Master's Thesis:
An Investigation into the application of Content-Centric Networking within Challenged Network Environments using CCNx

Tarek Elshaarani { tarek@tekbase.org }

February 6, 2014

Supervisors:
Frederik Hermans { frederik.hermans@it.uu.se }
Ferdrik Bjurefors { fredrik.bjurefors@it.uu.se }

Reviewer:
Christian Rohner { christian.rohner@it.uu.se }

Department of Information Technology
Uppsala Universitet

Abstract

Acknowledgements

Contents

Chapter 1

Introduction

A boom in user-generated content along with the widespread use of social networks has created a significant increase in the amount of data generated on a daily basis. A major contributor to this increase is the ubiquity of mobile devices that allow users to generate and consume data without being restricted to location. Despite the advances in wireless communications that allow larger coverage areas and higher transmission rates, current networks fall short in situations where there is no connectivity due to geographical limitations or when there are disruptions that affect the infrastructure. Such situations present two main challenges with regards to data distribution: 1) Unreliable connectivity 2) Location transparency. There is a continuous effort to improve upon both factors to enhance user experience. This thesis investigates Content-Centric Networking (CCN) as one such paradigm that attempts to overcome these challenges.

The conventional method of data distribution involves end-to-end connection between nodes on a network to query, publish, and retrieve information. Existing communication protocols offer unsatisfactory performance in situations where connectivity is unreliable. TCP/IP is known to perform poorly in such environments, especially because end-to-end paths between communicating nodes are often unreliable resulting in continuous interruptions. [ref?] While TCP/IP has evolved to accommodate different types of networks and suit a variety of environments, it still is not suited for some situations when used over high latency, unreliable, or dynamic networks. This is mainly due to the basic requirement to have an end-to-end connection between hosts for data to be exchanged.

Another shortcoming with conventional network architectures is in situations where node location is dynamic due to mobility. If the node holding particular data is not accessible on the network at a particular instance, the information is subsequently unavailable due to the basic requirement of hav-

ing stable end-to-end connectivity. Furthermore, if the node reconnects to the network from a different location, the data cannot be retrieved from the location it was previously known to be found. The dynamic nature of the nodes makes it unfeasible to use conventional indexing methods that map data to specific locations from which it can be retrieved.

1.1 Challenged Networks

To address the issues of both intermittent connectivity and location transparency, experimental network architectures offer different approaches that do not rely on reliable end-to-end connectivity between nodes or precise knowledge of their location. These alternate architectures are particularly relevant to one group of networks, henceforth referred to as Challenged Networks. This family of networks provide a set of unconventional architectures that attempt to address problems that are not offered by other widely deployed network architectures, namely TCP/IP based ones. They operate in conditions where there is a low power requirement, sparse connectivity, frequent mobility, and more importantly, unpredictable link disruptions. This is particularly true for wireless networks where connectivity can be intermittent due to power or coverage limitations, interference resulting in re-high transmission rates, and other mitigating factors. The nature of intermittent connectivity may vary from milliseconds to hours during often with only small windows of opportunity for data to be exchanged. [3]

The characteristics lead to the basic issue that concerns such networks, which is the the lack of continuous end-to-end communication between nodes. A generally accepted approach applied to address this drawback is a technique commonly referred to as in such network architectures as store-and-forward. This In-Network caching solution takes advantage of using resources of reachable nodes on the network as intermediaries to store data until it can be routed to the destination.

While they follow those general guidelines, there are several types of Challenged Networks that address various issues slightly differently than the others and each are a topic of research on their own. A number of the most prominent are introduced briefly below.

Delay Tolerant Networks: Mainly concerned with being resilient to communication delays, high latency, or round trip times. One application of this network would be in inter-planetary communication. [3][2]

Disruption Tolerant Networks: These are similar to Delay Tolerant Networks, and commonly grouped with them, but are more related to pre-

vention and recovery from link degradation and disruption. The nodes on the network must be able to both reconnect from link failures as well as recover from potential data loss or inconsistency if transfer is interrupted. This may be applicable to high availability networks and military use.

