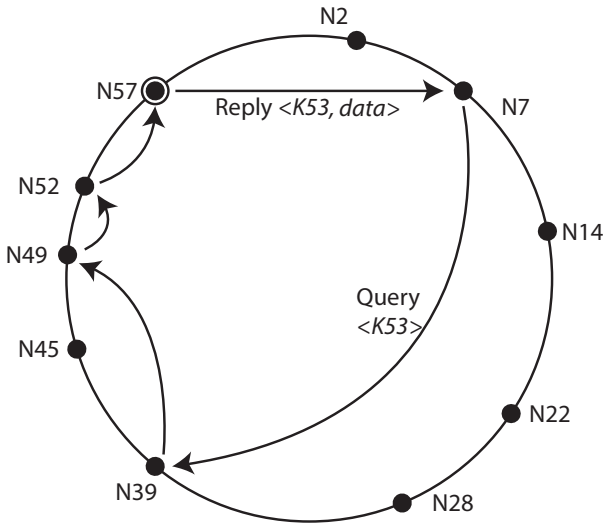
The last mapping technique to discuss is *LISP-DHT* [48]. LISP-DHT uses a distributed hash table approach to find mappings of EIDs to RLOCs by implementing Chord [58]. Chord is a protocol developed for peer-to-peer applications to quickly connect peers in a ring (ordered by their hashed node identifier) and map keys onto nodes in order to find them quickly. Since each node n, where n is its hashed identifier, has a so-called finger table which for each i-th entry contains the address of successor $n + 2^{i-1}$ and the (possibly hashed) keys of mappings are stored by increasing order on the nodes in a ring, a key can always be found within a time complexity of $O(logN)$. A node is responsible for delivering possible key-to-value mappings for keys whose hash is equal or smaller than its own, but greater than its predecessor's hashed identifier. Therefore, a node responsible for a key is called the key's successor, the node's hashed identifier is larger or equal to the mapping's identifier.

A node responsible for a key is found by traversing the chain of finger tables, each time selecting the entry whose identifier is closest preceding the queried identifier. Once the predecessor of the responsible node is found[6], the query is passed to its first successor (the responsible node) who replies directly to the original requester. Figure 4-4 shows an example of such a lookup.

For a node to join and access the ring network, it will need to know at least one node to find the place of its hashed identifier in the ring and negotiate with its successor and predecessor to change their pointers and update the finger tables of all other nodes. Authenticity of returned mappings will need to be checked by ways of signing the mappings by a central authority, though this issue still needs to be addressed.

A great advantage of LISP-DHT is that one can find a key-to-value association in a completely decentralized environment within a very limited time complexity ($O(logN)$). A disadvantage

[6] Which is trivially identified since its identifier is smaller than the queried identifier and its first successor's identifier is larger than the queried identifier



FingerTable N7	
N7+1	N14
N7+2	N14
N7+4	N14
N7+8	N22
N7+16	N28
N7+32	N39

Finger Table N39	
N39+1	N45
N39+2	N45
N39+4	N45
N39+8	N49
N39+16	N57
N39+32	N7

Figure 4-4: A graphical representation of a typical Chord lookup where the node with hashed identifier 7 is querying a lookup for data which is mapped under the hashed identifier 53. The image shows the highest predecessors chosen by nodes 7 and 39 and their respective finger tables. Node 52 is the highest predecessor for the key which makes its successor 57 responsible to reply with a set of data for the request.

of LISP-DHT is that it suggests to make the node identifiers equal to the highest address from the prefix they serve; when a node serves multiple non-adjacent prefixes it needs to run multiple independent instances of Chord nodes which can be placed non-adjacently in the ring. Another disadvantage posed by LISP-DHT is the fact that there is no central control over who enters the Chord-ring and claims to have any mappings for a specific EID. This poses security risks which means that mappings need to be checked using for example a centralized signing authority. The work and time complexity of $O(logN)$ is very low for a network where no central mapping authority is available, though the complexity may still be too large to support a network as large as the Internet.

Conclusion

Other LISP mapping techniques are disregarded in this thesis due to the fact that they either

- require all ITRs to download a complete mapping table when any change occurs [44],
- do not seem to be under serious active research (anymore) [7]
- or have many similarities to the mapping systems described above making it unnecessary to discuss them as well.

This section has discussed the Location Identifier Separation Protocol and 3 major types of mappings available for the protocol. All mapping types have their own advantages and disadvantages regarding scalability, configuration and security though we think it is safe to say the lowest work and time complexity is offered by DNS which can deliver mapping information within a nearly constant time. In the next section we will propose a mixture of the above mapping schemes to be applied in Named Data Networking.

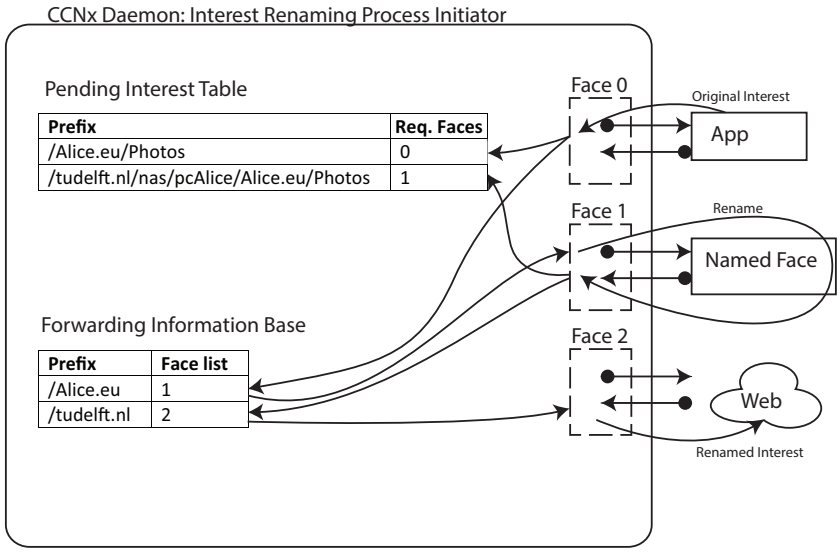


Figure 4-5: Interest Renaming Mechanism Initiator

4-2 Proposal

We propose to store name-to-location mappings for NDN, almost equal to the LISP-TREE solution, in a DNS solution. A big difference between our solution and the solution for LISP is that NDN users can benefit from the NDN itself to request mappings from DNS.

If DNS is enhanced by adding a new type of record, or reusing one of the generalized service (SRV) or text (TXT) descriptions, which could point domains towards NDN names in the same way we now point domains to IP addresses, we could use the DNS system to map the first-name components of unknown routes to a location-based aggregated name. Since there is no difference between a registered and location base name, registered names may be mapped to other registered names and this process might need to occur multiple times before a name for which its global route is known is revealed.

CCNx currently supports routes by pointing names towards nearby hosts in the form of TCP and UDP quadruples. Even though the philosophy is that NDN should be routeable over any type of network, including Ethernet, Bluetooth, IP, etc., currently only TCP and UDP encapsulated connections are implemented. Support for NDN encapsulated requests needs to be added in order for CCNx to work with mapped names.

When an incoming Interest has no prefix-matching forwarding rule in the Forwarding Information Base, CCNx can do a DNS lookup [12] [39] for the first-name component to find information to create a new forwarding rule for that domain. This behaviour conforms to the behaviour described in section 4-3. Our proposal contains the possibility for DNS lookups to also be able to return a location-based NDN name where the generator of the information hosts the original information. For example, if Alice were to share her photos using her personal domain name `ccnx:/Alice.eu/Photos`, a node without forwarding rules prefix-matching that name will do a DNS lookup for `Alice.eu` once an Interest comes in from a face (face 0 in figure 4-5) and may receive information claiming that the domain is hosted at `ccnx:/tudelft.nl/nas/pcAlice`. The CCNx daemon now needs to create a *named face* (face

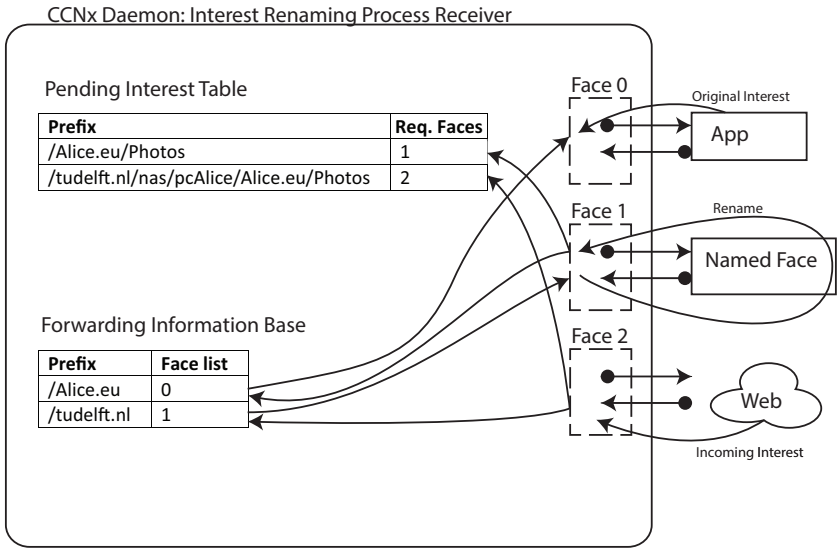


Figure 4-6: Interest Renaming Mechanism Receiver

1 in figure 4-5) configured to rename Interests to the location-based name `ccnx:/tudelft.nl/nas/pcAlice` and add a forwarding rule for `ccnx:/Alice.eu` pointing towards that face.

As shown in figure 4-5, the CCNx daemon forwards the Interest from face 0 to the named face 1, denoting the forwarded Interest originated from face 0 in the Pending Interest Table (PIT). The named face will translate the incoming Interest by prepending the location-based name (resulting in the name `ccnx:/tudelft.nl/nas/pcAlice/Alice.eu/Photos`) and present the renamed Interest to the CCNx daemon. The CCNx daemon forwards the renamed Interest from face 1 to face 2 (according to the longest prefix-match of the name in the FIB) and denotes the forwarded Interest originated from named-face 1. The translated name contains the location-based name plus the user-registered name the original user requested, since the original domain of the user-registered name is still included it is possible for a location-based name to host multiple user-registered names.

When a ContentObject satisfying the translated name returns, it prefix matches the PIT entry originating from named-face 1 and the ContentObject is offered to the named face. The named face 1 now translates the name of the ContentObject back to the original name requested. It does this by removing its own name from the front of the name and then returns the altered ContentObject to the CCNx daemon. The name of the altered ContentObject prefix matches the PIT entry originating from face 0 and the CCNx daemon will forward the ContentObject accordingly.

Figure 4-6 shows the process at the receiving node translating the Interest back to the original user-space name and forwards it to the application connected. Although the examples show the applications being directly connected to the node on which the renaming occurs, since a face can be any type of interface, application or network it is also possible to connect networks in which the user-registered name is routeable.

Note that if a client has a gateway forwarding rule (`ccnx:/`), all Interests not prefix-matching longer named forwarding rules will be forwarded to the face denoted in that rule. The possible DNS lookup will then be done by a router of the ISP which has no default route, though who

will know routes to other ASes by means of topology information discovery. The behaviour that a user-registered name is routeable within a limited domain until it reaches a border to the geographically named NDN network, shares many similar properties with the EID-routeable networks in LISP.

4-2-1 Signing and Encapsulation

One of the key aspects CCNx uses to guarantee the authenticity of received information is that each ContentObject has to be cryptographically signed in order to be accepted by a requesting party [20]. The disadvantage of the signing mechanism used in conjunction with the name-rewriting rules from above is the fact that the signature is calculated over the complete ContentObject including its name. Therefore rewriting the name renders the signature to become invalid. The party renaming the ContentObject could resign the new ContentObject, though since the packet is not signed by the original generator the receiver will not be able to review its authenticity.

The previous paragraph suggested a mechanism by merely renaming the user-based names to location-based names and vice versa, though this renders the signing mechanism to be superfluous since we change a piece of the information that is signed. The generator knows that the Interest it receives is translated from a user name to a location-based name, since the Interest it receives contains its location-based name. The previous paragraph suggested that the generator maintained a similar mechanism to translate the location-based name back to the user name and forwards this Interest to the face generating the content.

In order to ensure the authenticity of the signing process the returned ContentObject, still containing the original name, should not be resigned after renaming to the location-based name. *On-the-wire*, the returned ContentObject needs to have the location-based name in order to be successfully prefix matched to the according PIT entries. Once the ContentObject is renamed back to its original user-based name, the signature renders to be valid.

Although the user application receiving the ContentObject can now successfully verify the authenticity of the ContentObject, intermediate nodes are also allowed (however not obliged) to check authenticity of ContentObjects. Verifying the authenticity of a ContentObject while it is renamed results in the ContentObject to be dropped. Instead of merely renaming the ContentObject we suggest encapsulating the original signed ContentObject, without altering it in any way, in a new ContentObject which is given the renamed name and signed accordingly. When the encapsulated ContentObject reaches the node which renamed the original Interest, this node will unpack the original ContentObject from the encapsulating ContentObject and forward it accordingly.

Figures 4-7 and 4-8 show a graphical overview of the mechanisms needed when encapsulation, renaming and decapsulating of ContentObjects occurs. Figure 4-9 shows an example of encapsulating a user named ContentObject into a geographically named ContentObject.

4-2-2 Strategy

As discussed in section 2-2-7, the NDN strategy layer can employ heuristics to exploit multi-path routes based on previously received service. Since it is possible that content is accessible

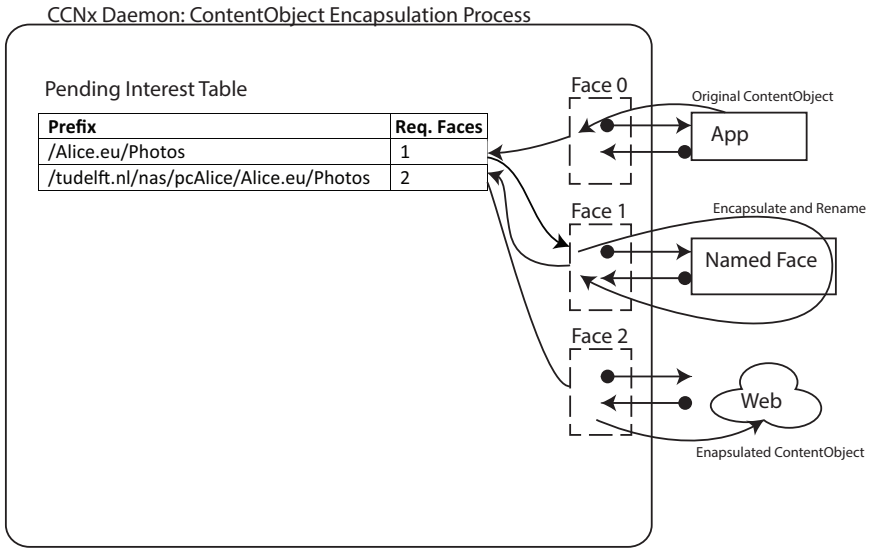


Figure 4-7: Content Object Encapsulation Mechanism

by multiple geographically assigned names (due to multihoming or geographic distribution of servers) we need ways to also employ these heuristics when a single user-registered name is accessible via multiple geographically assigned names.

In order to employ strategy heuristics over multiple paths, the CCNx daemon needs multiple forwarding entries in its FIB for which it can measure and steer the flow of Interests and ContentObjects. Therefore, when multiple mappings exist for a user-registered name, the strategy layer can be easily enabled by adding a named face for each geographic mapping between the user-registered name and its geographic name. For example, if Alice was to share her photos not only via her computer at the Delft University of Technology but also uses her home Internet connection, the user registered name `ccnx:/Alice.eu` not only maps to `ccnx:/tudelft.nl/nas/pcAlice` but also to `ccnx:/isp.net/alice`. For both geographically based names a named face will be created resulting in two forwarding entries, one to each named face, in the FIB between which the strategy layer heuristics can take place.

4-3 DNS over NDN

When DNS servers are not authoritative for a requested zone they can either redirect the querying client to another set of DNS servers, or recursively resolve the next servers themselves and return the answer. Since this can lead to an overload of identical requests, many ISPs deploy caching DNS servers to which all customers query their requests. The caching server resolves all requests recursively, returns it to the client and caches the answer for a given period of time in order to decrease the load on the network by many simultaneous DNS requests.

In addition to using DNS as a mapping service, we propose to request the DNS queries themselves over NDN giving an even larger efficiency over caching servers. A DNS request is also a request for specific content and is, therefore, suitable to be queried over an NDN.

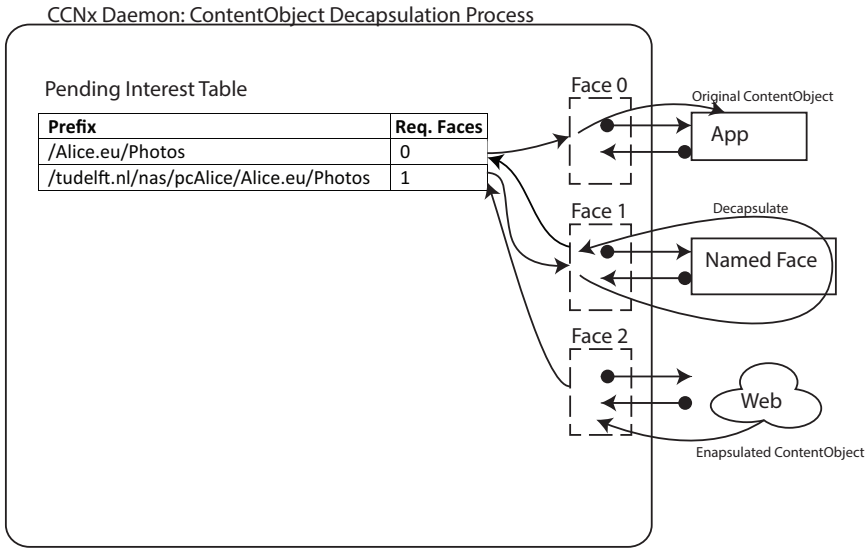


Figure 4-8: Content Object Decapsulation Mechanism

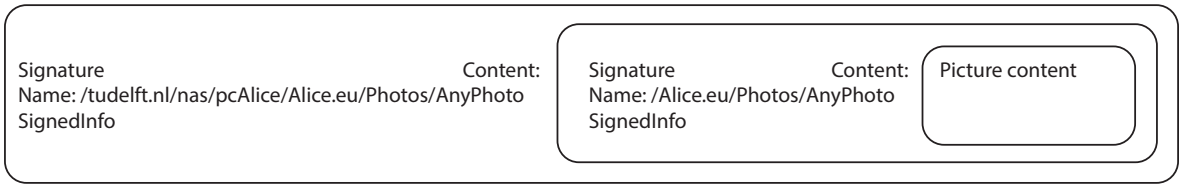


Figure 4-9: Encapsulation of a ContentObject with a user registered name into a geographically named ContentObject.

In order to make DNS suitable for NDN, we need to define an application specific name-scheme which describes the request adding more fine-grained information in each name component. The name for such an Interest should exist out of:

- the authority hosting the DNS server,
- the name one wishes to query,
- (optionally) the type of record one is interested in (e.g. A-, AAAA-, MX- or NS-records).

If we were to be interested in a (currently non-existent) NDN-record for the domain `AliceAndBob.eu`, in order to map this user-space name to a geographic name, we can now query the root-servers, TLD-servers and authoritative server via NDN using the following names:

- An Interest for the name `ccnx:/k.root-servers.org/dns/tudelft.nl/NDN` - in which `k.root-servers.org` can be any of the 13 root server - returning a list of TLD-servers for the `.eu` TLD.
- An Interest for the name `ccnx:/j.tld-server.org/dns/tudelft.nl/NDN` - in which `j.tld-server.org` can be any server from the returned TLD-servers - returning a list of authoritative servers for the domain `tudelft.nl`.

- An Interest for the name ccnx:/authoritative-server.net/dns/tudelft.nl/NDN possibly returning an authoritative answer or returning another set of servers to further recurse on.

One of the advantages of the system mentioned above, is the fact that answers to common DNS queries will be cached in the NDN network leading to a better availability of the information and a decreased load on the authoritative DNS servers. A negative side effect (which also exists in regular DNS over IP) is the fact that when the returned server to recurse to is a non-geographical name the DNS client also needs to make DNS requests for that name. This problem can be solved by adding so-called *glue records* [42] stating both the registered and the geographical name of the referred DNS servers in a single reply. In the case the name of the next DNS server is a subname of the requested name, the glue record is obligatory for the DNS system to function.

4-4 Conclusion

In this chapter we have presented and discussed two popular mechanisms to dynamically map (optionally user-readable) application names or identifiers to locations. We have proposed a mechanism to map user- or application-assigned names to geographically based names using the popular DNS system in a LISP-style manner. This mechanism offers the users of an NDN to use user-registered domain names, while aggregated location-based names keep global routing tables small and scalable. An extra advantage is offered when DNS requests themselves are requested and answered over an NDN, since this also is a one-to-many distribution of mainly identical information.

Chapter 5

Dynamic Tunnel Discovery

The currently running transition phase from IPv4 to IPv6 teaches us that not all routers (be it globally, within an AS, or even within a corporate network) can be upgraded at once.

This results in networks in which islands (groups of upgraded nodes) will form that are able to communicate via NDN internally, though not with other groups since the groups are not connected by series of nodes that all speak NDN. It is also possible that two nodes close to each other (in terms of links) are only connected by NDN enabled links via a large detour forming peninsulas which penalize the efficiency.

A possible solution to connect these islands or shortcut those peninsulas is to cross NDN incompatible nodes by setting up IP-tunnels. This is done by encapsulating NDN messages in IP, TCP or UDP and sending them over the NDN incompatible links to a node on another island which can unpack the carrying datagram and process the unpacked NDN message.

IPv6 knows several mechanisms to tunnel across IPv4 networks in order to obtain IPv6 connectivity between IPv6-islands during a transitioning phase. Unfortunately, all of these mechanisms require either

- manual configuration,
- static mapping between an IPv4 and IPv6 address-space,
- or a main, assumed that there is one, IPv6 network to connect to.

Since manual configuration is not plausible for a fast global implementation of NDN, NDN does not know address spaces mappable to IP and no main NDN network has formed yet, these mechanisms do not fit a fast global implementation of NDN.

Given a not fully NDN enabled network in which NDN-enabled nodes exist, we need to find NDN enabled paths between all NDN-enabled pairs of nodes, omitting NDN incompatible nodes and dynamically crossing NDN incompatible paths using IP tunnels efficiently chosen by a routing algorithm.

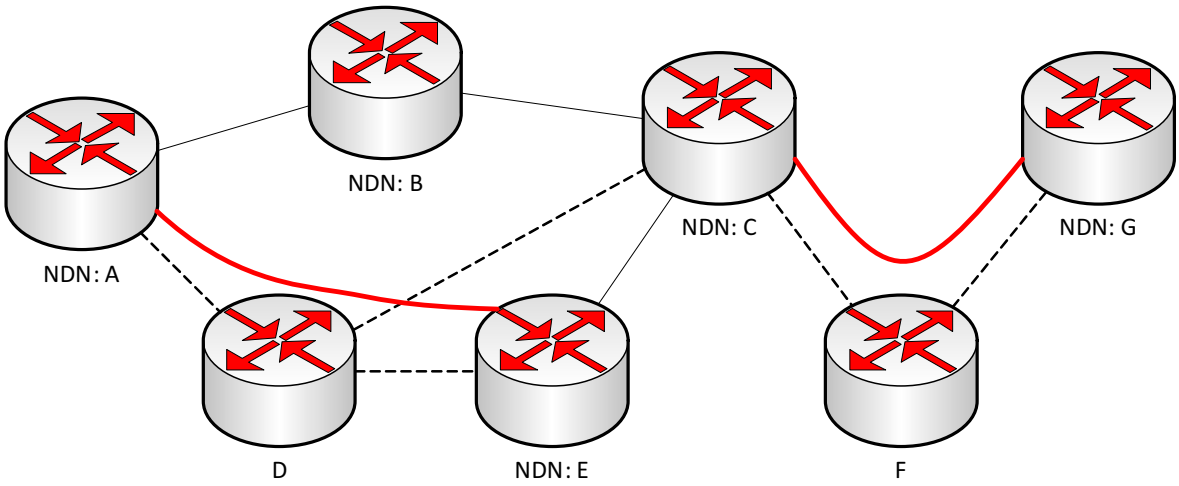


Figure 5-1: Sample partially NDN enabled network

In order to compute all NDN enabled paths, additions to routing discovery mechanisms are required to propagate knowledge about the ability of nodes to perform NDN. In turn, this knowledge has to be used in shortest path calculations to determine and create the most efficient paths automatically.

In figure 5-1, for example, in which nodes A, B, C, E and G are NDN enabled and nodes D and F are not, we need ways to dynamically connect the NDN island of node G using a tunnel. Though in the given example it is evident that the network needs an IP-tunnel from node C to G to cross node F, in larger networks decisions have to be made about multiple paths consisting out of multiple possible tunnels. In many IPv6 transitioning mechanisms the cost of a tunnel is considered to be 1 link when used in routing discovery mechanisms[1]. We propose to value a tunnel proportional to the cost of the underlying links to make the routing discovery mechanism running on top calculate forwarding rules more accurate. Next, local network policies may prefer the peninsula formed from node A via nodes B and C to node E to be shortcut by an IP-tunnel from A to E crossing node D. Our proposal offers calculation mechanisms in which shortcut tunnels can be preferred over native detours.

5-1 Related Work

5-1-1 OSPF

In IP topology discovery using OSPF, all available links between nodes are shared across all participating routers within an area. The OSPF daemons running on these routers create a graph from all shared links and use a shortest paths algorithm such as Dijkstra's algorithm or the Bellman-Ford algorithm to calculate shortest paths between nodes. The resulting shortest path routing table is then used to translate the IP subnets attached to those nodes into a OSPF network routing table which, in turn, is used to construct the IP forwarding rules.

[1]The tunnel is often connected as a virtual interface which makes it look like a directly connected IP/Ethernet interface.

5-1-2 OSPFN

In OSPFN, an addition to OSPF, opaque link state options (LSAs) are added to the OSPF IP topology discovery process when nodes are designated to generate information for a certain NDN name. The shared CCN name is denoted in the LSA and shared across all OSPF routers within the area. The same OSPF router and network routing tables are used to create the NDN forwarding rules.

OSPFN reads the nodes which share data for certain names and uses the shortest path calculation already done to create the shortest path routing table to forward NDN Interests in the right direction. This mechanism works very efficient, the same shortest path calculation used for IP is also used for NDN. Unfortunately, the algorithm fails when not all subsequent nodes are NDN enabled, i.e. some nodes can only forward IP packets.

When altering the OSPFN experiments from section 3-1-3 by disabling the CCNx and OSPFN daemon of a randomly chosen node (thereby making it IP-only) the NDN routing information will travel through the disabled node to the other nodes by means of the OSPF Opaque LSAs, though the other nodes will not recalculate forwarding rules to omit the disabled node. As a matter of fact, the NDN compatible nodes assume that all nodes within the OSPF group are NDN compatible. They will forward NDN packets to NDN incompatible nodes lying on the original shortest path. NDN incompatible nodes will discard incoming Interests as they cannot process them, which results in broken paths.

When one considers that the default CCNx application already has the possibility to let NDN packets cross multiple (NDN disabled) IP routers using IP encapsulated tunnels and it is possible to propagate NDN compatibility using OSPF's opaque LSAs, it becomes clear that this combination can be used optimally when a routing algorithm is designed that dynamically finds the most cost-effective mixture of NDN forwarding rules using both direct links and IP-tunnels across NDN incompatible paths.

5-2 Proposal

The basics of our proposal rely on the following properties. Due to the high presence of IP routers, we assume all NDN enabled nodes can reach all other NDN enabled nodes via either direct links, IPv4 or IPv6. We assume all NDN nodes can encapsulate and unpack NDN packets in IP packets, resulting in IP-encapsulated tunnels[2]. This means that each shortest path in IP from any NDN enabled nodes A to B is a possible IP encapsulated NDN tunnel.