Opportunistic Networks: Commonly considered a subset of Disruption Tolerant Networks, this type of network makes no assumptions about reliable underlying infrastructure. They are mainly associated with ad-hoc wireless mobile networks where there is a high rate of mobility and unpredictable connectivity or coverage between nodes. Due to the lack of consistent connectivity, most communication is asynchronous and routed on a hop-by-hop basis using opportunity routing techniques. This type of network can be used to disseminate information in cases of natural disasters or censorship.

1.2 Information-Centric Networking

One proposed architecture that offers a solution for Challenged Networks is Information-Centric Networking. In contrast to today's IP-based networks which are mainly concerned with host-based addressing on the network, Information-Centric Networks (ICNs) use naming schemes to label information that can be queried and retrieved from the network. In this sense, the data distribution method is abstracted from the node level to an information level and is independent of its location. Communication is carried out in terms of requesting and publishing named information where the replication of data is handled at the network layer regardless of the originating source. This characteristic makes ICN architectures attractive for use in Challenged Network environments including applications such as content distribution (CDNs), disruption recovery, and *flash-crowd* handling.

While a number of different ICN architectures have been proposed, they all rely on three basic principles. The first is that all data is represented as objects, which are manipulated by publish and subscribe operations obsoleting the importance of data locality. The second is caching is an inherent part of the architecture supported by all nodes in the network assisting in both the acquisition and dissemination of information. The third and final principle is objects, representing data, are intrinsically secure using cryptographic signatures associated created by the original publisher and can be verified by consumers. On the other hand, there are many more characteristics that differ between architectures. These include naming and name resolution,

routing, caching mechanisms, Error and Flow control, privacy, network heterogeneity, and performance. Each of these properties is worthy of being a research area of its own account. [4] Nonetheless, ICNs are claimed to be a scalable and cost-effective solution that may replace conventional end-to-end point networks and offer more in areas where such networks are lackluster such as mobility and disruption tolerance.

Some of the prominent ICN-based projects include 4WARD, Sail, Pursuit, Connect, and Comet. [8] One of such ICN projects that currently enjoys growing interest from the research community is Named Data Networking (NDN). NDN is based on principles of an ICN, yet still builds upon the strengths of host-centric networks. [11]

1.2.1 Named Data Networking

As with other ICN architectures, NDN is publish/subscribe driven using a basic message units called Interest and Data. Each node on the network is considered a router, which has one or more interfaces , commonly referred to as faces, that may not always be active or reach other nodes. Each node also has the ability to store data, or Content Objects, in a local repository called a Content Store. These objects are referred to by a unique naming scheme consistent throughout the network and are all crytograhically signed. Additionally, each node maintains two tables: 1) a Forwarding Information Base (FIB) that maps Content Object names to faces that they can be found; mainly used for routing 2) a Pending Interest Table (PIT) that maps Interests to the face from which that they were received.

A requester or consumer of a data controls the flow of communication and it is their responsibility to ensure that the required data is received correctly. The requester sends out an Interest packet on the network by it's unique name. An intermediate node on the network that receives this packet will look up the requested object in its local Content Store in an attempt to fulfill that request. If that object is found, then a Data message containing the Content Object is sent back to the over the same face. In the case that it is not locally cached, an entry is added to a PIT to record where the Interest message came from. This also helps limit the number of Interests generated on the network as only one will be forwarded regardless of how many are received by a single node. The node will then use the FIB to identify which face or route to use to satisfy this Interest request. Because FIB entries used for routing can be prefix based, they allow for flexibility in terms of identifying the destination interfaces. It is also possible to forward that request to all available faces. The Interest is forwarded over those faces until it reaches a node which can satisfy that request. That node will then use the PIT to

identify which face the request came from and send a Data message back containing the Content Object requested. Any intermediate nodes between that node which responds to the request and the original requester will store the response in their Content Store and remove the entry for that Interest from their PIT. [6]

Interest packets can be sent using multicast or broadcast mechanisms to allow for a greater search space. A Data packet is only sent as a response to a corresponding Interest and a node will only respond with one Data packet on each face for as many Interests it receives. These properties, along with Content Store caching, provide means that allow data to be sent returned to the requester efficiently from the closest node, with resilience to mobility and link disruption.