Considering a graph in which nodes exist which are possibly NDN enabled and are connected by links, a link is NDN enabled when both nodes connected by the link are NDN enabled. A virtual link, derived from an IP encapsulated tunnel, is any shortest path between two NDN enabled nodes that is no direct NDN enabled link itself. Since the two NDN enabled nodes can reach each other using IP over the shortest path, an IP encapsulated tunnel can be set up between the two. With this information we can create a second graph containing all NDN enabled nodes, NDN enabled links and virtual links (in the form of link represented tunnels) giving each virtual link a cost proportional to the cost of the underlying IP path. In short,

[2]Which is actually the case in the NDN implementation CCNx

from the graph in figure 5-2a representing a regular adjacency matrix, figure 5-2b shows all possible shortest paths between nodes. Each of these shortest paths is a possible NDN tunnel between two NDN enabled nodes.

First, the red coloured tunnels from figure 5-2b need to be removed, since we can only set up tunnels between NDN enabled nodes. Next, we need to filter out the orange coloured tunnels whose path is crossing one or more other NDN enabled nodes. Such a tunnel would omit the NDN ability of the node being crossed, it would merely forward the IP packets encapsulating the NDN packet to the next hop instead of processing the encapsulated Interest or ContentObject. When a tunnel A to C crosses an NDN enabled node B there are also two tunnels A to B and B to C which share the same underlying links, though whose path does benefit from the NDN options of node B. Our algorithm should decide to use previously described tunnels A to B and B to C over a direct tunnel from A to C, since the probability $\Pr[cacheHit]$ of a cache hit on a node decreases the average path cost $c(A \to C) = c(A \to B) + c(B \to C)$ to $c(A \to B) + (1 - \Pr[cacheHit]) * c(B \to C)$.

The decision of a path using tunnels from A to B and B to C over a path using a tunnel from A to C can be guaranteed by either

- removing virtual links passing other NDN enabled nodes from the second graph; which we will do in our initial algorithm proposal,
- or by preferring short links or penalizing long links by varying the cost of a tunnel not only by the underlying link costs but also its length; which we will also use in our extended proposal.

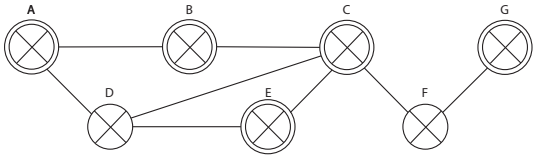
Each shortest path from the resulting graph in figure 5-2c is a valid NDN link or tunnel. Since it only contains NDN enabled nodes[3] and their NDN (virtual) links, the graph represents a virtual NDN adjacency matrix. Using an all-pairs shortest path algorithm[4], based on the cumulative cost of underlying links, we compute the shortest NDN paths and their respective forwarding rules between all NDN enabled nodes containing both direct links and virtual links to connect all NDN enabled nodes. The result of this calculation is shown in figure 5-2d, figure 5-3 shows the adjacency and forwarding matrices used for the computation of the NDN forwarding rules in respect to the graphs of figure 5-2.

Roughly speaking the proposed algorithm consists out of 3 steps:

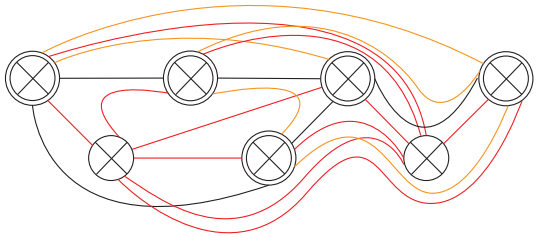
1. Generating an IP forwarding matrix from an adjacency matrix using an all-pairs shortest path calculation.
2. Filter out unsuitable links and calculate tunnel costs proportional to the underlying link costs.
3. Use another all-pairs shortest path calculation to compute NDN forwarding rules.

This proposal solves the need to dynamically connect NDN islands and shortcut large NDN detours by using IP encapsulated tunnels. In the initial algorithm, the NDN paths always

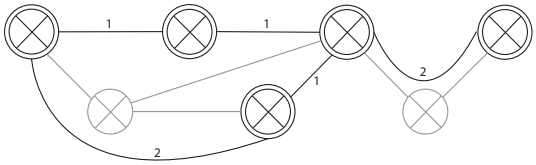
[3]Even though the NDN incompatible nodes still exist to support the IP encapsulated tunnels.
[4]Such as a Floyd-Warshall algorithm.



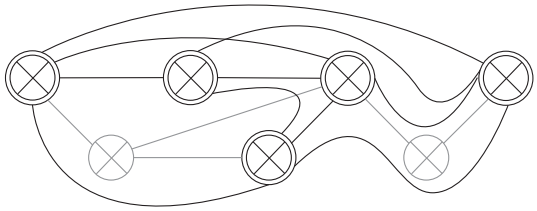
(a) A graph representing a partially NDN enabled network its nodes and adjacencies. Encircled nodes are considered NDN compatible, while others are not.



(b) A graph representing the IP forwarding matrix generated from figure 5-2a. Red coloured paths need to be removed due to NDN incompatible end nodes, orange coloured paths need to be removed since they cross NDN compatible nodes disabling the crossed nodes to process the NDN packets.



(c) The resulting NDN adjacency graph showing direct links, virtual links and their respective cost. Grey links will not be considered as they are NDN incompatible, they are, however, used to forward IP encapsulated NDN packets.



(d) A graph representing the resulting NDN paths and respective forwarding rules.

Figure 5-2: The graph representations of the original set of adjacencies of the network, its IP paths, its NDN adjacencies and its NDN paths.

From \ To	A	B	C	D	E	F	G
A	*0*	1		1			
B	1	*0*	1				
C		1	*0*	1	1	1	
D	1		1	*0*	1		
E			1	1	*0*		
F			1			*0*	1
G						1	*0*

(e) The adjacency matrix of figure 5-2a

From \ To	A	B	C	D	E	F	G
A	*0 -> A*	1 -> B	2 -> B	1 -> D	2 -> D	3 -> B	4 -> B
B	1 -> A	*0 -> B*	1 -> C	2 -> A	2 -> C	2 -> C	3 -> C
C	2 -> B	1 -> B	*0 -> C*	1 -> D	1 -> E	1 -> F	2 -> F
D	1 -> A	2 -> C	1 -> C	*0 -> D*	1 -> E	2 -> C	3 -> C
E	2 -> D	2 -> C	1 -> C	1 -> D	*0 -> E*	2 -> C	3 -> C
F	3 -> C	2 -> C	1 -> C	2 -> C	2 -> C	*0 -> F*	1 -> G
G	4 -> F	3 -> F	2 -> f	3 -> F	3 -> F	1 -> F	*0 -> G*

(f) The forwarding matrix of figure 5-2a as represented in figure 5-2b. Each entry exists out of a shortest path cumulative cost and the next-hop node.

From \ To	A	B	C	D	E	F	G
A	*0*	1			2		
B	1	*0*	1				
C		1	*0*		1		2
D							
E	2		1		*0*		
F							
G			2				*0*

(g) The NDN adjacency matrix of figure 5-2c

From \ To	A	B	C	D	E	F	G
A	*0 -> A*	1 -> B	2 -> B		2 -> E		4 -> B
B	1 -> A	*0 -> B*	1 -> C		2 -> C		3 -> G
C	2 -> B	1 -> B	*0 -> C*		1 -> E		2 -> G
D							
E	2 -> A	2 -> C	1 -> C		*0 -> E*		3 -> C
F							
G	4 -> C	3 -> C	2 -> C		3 -> C		*0 -> G*

(h) The NDN forwarding matrix of figure 5-2c as represented in figure 5-2d. Each entry exists out of a shortest path cumulative cost and the next-hop NDN node. In case the next-hop NDN node is not directly adjacent in figure 5-2a the forwarding rule implies setting up a tunnel. These forwarding rules are underlined for extra emphasis.

Figure 5-3: The adjacency and forwarding matrices in respect to figures 5-2a to 5-2d

follow the exact shortest path of the underlying links. However, in section 5-4 we will see that a carefully computed NDN detour may eventually be a more efficient route due to an average decreased path invoked by regular cache hits. In section 5-4 we will discover the possibilities of preferring many short tunnels and direct links resulting in NDN rich paths above long tunnels and NDN scarce paths.

5-3 Algorithm

The philosophy behind this algorithm is that we find suitable NDN connections (including tunnels and direct links) from all possible connections between NDN nodes. We do this by computing the three earlier described steps of:

1. Generating an IP forwarding matrix from the graph's adjacency matrix.
2. Removing unsuitable links and setting the cost of tunnels proportional to their underlying links generating an NDN adjacency matrix.
3. Generating an NDN forwarding matrix from the previously generated NDN adjacency matrix.

The three steps are clearly shown in algorithm 5.1 where (after the initialization):

1. A Floyd-Warshall [31] shortest path computation occurs to generate the IP forwarding matrix.
2. Unsuitable tunnels are removed and the cost of the remaining tunnels is determined.
3. Concluded by another Floyd-Warshall shortest path computation.

The final NDN forwarding matrix is contained in the pair of matrices $path'$ and $next'$ describing the cumulative cost and next-hops of all paths i to j.

The procedure functions most efficiently when the function *crossesNoNDN()* filters out virtual links that cross and bypass other NDN enabled nodes[5], as discussed in the previous section two tunnels travelling from nodes A to B and from B to C are preferred over a tunnel from A to C sharing the same underlying links but bypassing node B's NDN capabilities. The second run of the Floyd-Warshall all-pairs shortest path algorithm will compute the path from nodes A to C touching NDN enabled node B, guaranteeing optimal use of subsequent NDN nodes.

The function named *costFunction* returns the cumulative cost of all underlying links, this results in an algorithm in which each NDN path exactly follows the shortest path a packet would have traveled in IP. Even though the underlying path may be the shortest in the sense of total link cost, in NDN we need to consider the caching abilities of the network which decrease the actual usage of links on a path when a cache hit occurs.

In the next section we discover ways to vary the cost of tunnels in order to prefer NDN rich (cumulatively summed more expensive) paths over NDN scarce paths.

[5]The added complexity of $O(n)$ does not add to the total algorithm complexity of $O(n^3)$ since it occurs in a piece of the algorithm where the complexity equals $O(n^2)$.

Algorithm 5.1 Dynamic Tunnel Discovery

```
procedure TunnelDiscovery(G)                                   ▷ G(N, L)
    for i, j ← 1, N do                                         ▷ Initialization
        path[i][j] ← e_ij
        next[i][j] ← j
    end for
    for k, i, j ← 1, N do                                      ▷ Floyd-Warshall
        if path[i][k] + path[k][j] < path[i][j] then
            path[i][j] ← path[i][k] + path[k][j]
            next[i][j] ← k
        end if
    end for
    for i, j ← 1, N do
        if ndnEnabled(i) & ndnEnabled(j) & crossesNoNDN(i, j) then ▷ Virtual link selection
            path'[i][j] ← costFunction(i, j)                   ▷ Summed cost of links on path
            next'[i][j] ← j
        end if
    end for
    for k, i, j ← 1, N do                                      ▷ Floyd-Warshall
        if path'[i][k] + path'[k][j] < path'[i][j] then
            path'[i][j] ← path'[i][k] + path'[k][j]
            next'[i][j] ← k
        end if
    end for
    return path', next'
end procedure
```

5-4 Varying cost functions

Where the previous section computes NDN paths whose underlying links exactly match the shortest paths of the underlying network, in this section we will research possibilities to purposefully deviate from the exact shortest path. At each NDN node traversed by an Interest, the probability Pr[*cacheHit*] on a cache hit can render the rest of the path to the generator untouched because the traversed node has a copy of the requested ContentObject in its ContentStore. On average, the effectively traversed links in NDN rich paths will be lower than in equally long NDN scarce paths (i.e. paths with significantly more tunnels) due to these cache hits. Eventually, detours via NDN rich areas may be effectively cheaper in terms of average link traversal compared to one or more tunnels at most the same number of hops.

The lines from algorithm 5.1 reading *Virtual link selection* and *Summed cost of links on path* define the criteria for virtual link selection and cost function for virtual links. Since the cost function currently returns the cumulative cost of underlying links, the NDN path from any node A to any node B will always follow the same underlying links it would have done in IP, we merely replace subsequent sets of NDN incompatible links with NDN tunnels crossing those links. Although an exact shortest path calculation seems justified at first sight, we need to consider the caching possibilities of NDN and the decreased effective path usage by cache hits. For example, in figure 5-1 our initial algorithm will chose a tunnel from node A to node E with a total cost of 2 as the path from node A to E, above a NDN native path over nodes B and C with a total cost of 3 hops. A cache hit on node B, however, would decrease the effectively travelled path to 1 link instead of 3. Given a cache hit probability of Pr[*cacheHit*] the effective path cost from node A to node E over nodes B and C becomes $1+(1-\Pr[cacheHit])*(1+(1-\Pr[cacheHit]))) = 1+(1-\Pr[cacheHit])+(1-\Pr[cacheHit])^2$, solving for Pr[*cacheHit*] teaches us that an average cache hit probability of $\Pr[cacheHit] \geq 0.39$ results in an average path cost lower than 2, making the path over nodes B and C effectively cheaper than the initial tunnel over node D. A possible solution to prefer small detours over NDN richer areas is to value long tunnels to be more expensive in contrast to NDN native paths when they grow longer.

The combination of the link selection and cost function should select and define the virtual links in such a way that both

- the cost of a virtual link is proportional to the underlying path between the end nodes
- and virtual links bypassing NDN enabled nodes are either disregarded as a whole, or have a more expensive cost than the alternative smaller tunnels to have them disregarded by the second run of the Floyd-Warshall algorithm.

We will discuss and compare several combinations in which the cost function is altered to add extra functionality while maintaining the demands as stated above. In section 5-5 we will show simulation results showing which of the cost functions behave the most efficient.

5-4-1 Flat cost function

The first, most straightforward (and above all already implemented) function to calculate the cost of virtual links is to keep them identical to the cost of the underlying carrying links.

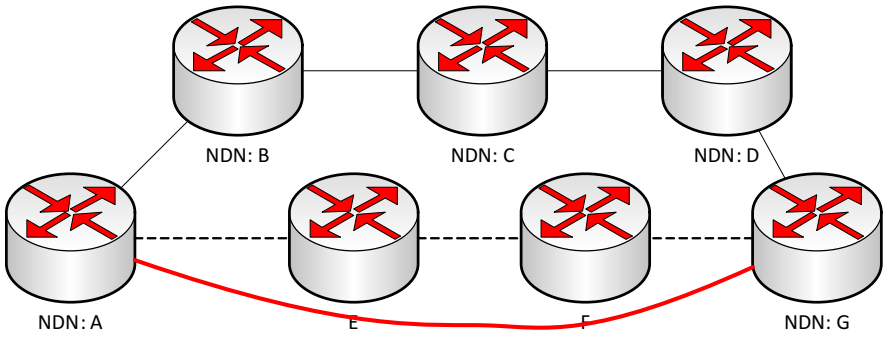


Figure 5-4: An example network of 7 nodes. The striped lines represent NDN incompatible links, while the solid lines represent NDN compatible links.

In such a case, the cost function would simply return the cost of the underlying path by returning the value of *path[i][j]*, while the virtual link selection is responsible for filtering out tunnels bypassing NDN enabled nodes.

The advantages of this function are that it guarantees an exact shortest path between nodes with a time and work complexity of $O(1)$ for the cost function and a complexity of $O(n)$ for the criteria function resulting that the algorithm complexity of $O(n^3)$ defined by the Floyd-Warshall algorithm is not exceeded.

The disadvantages of this function are that

1. the decision between an equally expensive native and tunneled path can still not be made, even when it seems obvious that an equally expensive native path may be preferred.

2. small detours allowing to touch NDN enabled nodes more frequently are not selected, since these paths will not be chosen by the second all-pairs shortest path.

I.e., when a part of a topology looks familiar to the topology shown in figure 5-4, the path to traverse from A to G will always be a tunnel from A to G over E and F, instead of traversing the native NDN path $A \rightarrow B \rightarrow C \rightarrow D$ since the lower cumulative link costs of the prior prevails. Even if there would exist a link in figure 5-4 connecting node C to G, the decision from A to G via a virtual link over E and F, or the NDN path $A \rightarrow B \rightarrow C \rightarrow G$ is untrivial since they both have the same path cost.

The two disadvantages show that in finding the optimal NDN path in a topology that is only partially NDN enabled is not trivial, and may need a less straight-forward mechanism to at least prefer equally expensive NDN links above tunnels.

5-4-2 Subsequent penalizing cost functions

A disadvantage of a true shortest path calculation is the fact that possible short detours leading to a higher density of available NDN enabled nodes, as shown in figure 5-4, are excluded since they are obviously not part of the shortest path. Given a certain probability that an intermediate NDN node already has the requested information in cache, it may be

more efficient for node A to always send Interests for content to the higher NDN populated detour $A \rightarrow B \rightarrow C \rightarrow D \rightarrow G$, even if it is not a shortest path.

In order to prefer short tunnels and direct links over longer tunnels we need a way to favor short and direct links with slightly smaller costs and penalize long links with a slightly higher cost.

A plausible solution is to introduce a penalty factor $f >= 1$ with which we multiply each second and subsequent link of a tunnel. The path from $A \rightarrow B \rightarrow C \rightarrow D \rightarrow G$ in figure 5-4 would still cost 4 hops, while the virtual link from A to G would be valued to $1 + 2f$. Since the virtual link (even with equal or smaller underlying link costs) can now be valued to be more costly than the short detour based on the value of f, we enable short detours via NDN richer areas in the graph. The higher the penalty factor f is configured, the more likely detours via NDN rich environments will occur.

A positive side effect is that virtual links crossing other NDN enabled nodes (e.g. A to C, B to D and C to G) are automatically discarded by the 2nd all-pairs shortest path calculation due to fact that a tunnel cost of $e_{ab} + f * e_{bc}$ will always be more costly than the two direct links from A to B (e_{ab}) and from B to C (e_{bc}; totaling $e_{ab} + e_{bc}$). Therefore, this cost function does not require, though does allow, the virtual link selection criteria to filter out tunnels crossing other NDN nodes. Although tunnels will be valued to be more expensive than their original underlying link costs, while direct links share their cost with the underlying link, the cost related to the tunnel is still proportional (upper bounded by the multiplication of penalty factor f) to the underlying network.

An interesting property of the resulting algorithm is that links will be used at most once within a path. Given a topology as shown in figure 5-5, 3 possible tunnels going from A to B, from B to C and from A to C exist. When a user connected by node A wants information served by node C, it should instinctively always use the tunnel from A to C and not first pass node B before it goes to C since one might not want Interests to travel the link from node B to node D twice. This cost function guarantees a link shall never be traversed twice since the cost of the path $A \rightarrow C$ ($e_{ad} + f * e_{cd}$) will always be smaller than the summed cost of $A \rightarrow B$ and $B \rightarrow C (e_{ad} + f * e_{bd} + e_{bd} + f * e_{cd} = \underline{e_{ad} + f * e_{cd}} + e_{bd} + f * e_{bd})$.

5-4-3 Faster growing penalty cost function

One scenario that is excluded by the previous cost functions is the scenario (as shown in figure 5-6) in which a long tunnel from A to G has to be set up when a user connected to A wants information served by node G, while node H is closer by and potentially already has a copy of the information or could store it for later requests by other users. Since the previous algorithms either take the shortest path on the underlying links, or only allow detours when links are only used once, node H will not be used as a tunnel endpoint since it is not on the shortest path from A to G and in the case of a detour link h needs to be traveled twice (once when traveling from A to H, and once when traveling from H to G).

We can say that the previous cost functions disallow the use of a potentially interesting node while travelling from A to G. In order to allow the use of nearby NDN nodes, which actually mean temporarily exiting the shortest path to potentially return later via the same route, we need to adapt our cost function to make tunnels even more expensive while they grow longer.

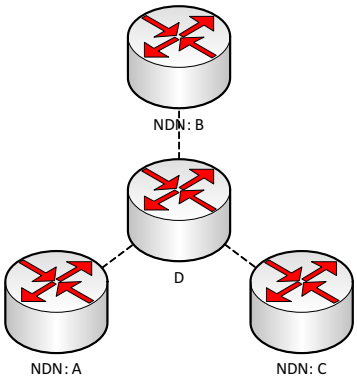


Figure 5-5: An example of a partially NDN enabled network. The center node D is NDN incompatible, forcing all other nodes to set up tunnels to each other.

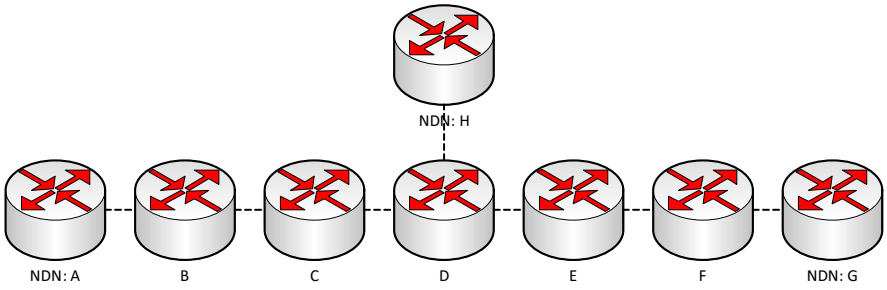


Figure 5-6: An example of a partially NDN enabled network. The image supports scenarios in which one might want to diverge from the shortest path, possibly using a link twice, in order to see if a cache entry is available.

At a cache hit ratio of $\Pr[cacheHit] \geq 0.5$ the path from nodes A to G over H becomes less expensive than the tunnel across the shortest path from nodes A to G.

In order to support these types of detours, we need one or more cost-functions whose penalty increases at each subsequent link in order to break the property from the subsequent penalty function (subsection 5-4-1) that each link within a path can only be traveled once. However, we also want the virtual cost of tunnels to be proportional to their underlying links to prevent inefficiently large detours. Initially, we can linearly increase the penalty at each traversed link with the penalty factor its delta $f' = f - 1$, being the penalizing addition of the original penalty factor. Using this function, the detour from nodes A to G via H is already taken with a penalty factor of $f \geq 1\frac{1}{3}$.

Another interesting, even faster increasing, function one can implement is to penalize each link using an increasing power of the penalty factor f. Using this function, the detour via node H is already taken with a penalty factor of $f \gtrsim 1.27$. To see the result of the difference in growth by the previous functions, we will enhance the linearly growing function with functions whose penalization grow at different rates. For a slower growth we will use a logarithmic multiplication of the penalty factor's delta f' and the link cost C, faster growth can be obtained by using a facultative or quadratic multiplication instead.

Table 5-1 describes all functions we will simulate in section 5-5. Using these simulations we will describe the effects of computing and using nearly shortest paths in combination with a caching network.

Description	Function	Complexity
Flat cost function	$\sum_{i=1}^{L} C_i$	$O(1)$*
Subsequent penalizing function	$\sum_{i=1}^{L} \begin{cases} C_i & \text{if } i = 1 \\ f * C_i & \text{if } i > 1 \end{cases}$	$O(1)$**
Linearly growing penalizing function	$\sum_{i=1}^{L} C_i * (1 + f' * (i-1))$	$O(L)$
Power-function growing penalizing function	$\sum_{i=1}^{L} C_i * f^{i-1}$	$O(L)$
Facultatively growing penalizing function	$\sum_{i=1}^{L} C_i * (1 + f' * (i-1)!)$	$O(L)$
Quadratically growing penalizing function	$\sum_{i=1}^{L} C_i * (1 + f' * (i-1)^2)$	$O(L)$
Logarithmically growing penalizing function	$\sum_{i=1}^{L} \begin{cases} C_i & \text{if } i = 1 \\ C_i * (1 + f' * log(i-1)) & \text{if } i > 1 \end{cases}$	$O(L)$

Table 5-1: Table of length penalizing cost functions that we have simulated in our experiments. The functions vary from a flat sum of all link costs C_i to more complex functions subsequently increasing link cost per link $i = 1..L$ part of the tunnel. The penalty factor $f > 1$ is network configurable, f' equals $f - 1$ being the penalizing addition of the penalty factor f. *Since the total sum of the shortest path is already known in the IP forwarding matrix, this value can also be copied. **Since the total sum of the shortest path is already known, this function can be rewritten to $FirstLinkCost + f * (OldPathCost - FirstLinkCost)$ leading to a smaller complexity.

5-5 Simulations

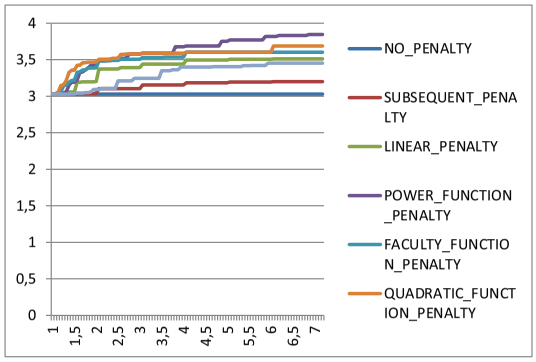
This subsection discusses simulations we have done to investigate which penalizing function, and eventually its optimal penalty factor, works best in finding efficient NDN paths. In order to do this we generate random connected graphs[6] on which we run the different penalizing functions with all penalty factors f varying from 1 (no penalty) to 7 (the point where all functions have reached their maximum) with a step size of $\frac{1}{16}$[7]. In order to value the efficiency for each generated network, we compute the average path cost $\overline{C}$ for all distinct paths from all nodes A to B in the network generated by the different runs of our algorithm. In order to get a clear estimate of the behaviour of our algorithm, we have generated 100 independent samples of both 100 node large and 200 node large graphs to run our simulations on. All networks are generated with a probability of $\frac{1}{20}$ that two nodes are connected and a probability of $\frac{1}{8}$ that any given node is NDN enabled.

Figures 5-7 and 5-8 show the average of all network average path costs $\overline{\overline{C}}$ for the generated networks. The path costs have been computed with a cache hit probability $\Pr[cacheHit]$ varying from 0 (which refers to a regular non-caching network) to $\frac{7}{8}$ with steps of $\frac{1}{8}$ in order to quickly determine the efficiency of the different cost functions in relation to the cache hit ratio of a network.

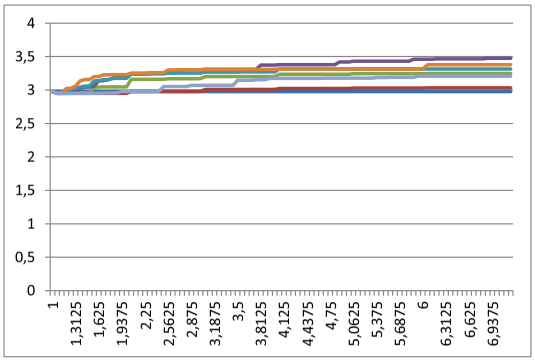
Each subfigure of figures 5-7 and 5-8 include a line of reference in the form of the average path

[6] We generate random graphs according to the Erdõs-Rényi model [29]. We ensure connectivity by regenerating until a connected graph (a graph without islands) has formed.

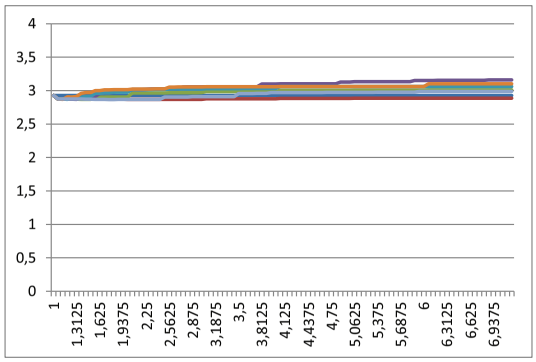
[7] Increasing with fractions of 2 helps keeping programmatic floating point approximations more accurate.