While one aim of the NDN project is to build a foundation for the next generation of an Internet- capable architecture, it is also particularly interesting when applied to Challenged Networks. While it would theoretically provide a good platform due to the nature of the architecture, it is important to test and validate these claims in order to identify potential issues or drawbacks and how they can be addressed. [11]

Chapter 2

Problem Definition

The fundamental principle of Information-Centric Networks of not relying on end-to-end connectivity makes an important claim that they are a viable contender for use with the perturbed nature of Challenged Networks. This thesis aims to evaluate the performance of the Named Data Networking (NDN) architecture in the context of Challenged Networks, with a particular focus on its disruption tolerance capabilities. The evaluation will focus on scenarios in which end-to-end connectivity between the producers and consumers of information is unreliable as a result of mobility or link disruption. The experiments are designed around a prototype NDN implementation called CCNx which are run on Haggle, a testbed that allows the simulation of link disruptions between nodes. The evaluation is broken down into two components:

Firstly, CCNx is evaluated using a number of simple experiments that focus on testing disruption tolerance of communication between nodes through Interest message retransmission scenarios. This allows for a better understanding of the messaging protocol and how retransmission works. The effect of caching, being an inherent feature of ICNs, is also observed. The results of these scenarios are then compared to expected behavior of a conventional TCP/IP network to evaluate the relative impact on performance, when applicable. This work is presented in chapter 3.

Secondly, NDN is tested in a more complex scenario in which it is used as a foundation for a multi- player gaming platform that operates in a Challenged Network environment. The understanding of how Interest retransmission works is applied in the design of the protocol used to implement such a platform. A basic TicTacToe implementation that runs on top of that platform is presented as a sample turn-based game. The behavior of the game is then evaluated using a simulation run on the Haggle testbed in a Opportunistic Network scenario. This work is presented in chapter 4.

The thesis does not investigate NDN issues related to addressing and routing. A basic naming and addressing mechanism for CCNx is assumed and introduced in section 4.1. The subject of security is also not discussed and for the purposes of our scenarios, all data is assumed to be in clear text with a minimal cryptographic footprint that is inherent to the protocol and is ignored in any calculations provided.

Chapter 3

Evaluating NDN Performance in a Challenged Network

The CCNx protocol [10] is used to evaluate the claims of how well NDN may perform in a Challenged Network setting. CCNx is a Content-Centric Networking transport protocol based on blocks of named data. This protocol follows the basic principles of ICNs in that it abstracts location by identifying data instead of addresses on the network. Nodes that are a part of a CCNx network also function as routers that forward requests and data as appropriate. This is done through the mechanisms already described in section 1.2.1. Follows are some of the key properties of CCNx when used as an implementation.

Message Exchange: Interest and Data messages are the only forms of communication among nodes with Content Objects encapsulating the data.

Data Delivery: While it can theoretically run using layer 2 broadcast or multicast delivery, CCNx also runs on top of UDP or TCP for experimental purposes. In the cases of this experiment, UDP is used to connect different nodes on the network to facilitate the simulation of link disruption. TCP is used to connect each CCNx daemon (CCND) with the application running on the same node instance.

Naming and Addressing: CCNx names follow a simple prefix with the structure ccnx:/data/object. No complex hierarchy or structure is required for the experiments. This also simplifies forwarding as each node on the network will have an entry in its FIB for that specific prefix and the address of the node that stores that data or acts as an intermediate router to it.

On each CCNx node a simple application runs that connects to the underlying CCNx network. This application runs on two main nodes; one as a receiver, and the other as a sender. The receiver sends Interest requests asking for a particular CCNx prefix. The sender listens for such prefixes and satisfies any that match the Content Object that it may have locally. Any intermediate nodes have no applications running on them and are pure CCNx relay nodes. Once the receiver node gets an acceptable response to one of its Interest messages, the experiment is terminated. In each case, Interest satisfaction response times are measured and presented at the end of the experiment.