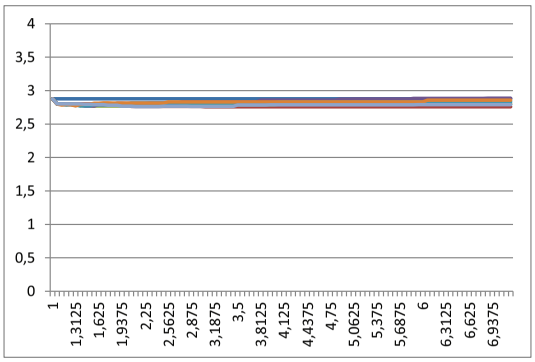
(a) $\Pr[cacheHit] = 0$



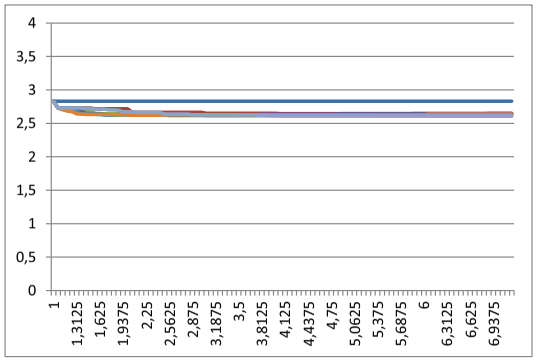
(b) $\Pr[cacheHit] = 0.125$



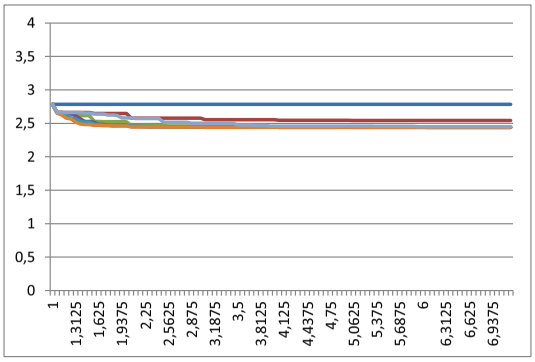
(c) $\Pr[cacheHit] = 0.25$



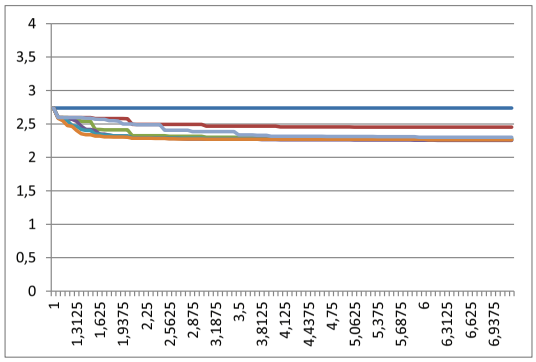
(d) $\Pr[cacheHit] = 0.375$



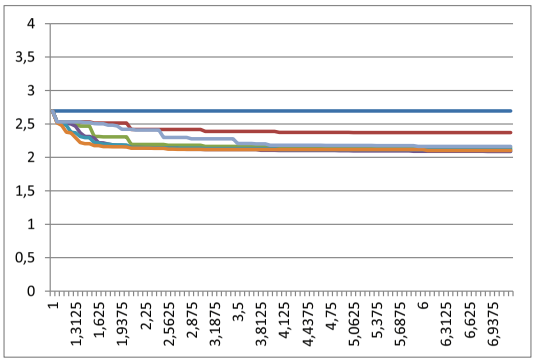
(e) $\Pr[cacheHit] = 0.5$



(f) $\Pr[cacheHit] = 0.625$

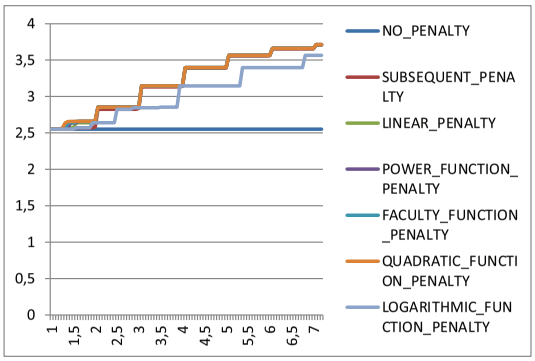


(g) $\Pr[cacheHit] = 0.75$

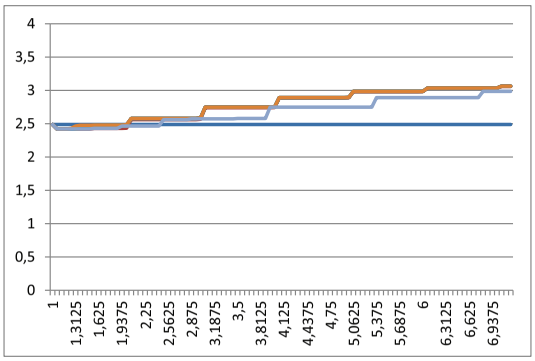


(h) $\Pr[cacheHit] = 0.875$

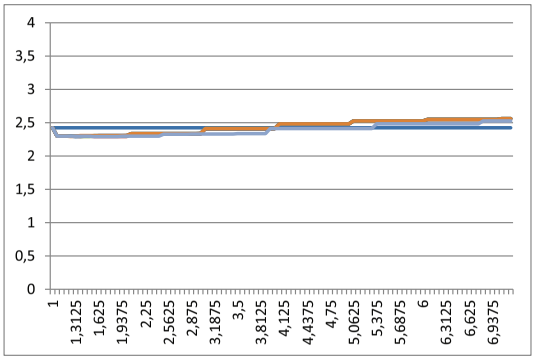
Figure 5-7: Graphs 100 nodes



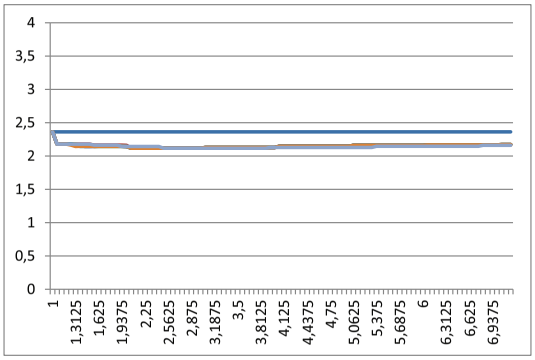
(a) $\Pr[cacheHit] = 0$



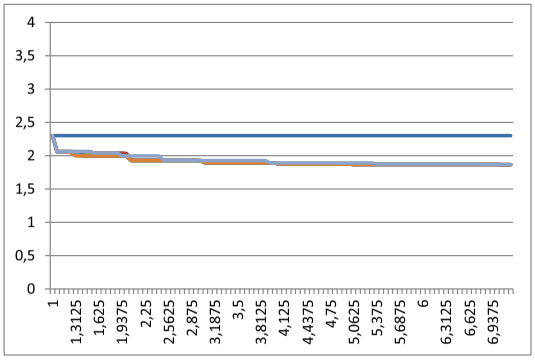
(b) $\Pr[cacheHit] = 0.125$



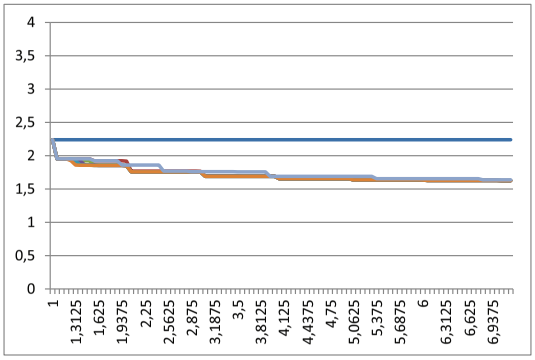
(c) $\Pr[cacheHit] = 0.25$



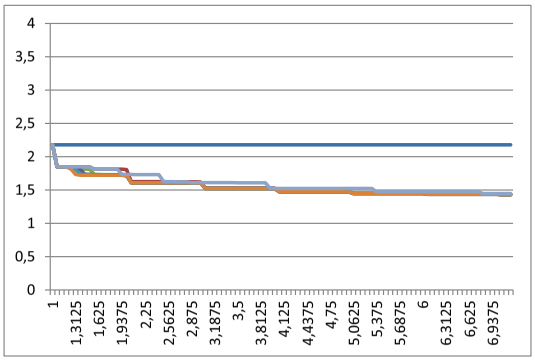
(d) $\Pr[cacheHit] = 0.375$



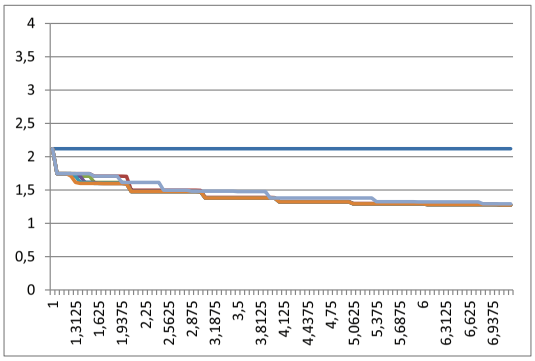
(e) $\Pr[cacheHit] = 0.5$



(f) $\Pr[cacheHit] = 0.625$



(g) $\Pr[cacheHit] = 0.75$



(h) $\Pr[cacheHit] = 0.875$

Figure 5-8: Graphs 200 nodes

cost $\overline{\overline{C}}$ when no penalizing function would have been applied. Using this line of reference we can see that the average path cost $\overline{\overline{C}}$ decreases from 3.03 to 2.69 hops in and from 2.55 to 2.12 respectively for the 100 node large and 200 node large generated networks in figures 5-7 and 5-8 due to the increased probability of a cache hit. In both figures, it is clear that at a low cache hit ratio the detours taken by the penalizing functions form an overhead in terms of increased average path costs. At higher cache hit ratios, starting around $\Pr[cacheHit] = 0.4$ and $\Pr[cacheHit] = 0.25$ respectively, *the detours result in lower average path cost.*

Looking more carefully at the charts, the characteristics of the different penalizations become more clear. Where the subsequently penalizing function shows different characteristics in the smaller network of figure 5-7, the faster growing functions' characteristics look very similar in both figures. The only difference between the faster growing function lies in the speed (compared to the penalty factor) in which they grow to their limit. Since the penalty factor can be defined as precise as a network administrator needs it to be, the speed at which the results of the penalizing function grow is no argument favoring any of the faster growing functions. Therefore, we can say the decision between any of the faster growing penalizing functions opposed to another is equal. The characteristics of the (slower growing) subsequently penalizing function, however, appear to be different from the faster growing functions in the smaller network of figure 5-7. The function's limit generally lies closer to the line of reference. This is due to the subsequent penalizing function allowing links to be traveled at most once[8] along any path, thereby disallowing possible detours.

The graphs from figure 5-8 show that taking detours over NDN richer paths can give a decrease in average traveled paths varying from 8.51% in subfigure 5-8d to 39.00% in subfigure 5-8h. Our algorithm offers an undeniable increase in efficiency in NDNs with a sufficient cache hit ratio. Network administrators can choose to implement the subsequent penalizing function when they want to profit from NDN richer paths but do not want links to be crossed twice. One of the faster growing functions can be chosen to optimally benefit from NDN richer areas.

5-6 Future work

An interesting topic connected to the idea of finding detours which are effectively more efficient due to regular cache hits, is the usage of the actually measured cache hit ratio of nodes in the path selection mechanism. This involves taking a node's truly delivered caching service into account when the path selection takes place. Although this behaviour may initially be implemented fairly easy by altering the second Floyd-Warshall all-pairs shortest path computation in algorithm 5.1, taking a node's truly experienced cache hit ratio has a reimbursing side effect on the formation of the paths in the network which requires further research.

In order for a node to deliver a high cache hit ratio, it needs to meet 2 requirements:

1. A cache large enough to store the copies of frequently requested data.
2. A fair amount of requests and deliveries to initially fill its cache and keep it filled to enable it to deliver content from cache regularly.

[8] As discussed in section 5-4-2

Whenever a node does not receive many Interests it will also have a less rich populated cache, resulting in a lower cache hit ratio. The lower cache hit ratio results in nodes choosing other (cache hit richer) paths which results in the node still having a poorly filled cache with equal cache hit ratio. Nodes that already have richly populated caches with high cache hit ratios will keep receiving a large amount of Interests because other nodes prefer their paths due to the high cache hit ratio. This behavior may result in a deadlock where new nodes, with the potential to become very efficient cache hit performers, will not get a fair chance to play their cards because earlier settled nodes with high cache hit ratios refrain them from doing so.

Since using the actually delivered cache hit ratio of nodes within an information discovery process effects the cache hit ratio itself, one can see that before using parametrization of historically delivered service within path selection computation[9] we first need to research the implications of these side effects and whether they can be overcome.

Taking the truly delivered service of nodes into account in the path selection process overlaps with the philosophy of the NDN Strategy layer[10]. This gives another argument for future research on selecting paths by the quality of the service they deliver.

Although this proposal has been designed specifically for dynamically implementing NDN in partially enabled networks without user configuration of tunnels or other transition mechanisms, it is very suitable to be used without major change by other types of application specific networks such as LISP [30], sensor networks and smart grids in order to dynamically connect over an overlay network using dynamically chosen tunnels.

5-7 Conclusion

In this chapter we proposed a solution to dynamically connect NDN islands and detours by creating an overlay network of all possible IP encapsulated NDN tunnels. Our algorithm computes optimal forwarding rules based on all direct links and tunnels from this overlay network using an all-pairs shortest path algorithm.

For networks in which a significant probability of cache hits exist, we have extended our proposal to find detours over NDN richer areas. Our simulations show that regular cache hits by the increased number of NDN nodes on a path, result in a lower average path utilization.

Finally, future work is suggested in the field of information discovery, by explicitly preferring paths containing well performing caches even more efficient networks may be formed.

[9] However useful it seems, everyone wants to shop at the store with the best service and the lowest price

[10] Which (as discussed in section 2-2-7) also enables path selection based on previously delivered service.

Chapter 6

Conclusion

In this thesis, we investigated several problems that arise when a Named Data Network (NDN) [64] is globally implemented. In each chapter we dived into the core of one of those problems and proposed solutions to overcome them, with each proposal ultimately contributing to the foreseen global implementation of an NDN.

After discussing the basic foundations of NDN and related work in this research area in chapter 2, we start off in chapter 3 by solving the problem of end-user device configuration in small office and home office networks. In order for end-users to embrace the philosophy of NDN, the process of accessing and sharing information on the NDN must be simple and straightforward. As already common with DHCP [28] and IP [55], *we suggest a mechanism which dynamically configures devices without user interaction to keep the complexity towards users as low as possible.* The mechanism dynamically finds paths in local networks towards the gateways delivered by ISPs. Based on these paths, globally unique names are generated which solves an important prerequisite for content to be shared globally.

Although the names generated in chapter 3 are globally unique and can be easily used to share data, users may prefer context related names in the form of user-registered names over the longer topology related generated names. To support the translation from user-registered names to dynamically generated names, *we propose a mapping lookup mechanism based on DNS [52] and LISP [30].* Users are enabled to share content with user-registered names, while the Interests find their way using the respective aggregated dynamically generated names. The Interests and returning content still travel over an NDN network, just with a different name, maintaining the profits of NDNs invoked by caching. A proper setup of the renaming instances enables NDN implementations to benefit from multipath and multihomed connections as if it were a genuinely globally propagated name space.

While the mapping mechanism is useful for renaming on top of the dynamically configured networks, it can play an even greater role when used to reduce the size of global routing tables. Storing all user registered names in global routing tables is too complex due to the large number of individuals and institutions that may acquire registered names. Therefore, we propose to map all registered names to the topologically aggregated names and base global routing purely on the smaller set of aggregated names keeping global routing tables scalable.

Chapter 5 focuses on dynamically connecting NDN islands and shortcutting large NDN native detours using IP encapsulated tunnels. Using our proposed algorithm we generate an overlay network of tunnels and direct links and compute shortest NDN paths over that network. *We introduce a transition mechanism combined with a topology discovery algorithm connecting partially NDN enabled networks.* Since cache hits reduce traveled path costs in NDN rich paths, in the second half of the chapter we enhance the algorithm by allowing to take detours over NDN richer paths. Simulations show these detours over NDN richer paths result in effectively reduced path costs in randomly generated networks with high cache hit ratios. The proposal needs full knowledge of the network and can, therefore, be implemented by enhancing a link-state routing protocol such as OSPF [53].

Conclusive, we have solved the following items in this thesis:

- Propose a system for dynamic host configuration, local network information discovery and the generation of globally unique names to quickly enable end-users to access and share data across the NDN.
- Enabling the use of user registered names in conjunction with dynamically generated names, as well as reducing the complexity of global routing tables by mapping registered names to topology aggregated names.
- Solving routing discovery and employing a transitioning mechanism by proposing a single algorithm solving both.

6-1 Future work

In this thesis we have already proposed solutions for dynamic configuration, information discovery and transitioning suitable for use within local and medium sized networks. The network size supported by the proposal of chapter 5 is determined by the fact that it needs full knowledge of the local network in order to fulfill its duty. Though suitable for networks of Autonomous Systems, such as ISPs, the complexity of the algorithm ($O(n^3)$, equal to Floyd-Warshall's all-pairs shortest path algorithm) prevents it from being implemented globally. Ultimately, we also need a mechanism for global information discovery.

Where the proposal from chapter 5 can be easily implemented (in both fully native and partially NDN enabled networks) using the opaque options from OSPF, a similar approach may be taken for inter-AS topology discovery. Future work should aim at extending the Border Gateway Protocol (BGP) [49] in such a way that NDN enabled clients can use it to propagate their availability and compute NDN paths to each other. If NDN incompatible parties ignore and silently forward the NDN specific information, NDN enabled nodes can use this information to create an overlay network of NDN similar to the overlay network proposed in chapter 5. Using this information nodes can compute which tunnels they should create to cross NDN incompatible ASes and set up forwarding rules accordingly.

Completing the inter-AS information discovery means all layers (local networks, intra-AS networks and inter-AS networks) of information discovery have been solved, which means a great part of research and engineering necessary for the global implementation of NDN will be finished.

Appendix A

Files CCNx-DHCP Experiments

A-1 Single server single client

A-1-1 Single Server Single Client - Node 1

ubuntu ccnd[2021] local port 9695 api 4001 start 1324631846.519598 now 1324633619.424284

Content items: 34 accessioned, 34 stored, 33 stale, 0 sparse, 0 duplicate, 35 sent
Interests: 12 names, 0 pending, 0 propagating, 1 noted
Interest totals: 35 accepted, 0 dropped, 34 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 35
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 2 **remote:** 224.0.23.170:59695

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	42 / 21	0 / 0	0 / 0
face: 6	5 / 42	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147481877
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147481877
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147481877
- ccnx:/local/dhcp **face:** 6 **flags:** 0x3 **expires:** 55
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147481937
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147481877
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147481877

A-1-2 Single Server Single client - Node 2

ubuntu ccnd[2603] local port 9695 api 4001 start 1324632190.219527 now 1324633622.216034

Content items: 7 accessioned, 7 stored, 7 stale, 0 sparse, 0 duplicate, 7 sent
Interests: 10 names, 0 pending, 0 propagating, 0 noted
Interest totals: 8 accepted, 0 dropped, 6 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 3 **remote:** 224.0.23.170:59695

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147482217
- ccnx:/ **face:** 7 **flags:** 0x3 **expires:** 2147482237
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147482217
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147482217
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147482237
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147482217
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147482217

A-1-3 Single Server Multiple Clients - Node 1

ubuntu ccnd[2352] local port 9695 api 4001 start 1324634977.962926 now 1324635140.730244

Content items: 6 accessioned, 6 stored, 5 stale, 0 sparse, 1 duplicate, 7 sent
Interests: 12 names, 0 pending, 0 propagating, 1 noted
Interest totals: 8 accepted, 0 dropped, 6 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 7
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 3 **remote:** 224.0.23.170:59695

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483487
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483487
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483487
- ccnx:/local/dhcp **face:** 6 **flags:** 0x3 **expires:** 50
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147483572
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483487
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483487

A-1-4 Single Server Multiple Clients - Node 2

ubuntu ccnd[2623] local port 9695 api 4001 start 1324634975.315669 now 1324635097.578644

Content items: 6 accessioned, 6 stored, 4 stale, 0 sparse, 0 duplicate, 8 sent
Interests: 11 names, 0 pending, 0 propagating, 0 noted
Interest totals: 8 accepted, 0 dropped, 6 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 2 **remote:** 224.0.23.170:59695
- **face:** 8 **flags:** 0x40412 **pending:** 0 **remote:** 172.19.5.1:59695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483527
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483527
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483527
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147483612
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483527
- ccnx:/myNamespace **face:** 8 **flags:** 0x3 **expires:** 2147483612
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483527

A-1-5 Single Server Multiple Clients - Node 3

ubuntu ccnd[2605] local port 9695 api 4001 start 1324634980.555202 now 1324635104.972941

Content items: 6 accessioned, 6 stored, 4 stale, 0 sparse, 0 duplicate, 7 sent
Interests: 12 names, 0 pending, 0 propagating, 6 noted
Interest totals: 7 accepted, 0 dropped, 6 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 1 **remote:** 224.0.23.170:59695
- **face:** 8 **flags:** 0x40412 **pending:** 0 **remote:** 172.19.5.1:59695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483527
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483527
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483527
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147483637
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483527
- ccnx:/myNamespace **face:** 8 **flags:** 0x3 **expires:** 2147483637
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483527

A-2 Multiple server

A-2-1 Multiple Server - Node 1

ubuntu ccnd[2549] local port 9695 api 4001 start 1324635923.293676 now 1324635995.835746

Content items: 6 accessioned, 6 stored, 3 stale, 0 sparse, 0 duplicate, 6 sent
Interests: 10 names, 0 pending, 0 propagating, 1 noted
Interest totals: 6 accepted, 0 dropped, 5 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 6
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 2 **remote:** 224.0.23.170:59695

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483577
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483577
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483577
- ccnx:/local/dhcp **face:** 6 **flags:** 0x3 **expires:** 25
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147483612
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483577
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483577

A-2-2 Multiple Server - Node 2

ubuntu ccnd[2684] local port 9695 api 4001 start 1324635920.874792 now 1324635987.730655

Content items: 6 accessioned, 6 stored, 3 stale, 0 sparse, 0 duplicate, 6 sent
Interests: 10 names, 0 pending, 0 propagating, 1 noted
Interest totals: 6 accepted, 0 dropped, 5 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 6
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 2 **remote:** 224.0.23.170:59695

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	7 / 0	0 / 0	0 / 0
face: 7	7 / 7	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483582
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483582
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483582
- ccnx:/local/dhcp **face:** 6 **flags:** 0x3 **expires:** 25
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147483612
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483582
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483582

A-2-3 Multiple Server - Node 3 - 1st Result

ubuntu ccnd[2623] local port 9695 api 4001 start 1324635960.959503 now 1324635991.307776

Content items: 7 accessioned, 7 stored, 0 stale, 0 sparse, 0 duplicate, 7 sent
Interests: 18 names, 0 pending, 0 propagating, 6 noted
Interest totals: 7 accepted, 0 dropped, 6 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 2 **remote:** 224.0.23.170:59695
- **face:** 8 **flags:** 0x40412 **pending:** 0 **remote:** 172.19.5.2:59695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483617
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483617
- ccnx:/mySecondNamespace **face:** 8 **flags:** 0x3 **expires:** 2147483637
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483617
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147483637
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483617
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483617

A-2-4 Multiple Server - Node 3 - 2nd Result

ubuntu ccnd[2655] local port 9695 api 4001 start 1324636327.681609 now 1324636340.987579

Content items: 7 accessioned, 7 stored, 0 stale, 0 sparse, 0 duplicate, 7 sent
Interests: 18 names, 0 pending, 0 propagating, 6 noted
Interest totals: 7 accepted, 0 dropped, 6 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 2 **remote:** 224.0.23.170:59695
- **face:** 8 **flags:** 0x40412 **pending:** 0 **activity:** 1 **remote:** 172.19.5.1:59695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	43 / 21	0 / 0	0 / 0
face: 7	7 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483637
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483637
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483637
- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147483642
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483637
- ccnx:/myFirstNamespace **face:** 8 **flags:** 0x3 **expires:** 2147483642
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483637

A-3 Multiple Interfaces

A-3-1 Multiple Interfaces - Server Node 3

```
./ccndhcpserver
1325065590.057118 ccnd[1913]: accepted client fd=8 id=6
1325065590.083627 ccnd[1913]: at ccn_sockcreate.c:191
1325065590.083647 ccnd[1913]: at ccn_sockcreate.c:205
1325065590.083654 ccnd[1913]: at ccn_sockcreate.c:216
1325065590.083661 ccnd[1913]: at ccn_sockcreate.c:224
1325065590.083669 ccnd[1913]: at ccn_sockcreate.c:231
1325065590.083676 ccnd[1913]: at ccn_sockcreate.c:245
1325065590.083706 ccnd[1913]: at ccn_sockcreate.c:254
1325065590.083714 ccnd[1913]: at ccn_sockcreate.c:259
1325065590.083720 ccnd[1913]: at ccn_sockcreate.c:266
1325065590.083743 ccnd[1913]: at ccn_sockcreate.c:270
1325065590.083754 ccnd[1913]: at ccn_sockcreate.c:315
1325065590.083761 ccnd[1913]: IPv4 multicast
1325065590.083787 ccnd[1913]: setsockopt(..., IP_ADD_MEMBERSHIP, ...): No
    such device
1325065590.086998 ccndhcp[1919]:424: OnNull cleanup
ccn_client.c:0[1919] - error 0: Cannot join DHCP group.
1325065590.087784 ccnd[1913]: shutdown client fd=8 id=6
1325065590.087800 ccnd[1913]: releasing face id 6 (slot 6)

route add -net 224.0.0.0 netmask 224.0.0.0 eth3

./ccndhcpserver
1325064756.084005 ccnd[1886]: accepted client fd=8 id=6
1325064756.126891 ccnd[1886]: at ccn_sockcreate.c:191
1325064756.126910 ccnd[1886]: at ccn_sockcreate.c:205
1325064756.126917 ccnd[1886]: at ccn_sockcreate.c:216
1325064756.126923 ccnd[1886]: at ccn_sockcreate.c:224
1325064756.126931 ccnd[1886]: at ccn_sockcreate.c:231
1325064756.126937 ccnd[1886]: at ccn_sockcreate.c:245
1325064756.126968 ccnd[1886]: at ccn_sockcreate.c:254
1325064756.126976 ccnd[1886]: at ccn_sockcreate.c:259
1325064756.126981 ccnd[1886]: at ccn_sockcreate.c:266
1325064756.127005 ccnd[1886]: at ccn_sockcreate.c:270
1325064756.127016 ccnd[1886]: at ccn_sockcreate.c:315
1325064756.127022 ccnd[1886]: IPv4 multicast
1325064756.340403 ccnd[1886]: at ccn_sockcreate.c:337
1325064756.340431 ccnd[1886]: SO_RCVBUF for fd 9 is 131072
1325064756.340449 ccnd[1886]: multicast on fd=9 id=7, sending on face 7
1325065436.983643 ccnd[1886]: shutdown client fd=8 id=6
1325065436.983666 ccnd[1886]: releasing face id 6 (slot 6)

ccndstatus
ubuntu ccnd[1930] local port 9695 api 4001 start 1325065633.320681 now
    1325065676.776780
Content items: 4 accessioned, 4 stored, 3 stale, 0 sparse, 0 duplicate, 5
    sent
```

```
46 Interests: 10 names, 0 pending, 0 propagating, 0 noted
47 Interest totals: 5 accepted, 0 dropped, 4 sent, 0 stuffed
48 Faces
49  face: 0 flags: 0xc pending: 0
50  face: 1 flags: 0x400c pending: 0
51  face: 2 flags: 0x5012 pending: 0 local: 0.0.0.0:9695
52  face: 3 flags: 0x5010 pending: 0 local: 0.0.0.0:9695
53  face: 4 flags: 0x4042 pending: 0 local: [::]:9695
54  face: 5 flags: 0x4040 pending: 0 local: [::]:9695
55  face: 6 flags: 0xc pending: 0 activity: 5
56  face: 7 flags: 0x432 pending: 0 remote: 224.0.23.170:59695
57 Face Activity Rates
58                  Bytes/sec In/Out        recv data/intr sent     sent
                       data/intr recv
59  face: 0             0 / 0                       0 / 0                    0
        / 0
60  face: 6             0 / 0                       0 / 0                    0
        / 0
61  face: 7             0 / 0                       0 / 0                    0
        / 0
62 Forwarding
63  ccnx:/ccnx/ping face: 0 flags: 0x3 expires: 2147483607
64  ccnx:/%C1.M.S.neighborhood face: 0 flags: 0x3 expires: 2147483607
65  ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA
       %AC%E6i%85-%02%83g%8A face: 0 flags: 0x17 expires: 2147483607
66  ccnx:/local/dhcp face: 6 flags: 0x3 expires: 25
67  ccnx:/local/dhcp face: 7 flags: 0x3 expires: 2147483612
68  ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd face: 0 flags: 0x3 expires:
       2147483607
69  ccnx:/%C1.M.S.localhost face: 0 flags: 0x23 expires: 2147483607
```

Appendix B

OSPFN Experiments

B-1 Multipath with single name sharing

B-1-1 Multipath with single name - Node 2

ccnx2 ccnd[2067] local port 9695 api 4005 start 1326269669.838528 now 1326270252.028748