3.1 Performance Analysis

A number of simple experiments using CCNx nodes were conducted in order to identify how the protocol works in a challenged networking setting, particularly with regards to Interest retransmission. The experiments consist of two main nodes: a receiver node that requests a certain CCNx URI, and a sender node that holds data corresponding to that URI and responds to Interest requests. Additional nodes are added to act as relays between the two main nodes. The scenarios presented below help analyze CCNx Interest retransmission behavior using variations in link availability and interruption between the nodes throughout different stages in the CCNx message exchange.

3.2 Haggle Testbed

To facilitate the simulation of link disruptions, the Haggle testbed [7] is used. This testbed allows the specification of intervals during which links between specific nodes are either up or down. The application running on each node will then react to the state of the network. While it is out of scope of this document to go into detail of how the testbed is designed, a number of modifications were made that are worth mentioning as they had a considerable effect on accuracy and consistency of the results collected and behavior observed.

Threading: Early attempts to run the experiments on the testbed resulted in inconsistent results due to synchronization issues between the testbed components, namely the two main nodes and the testbed driver. To resolve this, a control thread was added to the testbed driver to provide a centralized launch mechanism amongst all nodes participating in an

experiment. After each node completes its initialization, it sends a beacon back to the testbed driver. The testbed driver accounts for all the nodesparticipating in that specific experiment and then broadcasts a start signal to those nodes. When the nodes receive this signal, they start their processing. This solution allowed the results to be more predictable, especially with regards to the first Interest packet in each experiment.

Uni-directional Link Disruption: The Haggle testbed utilizes iptables as a method to simulate and control link disruptions. The configuration file associated with each experiment specifies the interval when a link is considered active or up. If an interval is not specified, the link is considered broken or down. When simulating situations where there is a link breakage immediately after an Interest is sent by one node and before it is sent by another, (ie. experiment 1c) it is difficult to guarantee that iptables would have executed the firewall rules in time, especially when the window is so small. As an alternative, a modification was made to allow the configuration file for an experiment to specify when a link between two nodes can be considered half open effectively allowing uni-directional traffic for the duration of that interval.

Configuration: Various configuration changes were made to the structure of the configuration file and associated scripts to allow for the modifications presented above as well as other logistics such as specifying different applications or configuration files for on each node.

3.3 Notes on Data Collection

The intervals measured include all CCNx protocol activities, including key retrieval and content object verification, beyond the initial registration of the application with its CCND instance. The time involved in cryptographic activity is inherently absolved from the comparison by being included in all measurements.

In certain situations, responses are delayed either due to network latency, hard drive response times, application response times, or other factors. In addition, because the CCND Interest lifetime timer is checked every 250ms, there maybe be multiples of 250ms gaps in the response time. This explains why in certain scenarios the response time could be much higher than in others testing the same case. While those can be ignored as outliers, they should have no real impact on the total time of Interest success, especially when an Interest must be retransmitted.

3.4 CCNx Interest Re-transmission Behavior

3.4.1 Experiment 1: Network with 2 nodes and 1 link

This experiment involves 2 nodes, N_1 and N_2, which communicate over a single link. N_2 is the receiver node, while N_1 is the sender node.

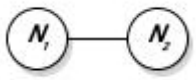

Figure 3.1 Illustration of the Experiment 1 Network Topology

It includes 4 difference scenarios to analyse Interest re-transmission between the sender and receiver nodes. The following scenarios are tested:

(a) No interruptions of disconnections.

(b) Link is down from start of run. (Interest not transmitted)

(c) Link is down shortly after start of run. (Interest transmitted, reponse lost)

A number of paramters are defined for the scenarios in the experiment:

- *Interest timeout period:* 2 seconds
- *Interval between Interest retries:* 5 seconds
- *File size:* 512 bytes

Experiment (1a): No link disruptions

This scenario involves no link interruptions or disconnections. The receiver (N_2)requests information from the sender (N_1) and this data is immediately satisfied.

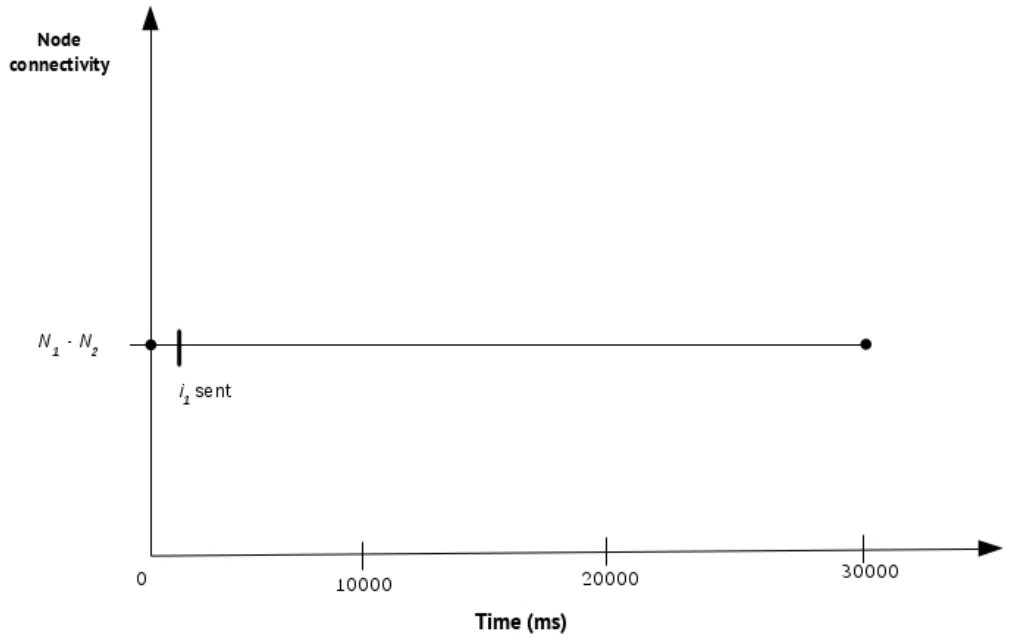

Figure 3.2 Experiment (1a) Link between nodes is always up

Observations:

Based on a number of 10 runs, the experiments yields the expected end result of the data being received based on the first Interest request (i_1). The following information shows response times for the requested data being successfully received:

Minimum=522, Maximum=609, Average=538.1 (milliseconds)

Discussion:

N_2 requests the CCN URI `ccnx:/test/1` at the application level which runs on top of the CCN daemon instance running on the same node. The application sends an Interest packet specifying the URI using the network face that is connected to the daemon. The Interest packet has a number of parameters, most notably its lifetime[1], which was set to 4 seconds for this experiment. [9] The Interest packet triggers an `interest_from` event on the the local CCN daemon which results in that Interest being added to the local *Pending Interest Table* (PIT) as there is no existing entry. A look-up is then

[1]The lifetime is not readily configurable based on the implementation code and http://www.ccnx.org/pipermail/ccnx-dev/2010-August/000249.html up to version 0.6.0).

performed on the *Forwarding Information Base* (FIB) which returns a match for the prefix `ccnx:/test` on a face, UDP in this experiment, that connects to the other node. The CCN daemon triggers an `interest_to` event which relays the request over that network face to N_1. Once the Interest is sent, the CCN daemon will monitor that prefix for the lifetime of the Interest.

On N_1, the CCN daemon receives the Interest packet on its network face. An `interest_from` event on the daemon which queries its PIT for a match, which results in no matches. The FIB is then checked for the same prefix and a match is found. The daemon then forwards the Interest to the application face. Signature verification is performed to verify the Content Object. When verification is complete, a `consume` event is issued by the daemon followed by a `content_from` event to alert the application that it should send the data across the network.

At this point, the file is cached as a Content Object in the Content Store on N_1. A `consume` event followed by a `content_to` event results in the Content Object being sent the data over the network. The data packet is received on N_2 and processed through a `content_from` event on the daemon. A lookup is performed on the PIT on the prefix. A match is found in the PIT for that Content Object and consequently a `consume` event sends it to the application face interested in the prefix. Finally, a `content_to` event signals that the application reads the data from the local CCN daemon Content Store into memory and writing it to disk to conclude the transfer.