Content items: 38 accessioned, 38 stored, 35 stale, 0 sparse, 0 duplicate, 47 sent
Interests: 17 names, 0 pending, 0 propagating, 8 noted
Interest totals: 47 accepted, 0 dropped, 37 sent, 2 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 25
- **face:** 7 **flags:** 0x40412 **pending:** 0 **remote:** 10.12.12.131:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.17.131:9695 **via:** 2
- **face:** 10 **flags:** 0xc **pending:** 0 **activity:** 7
- **face:** 11 **flags:** 0x432 **pending:** 0 **activity:** 13 **remote:** 224.0.23.170:59695

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0
face: 10	0 / 0	0 / 0	0 / 0
face: 11	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/mySixthNode **face:** 8 **flags:** 0x3 **expires:** 2147483397
- ccnx:/local/dhcp **face:** 10 **flags:** 0x3 **expires:** 55
- ccnx:/local/dhcp **face:** 11 **flags:** 0x3 **expires:** 2147483582
- ccnx:/myThirdNode **face:** 7 **flags:** 0x3 **expires:** 2147483147
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483067
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483067
- ccnx:/myFifthNode **face:** 8 **flags:** 0x3 **expires:** 2147483397

B-1-2 Multipath with single name - Node 3

ccnx3 ccnd[1913] local port 9695 api 4005 start 1326269674.845872 now 1326270253.437585

Content items: 13 accessioned, 13 stored, 12 stale, 0 sparse, 0 duplicate, 19 sent
Interests: 12 names, 0 pending, 0 propagating, 0 noted
Interest totals: 19 accepted, 0 dropped, 13 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 19
- **face:** 7 **flags:** 0x40412 **pending:** 0 **remote:** 10.12.12.130:9695 **via:** 2
- **face:** 8 **flags:** 0x40412 **pending:** 0 **remote:** 10.12.13.131:9695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/mySixthNode **face:** 7 **flags:** 0x3 **expires:** 2147483397
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483072
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483072
- ccnx:/myFifthNode **face:** 8 **flags:** 0x3 **expires:** 2147483147
- ccnx:/myFourthNode **face:** 8 **flags:** 0x3 **expires:** 2147483147
- ccnx:/mySecondNode **face:** 7 **flags:** 0x3 **expires:** 2147483147
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483072
- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483072
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483072

B-1-3 Multipath with single name - Node 4

ccnx4 ccnd[1930] local port 9695 api 4005 start 1326269676.147192 now 1326270254.512105

Content items: 15 accessioned, 15 stored, 14 stale, 0 sparse, 0 duplicate, 17 sent
Interests: 12 names, 0 pending, 0 propagating, 0 noted
Interest totals: 17 accepted, 0 dropped, 14 sent, 1 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 12
- **face:** 7 **flags:** 0x40412 **pending:** 0 **remote:** 10.12.13.130:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.15.130:9695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/mySixthNode **face:** 8 **flags:** 0x3 **expires:** 2147483147
- ccnx:/myThirdNode **face:** 7 **flags:** 0x3 **expires:** 2147483147
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483072
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483072
- ccnx:/myFifthNode **face:** 8 **flags:** 0x3 **expires:** 2147483147
- ccnx:/mySecondNode **face:** 7 **flags:** 0x3 **expires:** 2147483147
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483072
- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483072
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147483072

B-1-4 Multipath with single name - Node 5

ccnx5 ccnd[1926] local port 9695 api 4005 start 1326269677.398714 now 1326270255.552877

Content items: 37 accessioned, 37 stored, 36 stale, 0 sparse, 0 duplicate, 46 sent
Interests: 14 names, 0 pending, 0 propagating, 0 noted
Interest totals: 46 accepted, 0 dropped, 39 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 19
- **face:** 7 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.15.131:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.16.131:9695 **via:** 2
- **face:** 9 **flags:** 0xc **pending:** 0 **activity:** 24

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0
face: 9	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/mySixthNode **face:** 8 **flags:** 0x3 **expires:** 2147483147
- ccnx:/myThirdNode **face:** 7 **flags:** 0x3 **expires:** 2147483147
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147483072
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147483072
- ccnx:/myFourthNode **face:** 7 **flags:** 0x3 **expires:** 2147483147
- ccnx:/myFifthNode/ping **face:** 9 **flags:** 0x3 **expires:** 35
- ccnx:/mySecondNode **face:** 8 **flags:** 0x3 **expires:** 2147483397
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147483072
- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147483072

B-1-5 Multipath with single name - Node 6

ccnx6 ccnd[1884] local port 9695 api 4005 start 1326269678.210260 now 1326270335.842730

Content items: 40 accessioned, 40 stored, 37 stale, 0 sparse, 0 duplicate, 51 sent
Interests: 17 names, 0 pending, 0 propagating, 4 noted
Interest totals: 51 accepted, 0 dropped, 40 sent, 2 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 25
- **face:** 7 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.16.130:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.17.130:9695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	31 / 16	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/myThirdNode **face:** 8 **flags:** 0x3 **expires:** 2147483312
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147482992
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147482992
- ccnx:/myFifthNode **face:** 7 **flags:** 0x3 **expires:** 2147483617
- ccnx:/myFourthNode **face:** 7 **flags:** 0x3 **expires:** 2147483067
- ccnx:/mySecondNode **face:** 8 **flags:** 0x3 **expires:** 2147483312
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147482992
- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147482992
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147482992

B-2 Multipath with multiple name sharing

B-2-1 Multipath with multiple name - Node 2

ccnx2 ccnd[2067] local port 9695 api 4005 start 1326269669.838528 now 1326278615.190375

Content items: 268 accessioned, 268 stored, 267 stale, 0 sparse, 0 duplicate, 315 sent
Interests: 18 names, 0 pending, 0 propagating, 6 noted
Interest totals: 335 accepted, 0 dropped, 277 sent, 4 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 158
- **face:** 7 **flags:** 0x20412 **pending:** 0 **activity:** 1 **remote:** 10.12.12.131:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **activity:** 1 **remote:** 10.12.17.131:9695 **via:** 2
- **face:** 11 **flags:** 0x432 **pending:** 0 **activity:** 19 **remote:** 224.0.23.170:59695

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0
face: 11	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/mySixthNode **face:** 8 **flags:** 0x3 **expires:** 2147483417
- ccnx:/local/dhcp **face:** 11 **flags:** 0x3 **expires:** 2147475217
- ccnx:/myThirdNode **face:** 7 **flags:** 0x3 **expires:** 2147483267
- ccnx:/ourRedundantName **face:** 8 **flags:** 0x3 **expires:** 2147482757
- ccnx:/ourRedundantName **face:** 7 **flags:** 0x3 **expires:** 2147482757
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147474702
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147474702
- ccnx:/myFifthNode **face:** 8 **flags:** 0x3 **expires:** 2147482757
- ccnx:/myFourthNode **face:** 7 **flags:** 0x3 **expires:** 2147482757

B-2-2 Multipath with multiple name - Node 3

ccnx3 ccnd[1913] local port 9695 api 4005 start 1326269674.845872 now 1326278617.567024

Content items: 102 accessioned, 102 stored, 101 stale, 0 sparse, 0 duplicate, 151 sent
Interests: 23 names, 0 pending, 0 propagating, 7 noted
Interest totals: 168 accepted, 0 dropped, 111 sent, 1 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 148
- **face:** 7 **flags:** 0x20412 **pending:** 0 **activity:** 1 **remote:** 10.12.12.130:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **activity:** 1 **remote:** 10.12.13.131:9695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/mySixthNode **face:** 7 **flags:** 0x3 **expires:** 2147483417
- ccnx:/ourRedundantName **face:** 8 **flags:** 0x3 **expires:** 2147482757
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147474707
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147474707
- ccnx:/myFifthNode **face:** 8 **flags:** 0x3 **expires:** 2147482757
- ccnx:/myFourthNode **face:** 8 **flags:** 0x3 **expires:** 2147482757
- ccnx:/mySecondNode **face:** 7 **flags:** 0x3 **expires:** 2147482397
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147474707
- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147474707
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147474707

B-2-3 Multipath with multiple name - Node 4

ccnx4 ccnd[1930] local port 9695 api 4005 start 1326269676.147192 now 1326278643.218972

Content items: 134 accessioned, 134 stored, 133 stale, 0 sparse, 0 duplicate, 189 sent
Interests: 16 names, 0 pending, 0 propagating, 1 noted
Interest totals: 206 accepted, 4 dropped, 147 sent, 2 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 7 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.13.130:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.15.130:9695 **via:** 2
- **face:** 10 **flags:** 0xc **pending:** 0 **activity:** 39
- **face:** 11 **flags:** 0xc **pending:** 0 **activity:** 21

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0
face: 10	0 / 0	0 / 0	0 / 0
face: 11	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ourRedundantName/ping **face:** 11 **flags:** 0x3 **expires:** 45
- ccnx:/mySixthNode **face:** 8 **flags:** 0x3 **expires:** 2147483392
- ccnx:/myThirdNode **face:** 7 **flags:** 0x3 **expires:** 2147483237
- ccnx:/ourRedundantName **face:** 8 **flags:** 0x3 **expires:** 2147482732
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147474682
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147474682
- ccnx:/myFifthNode **face:** 8 **flags:** 0x3 **expires:** 2147482732
- ccnx:/mySecondNode **face:** 7 **flags:** 0x3 **expires:** 2147482732
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147474682

B-2-4 Multipath with multiple name - Node 5

ccnx5 ccnd[1926] local port 9695 api 4005 start 1326269677.398714 now 1326278640.514956

Content items: 284 accessioned, 284 stored, 283 stale, 0 sparse, 0 duplicate, 344 sent
Interests: 14 names, 0 pending, 0 propagating, 1 noted
Interest totals: 361 accepted, 4 dropped, 298 sent, 1 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 7 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.15.131:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.16.131:9695 **via:** 2
- **face:** 10 **flags:** 0xc **pending:** 0 **activity:** 34
- **face:** 11 **flags:** 0xc **pending:** 0 **activity:** 21

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0
face: 10	0 / 0	0 / 0	0 / 0
face: 11	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/ourRedundantName/ping **face:** 11 **flags:** 0x3 **expires:** 50
- ccnx:/mySixthNode **face:** 8 **flags:** 0x3 **expires:** 2147483397
- ccnx:/myThirdNode **face:** 7 **flags:** 0x3 **expires:** 2147483242
- ccnx:/ourRedundantName **face:** 7 **flags:** 0x3 **expires:** 2147482737
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147474687
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147474687
- ccnx:/myFourthNode **face:** 7 **flags:** 0x3 **expires:** 2147482737
- ccnx:/mySecondNode **face:** 8 **flags:** 0x3 **expires:** 2147482732
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147474687

B-2-5 Multipath with multiple name - Node 6

ccnx6 ccnd[1884] local port 9695 api 4005 start 1326269678.210260 now 1326278633.296220

Content items: 128 accessioned, 128 stored, 128 stale, 0 sparse, 0 duplicate, 186 sent
Interests: 13 names, 0 pending, 0 propagating, 0 noted
Interest totals: 200 accepted, 0 dropped, 138 sent, 3 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 6 **flags:** 0xc **pending:** 0 **activity:** 154
- **face:** 7 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.16.130:9695 **via:** 2
- **face:** 8 **flags:** 0x20412 **pending:** 0 **remote:** 10.12.17.130:9695 **via:** 2

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 6	0 / 0	0 / 0	0 / 0
face: 7	0 / 0	0 / 0	0 / 0
face: 8	0 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/myThirdNode **face:** 8 **flags:** 0x3 **expires:** 2147483247
- ccnx:/ourRedundantName **face:** 7 **flags:** 0x3 **expires:** 2147482737
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147474692
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147474692
- ccnx:/myFifthNode **face:** 7 **flags:** 0x3 **expires:** 2147482742
- ccnx:/myFourthNode **face:** 7 **flags:** 0x3 **expires:** 2147482737
- ccnx:/mySecondNode **face:** 8 **flags:** 0x3 **expires:** 2147482382
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147474692
- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147474692
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147474692

Appendix C

Combined OSPFN and CCNx-DHCP Experiments

C-1 CCNx-DHCP client connected to OSPFN network

ccnx1 ccnd[1900] local port 9695 api 4005 start 1326269705.474222 now 1326278613.038448

Content items: 24 accessioned, 24 stored, 24 stale, 0 sparse, 0 duplicate, 25 sent
Interests: 18 names, 0 pending, 0 propagating, 6 noted
Interest totals: 25 accepted, 0 dropped, 24 sent, 0 stuffed

Faces

- **face:** 0 **flags:** 0xc **pending:** 0
- **face:** 1 **flags:** 0x400c **pending:** 0
- **face:** 2 **flags:** 0x5012 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 3 **flags:** 0x5010 **pending:** 0 **local:** 0.0.0.0:9695
- **face:** 4 **flags:** 0x4042 **pending:** 0 **local:** [::]:9695
- **face:** 5 **flags:** 0x4040 **pending:** 0 **local:** [::]:9695
- **face:** 7 **flags:** 0x432 **pending:** 0 **activity:** 19 **remote:** 224.0.23.170:59695

Face Activity Rates

	Bytes/sec In/Out	recv data/intr sent	sent data/intr recv
face: 0	0 / 0	0 / 0	0 / 0
face: 7	9 / 0	0 / 0	0 / 0

Forwarding

- ccnx:/local/dhcp **face:** 7 **flags:** 0x3 **expires:** 2147475237
- ccnx:/%C1.M.S.neighborhood **face:** 0 **flags:** 0x3 **expires:** 2147474742
- ccnx:/ccnx/%A5%97%CF%27%E10%0CE%21%7D%EE%C9%87W%FC%D5%FC%24w%C5%81%29%FA%AC%E6i%85-%02%83g%8A **face:** 0 **flags:** 0x17 **expires:** 2147474742
- ccnx:/ **face:** 7 **flags:** 0x3 **expires:** 2147475237
- ccnx:/%C1.M.S.localhost **face:** 0 **flags:** 0x23 **expires:** 2147474742
- ccnx:/ccnx/ping **face:** 0 **flags:** 0x3 **expires:** 2147474742
- ccnx:/%C1.M.S.localhost/%C1.M.SRV/ccnd **face:** 0 **flags:** 0x3 **expires:** 2147474742