Experiment (1b): Link interruption before request is sent

This scenario tests Interest packet re-transmission by the receiving node (N_2). This is done by simulating a loss of connectivity for a duration of 4 seconds when the experiment is first started. This loss results in the first Interest packet being sent from N_2 to be lost. The will force a timeout before another Interest packet is resent, which is then immediately fulfilled.

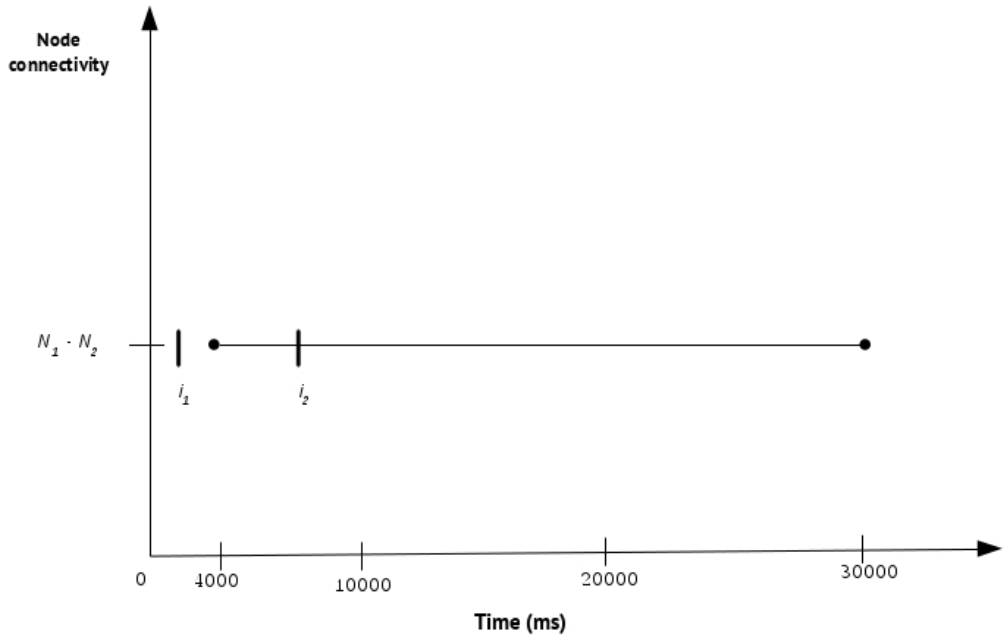

Figure 3.3 Experiment (1b) Link down before Interest is sent

Observations:

Based on a number of 10 runs, the experiments yields the expected end result of the data being received based on the second Interest request. The following information shows response times for the requested data being successfully received:

Minimum=7020, Maximum=7055, Average=7026.3 (milliseconds)

These times are noticeably longer than the ones from experiment 1b and indicate the timeout period for the first Interest, the retry interval, and the successful Interest response time.

In addition, the following times identify the response time for the (second) successful Interest request:

Minimum=19, Maximum=54, Average=25.2 (milliseconds)

These times are much lower than the time recorded in experiment 1a for an immediate (first) successful Interest.

Analysis:

N_2 starts in the same manner it did for experiment 1a. It requests the CCN URI `ccnx:/test/1` at the application level. The prefix is not found in the local Content Store or the PIT, so the Interest is added to the *Pending Interest Table* (PIT) for the lifetime of the that Interest. The FIB is then searched and a match is found for that prefix. The Interest is sent over the network and the daemon awaits a response. Because the connection between the two nodes is down at this point in time, the Interest never reaches N_1. After a lifetime of 4 seconds, an `interest_expiry` event is triggered signalling the end of the lifetime and corresponding entry is removed from the PIT.

The application on the N_2 is designed to retry the requests 3 times with a retry interval in between attempts. After it timeouts from the first Interest request, it waits for the user specified retry interval and sends another request. The second time an Interest request is sent, the link between the two nodes is up and the Interest reaches N_1.