C-2 Ping results from client

```
1  ./ccnping ccnx:/ourRedundantName
2  CCNPING ccnx:/ourRedundantName
3  content from ccnx:/ourRedundantName: random_number = 257297670  rtt =
       108.277ms body = ping ack node5
4  content from ccnx:/ourRedundantName: random_number = 1780917240
       rtt = 125.244ms body = ping ack node4
5  content from ccnx:/ourRedundantName: random_number = 101957390  rtt =
       110.694ms body = ping ack node4
6  content from ccnx:/ourRedundantName: random_number = 688053013  rtt =
       125.358ms body = ping ack node4
7  content from ccnx:/ourRedundantName: random_number = 129872471  rtt =
       125.261ms body = ping ack node4
8  content from ccnx:/ourRedundantName: random_number = 1328907276
       rtt = 125.291ms body = ping ack node4
9  content from ccnx:/ourRedundantName: random_number = 711068187  rtt =
       117.381ms body = ping ack node4
10 content from ccnx:/ourRedundantName: random_number = 1912353359
       rtt = 108.058ms body = ping ack node4
11 content from ccnx:/ourRedundantName: random_number = 1226765568
       rtt = 125.309ms body = ping ack node4
12 content from ccnx:/ourRedundantName: random_number = 35266697   rtt =
       110.104ms body = ping ack node4
13 content from ccnx:/ourRedundantName: random_number = 1995088542
       rtt = 125.405ms body = ping ack node4
14 content from ccnx:/ourRedundantName: random_number = 1912051879
       rtt = 125.261ms body = ping ack node4
15 content from ccnx:/ourRedundantName: random_number = 210346916  rtt =
       125.306ms body = ping ack node5
16 content from ccnx:/ourRedundantName: random_number = 219609515  rtt =
       140.765ms body = ping ack node5
17 content from ccnx:/ourRedundantName: random_number = 13340366   rtt =
       109.033ms body = ping ack node5
18 content from ccnx:/ourRedundantName: random_number = 87208733   rtt =
       125.047ms body = ping ack node5
19 content from ccnx:/ourRedundantName: random_number = 1242811001
       rtt = 78.593ms  body = ping ack node5
20 content from ccnx:/ourRedundantName: random_number = 1939781540
       rtt = 130.124ms body = ping ack node5
21 content from ccnx:/ourRedundantName: random_number = 1091973776
       rtt = 125.448ms body = ping ack node5
22 content from ccnx:/ourRedundantName: random_number = 1782153704
       rtt = 125.291ms body = ping ack node5
23 content from ccnx:/ourRedundantName: random_number = 626884905  rtt =
       125.467ms body = ping ack node5
24 content from ccnx:/ourRedundantName: random_number = 831496770  rtt =
       125.208ms body = ping ack node5
25 content from ccnx:/ourRedundantName: random_number = 301668207  rtt =
       125.609ms body = ping ack node5
26 content from ccnx:/ourRedundantName: random_number = 791752698  rtt =
       125.246ms body = ping ack node5
```

Appendix D

Proposal Implementation

D-1 Dynamic Host Configuration and Name Generation screen capture

A typical initialization of a client entering a network dynamically configurable using the DHCNGP. The client and server HostIDs are replaced with the human readable names ccnx1 and ccnx2. The client receives a non-aggregational gateway forwarding rule (ccnx:/) in order to access information and receives the name ccnx:/myEntrypointNode/ccnx1 which it can use to share information and further aggregate upon.

```
1  Welcome to the Dynamic Host Configuration and Name Generation daemon.
2  HostID = ccnx1
3  HostName = ccnx1
4  Creating multicast faces.
5  Sending discovery message to local subnets
6  A discovery packet will be send.
7  Sending Discover message from ccnx1 to _null
8  Receiving a message from ccnx2 to ccnx1
9  Processing the Offer message
10 Done waiting for Offer-responses, calculating preferred forwarding table
11 The following aggregationTable has formed. The usage of dynamically found
       rules is requested.
12         Entrypoint: /   Cost: 40        pathVector: /ccnx2      aggregate
               : false        Name: null
13         Entrypoint: /myEntrypointNode   Cost: 30        pathVector: /
               ccnx2      aggregate: true Name: /myEntrypointNode/ccnx1
14 Sending Request message from ccnx1 to ccnx2
15 Receiving a message from ccnx2 to ccnx1
16 Processing the Acknowledgement message
17         Creating face to host ccnx2 on 10.12.14.130
18                 Adding forwarding rule for entrypoint /
19                 Adding forwarding rule for entrypoint /myEntrypointNode
```

Bibliography

[1] Carlisle Adams and Steve Lloyd. *Understanding PKI second edition*. Pearson Education, 2002.

[2] Niels L.M. van Adrichem. *Multicast Bug Report*. Apr. 2012. URL: http://redmine.ccnx.org/issues/100045.

[3] Niels L.M. van Adrichem. *NvanAdrichem/CCNx-DHCNGP*. Mar. 2012. URL: https://github.com/NvanAdrichem/CCNx-DHCNGP.

[4] R. Arends et al. *RFC 4033 DNS Security Introduction and Requirements*. Mar. 2005. URL: http://tools.ietf.org/html/rfc4033.

[5] T. Bates et al. *RFC 2858 Multiprotocol Extension for BGP-4*. June 2000. URL: http://tools.ietf.org/html/rfc2858.

[6] L. Berger et al. *RFC 5250 The OSPF Opaque LSA Option*. July 2008. URL: http://tools.ietf.org/html/rfc5250.

[7] S. Brim et al. *LISP-CONS: A Content distribution Overlay Network Service for LISP*. Apr. 2008. URL: http://tools.ietf.org/html/draft-meyer-lisp-cons-04.

[8] R. Callon. *RFC 1198 Use of OSI IS-IS for Routing in TCP/IP and Dual Environments*. Dec. 1990. URL: http://www.rfc-editor.org/rfc/rfc1195.txt.

[9] B. Carpenter. *RFC 6343 Advisory Guidelines for 6to4 Deployment*. Aug. 2011. URL: http://tools.ietf.org/html/rfc6343.

[10] Palo Alto Research Center. *Canonical CCNx Ordering*. Apr. 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/CanonicalOrder.html.

[11] Palo Alto Research Center. *CCNx*. June 2012. URL: http://www.ccnx.org/.

[12] Palo Alto Research Center. *CCNx Basic Name Conventions*. Feb. 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/NameConventions.html.

[13] Palo Alto Research Center. *CCNx Binary Encoding (ccnb)*. Apr. 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/BinaryEncoding.html.

[14] Palo Alto Research Center. *CCNx ContentObject*. June 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/ContentObject.html.

[15] Palo Alto Research Center. *CCNx DTAG Values*. Apr. 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/DTAG.html.

[16] Palo Alto Research Center. *CCNx DTD*. Apr. 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/dtd.html.

[17] Palo Alto Research Center. *CCNx InterestMessage*. June 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/InterestMessage.html.

[18] Palo Alto Research Center. *CCNx Main Schema*. Apr. 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/xsd.html.

[19] Palo Alto Research Center. *CCNx Protocol*. Mar. 2012. URL: www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html.

[20] Palo Alto Research Center. *CCNx Signature Generation and Verification*. June 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/SignatureGeneration.html.

[21] Palo Alto Research Center. *CCNx Technical Documentation*. July 2012. URL: http://www.ccnx.org/releases/latest/doc/technical/.

[22] Palo Alto Research Center. *Content-Centric Networking in C Documentation*. July 2012. URL: http://www.ccnx.org/releases/latest/doc/ccode/html/index.html.

[23] Palo Alto Research Center. *Content-Centric Networking in Java Documentation*. July 2012. URL: http://www.ccnx.org/releases/latest/doc/javacode/html/index.html.

[24] Palo Alto Research Center. *Named Data Networking*. Nov. 2011. URL: http://www.named-data.net.

[25] Palo Alto Research Center. *Named Data Networking - Resources*. June 2012. URL: http://www.named-data.net/education.html.

[26] CIDR. *CIDR Report*. June 2012. URL: http://www.cidr-report.org/as2.0/.

[27] Christian Dannewitz and Thorsten Biermann. "Prototyping a Network of Information". In: *IEEE Local Computer Networks* 34 (2009). URL: http://www.ieeelcn.org/prior/LCN34/lcn34demos/lcn-demo2009_dannewitz.pdf.

[28] R. Droms. *RFC 2131 Dynamic Host Configuration Protocol*. Mar. 1997. URL: http://www.ietf.org/rfc/rfc2131.txt.

[29] P. Erdõs and A. Rényi. "On the evolution of random graphs". In: *Publications of the Mathematical Institute of the Hungarian Academy of Services* 5 ().

[30] D. Farinacci. *Locator/ID Separation Protocol (LISP) draft-ietf-lisp-23*. May 2012. URL: http://tools.ietf.org/html/draft-ietf-lisp-23.

[31] Robert W. Floyd. "Algorithm 97: Shortest Path". In: *Communications of the ACM* 5 (6 June 1962).

[32] V. Fuller. *LISP Alternative Topology (LISP+ALT)*. Dec. 2011. URL: http://tools.ietf.org/html/draft-ietf-lisp-alt-10.

[33] Vince Fuller and Glen Wiley. *LISP-DDT*. May 2012. URL: http://www.nanog.org/meetings/nanog55/presentations/Tuesday/Fuller.pdf.

[34] R. Gilligan and E. Nordmark. *RFC 2893 Transition Mechanisms for IPv6 Hosts and Routers*. Aug. 2000. URL: http://tools.ietf.org/html/rfc2893.

[35] Todd Greanier. *Discover the secrets of the Java Serialization API*. July 2000. URL: http : //java.sun.com/developer/technicalArticles/Programming/serialization/.

[36] Miniwatts Marketing Group. *World Internet Usage Statistics*. Aug. 2012. URL: http://www.internetworldstats.com/stats.htm.

[37] R. Hinden and S. Deering. *RFC 4291 IP Version 6 Addressing Architecture*. Feb. 2006. URL: http://tools.ietf.org/html/rfc4291.

[38] Van Jacobson et al. "Custodian-Based Information Sharing". In: *IEEE Communications Magazine* 50 (7 July 2012).

[39] Van Jacobson et al. "Networking Named Content". In: *CoNEXT 2009* (2009).

[40] Lorï£¡nd Jakab et al. "LISP-TREE: A DNS Hierarchy to Support the LISP Mapping System". In: *IEEE Journal on Selected Areas in Communications* 28.8 (Oct. 2010), pp. 1332 –1343.

[41] Paul Jakma et al. *Quagga Software Routing Suite*. 2012. URL: http://www.nongnu.org/quagga.

[42] P. Koch. *DNS Glue RR Survey and Terminology Clarification draft-koch-dns-glue-clarifications-04*. July 2012. URL: draft-koch-dns-glue-clarifications-04.

[43] Teemu Koponen et al. "A Data-Oriented (and Beyond) Network Architecture". In: *SIGCOMM* (2007).

[44] E. Lear. *NERD: A Not-so-novel EID to RLOC Database*. Apr. 2012. URL: http://tools.ietf.org/html/draft-lear-lisp-nerd-09.

[45] Hogbin Luo, Yajuan Qin, and Hongke Zhang. "A DHT-Based Identifier-to-Locator Mapping Approach for a Scalable Internet". In: *IEEE Transactions on Parallel and Distributed Systems* 20 (12 Dec. 2009).

[46] Greg Lutostanski and Beichuan Zhang. *NDN-Routing/ccnx-dhcp*. Dec. 2011. URL: https://github.com/NDN-Routing/ccnx-dhcp.

[47] G. Malkin. *RFC 1723 RIP Version 2 Carrying Additional Information*. Nov. 1994. URL: http://tools.ietf.org/html/rfc1723.

[48] Laurent Mathy and Luigi Iannone. "LISP-DHT: Towards a DHT to map identifiers onto locators". In: *ReArch'08* (Dec. 2008). URL: http://conferences.sigcomm.org/co-next/2008/CoNext08_proceedings/ReArch08Papers/1569143769.pdf.

[49] D. Meyer and K. Patel. *RFC 4274 BGP-4 Protocol Analysis*. Jan. 2006. URL: http://tools.ietf.org/html/rfc4274.

[50] David Meyer. "LISP-TREE: A DNS Hierarchy to Support the LISP Mapping System". In: *Cisco: The Internet Protocol Journal* 11.1 (Mar. 2008).

[51] Piet Van Mieghem. *Data Communications Networking*. Techne Press, Amsterdam, 2006.

[52] P. Mockapetris. *RFC 1034 DOMAIN NAMES - CONCEPTS AND FACILITIES*. Nov. 1987. URL: http://www.ietf.org/rfc/rfc1034.txt.

[53] J. Moy. *RFC 2328 OSPF Version 2*. Apr. 1998. URL: http://tools.ietf.org/html/rfc2328.

[54] NetInf. *NetInf*. 2012. URL: http://www.netinf.org.

[55] Jon Postel. *RFC 791 INTERNET PROTOCOL*. Sept. 1981. URL: http://tools.ietf.org/html/rfc791.

[56] *SIGCOMM Award Recipients*. June 2012. URL: http://www.sigcomm.org/awards/sigcomm-awards.

[57] Richard A. Steenbergen and Rob Mosher. *An Inconvenient Prefix: Is Routing Table Pollution Leading To Global Datacenter Warming*. Oct. 2010. URL: http://www.nanog.org/meetings/nanog50/presentations/Monday/NANOG50.Talk49.Steenbergen.routingtable.pdf.

[58] Ion Stoica et al. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In: *SIGCOMM* (Aug. 2001). URL: http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf.

[59] Stanford University. *TRIAD home page*. Nov. 2011. URL: http://www-dsg.stanford.edu/triad/.

[60] Verisign. *FORM 8-K CURRENT REPORT*. Sept. 2010. URL: http://www.cidr-report.org/as2.0/.

[61] Cheng Yi. *NDN-Routing/ccnping*. Jan. 2012. URL: https://github.com/NDN-Routing/ccnping.

[62] Cheng Yi. *Properties (Java 2 Platform SE v1.4.2*. July 2012. URL: http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Properties.html.

[63] Cheng Yi et al. *NDN-Routing/OSPFN*. Dec. 2011. URL: https://github.com/NDN-Routing/OSPFN.

[64] Lixia Zhang et al. *Named Data Networking (NDN) Project NDN-0001*. Tech. rep. Oct. 2010.