On N_1, the CCN daemon receives the Interest and follows the same steps outlined in experiment 1a until the file is correctly received.

Experiment (1c): Link interruption after request is sent

This scenario tests Interest packet retransmission by the receiving node (N_2). In this case, the Interest packet is received by the sender node (N_1), but the response is lost due to connection loss. This loss results in the first Interest packet considered void and after its lifetime expires, a second Interest is sent, which is immediately fulfilled.

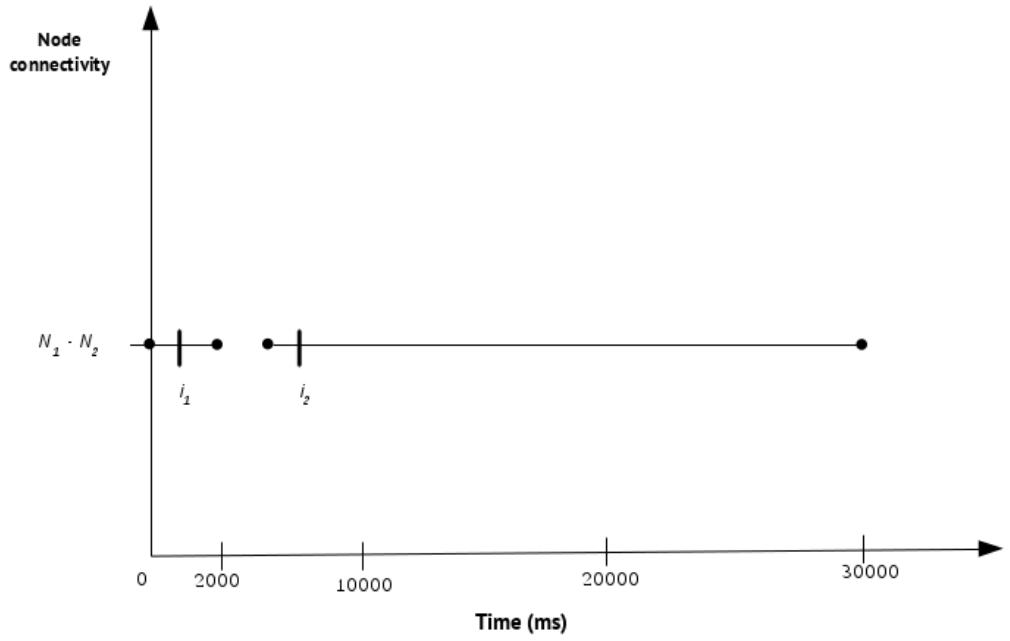

Figure 3.4 Experiment (1c) Link down after Interest is sent

Observations:

Based on a number of 3 runs, the experiments yields the expected end result of the data being received based on the second Interest request. The following information shows response times for the requested data being successfully received:

Minimum=7005, Maximum=7007, Average=7006.33 (milliseconds)

These times are very similar to the ones observed in experiment 1b due to the Interest lifetime expiry, retry delay, and second Interest request.

In addition, the following times identify the response time for the (second) successful Interest request:

Minimum=4, Maximum=6, Average=5.33333 (milliseconds)

These times are much faster than the ones recorded in 1a and 1b.

Analysis:

In this scenario, the N_2 behaves the same way as it did in experiments 1a and 1b. A request is made for the CCN URI by the application which

triggers an Interest request that is sent over the Network. In this case, the link between the two nodes is up when the Interest is received by N_1, however, the link drops before a response in the form of a Content Object is sent back.

N_1 follows the normal procedure by searching for the requested prefix in its Content Store, loading the Content Object from the local application, then sending it back over the network. However, because the connection is lost by the time those Content Object is sent back, the CCN daemon on N_2 had already expired the Interest from its PIT which results in the Content Object being discarded. The receiving application will then send a new Interest request after its retry interval elapses, which corresponds to when the connection is restored. This allows the second Interest to propagate successfully, and the Content Object to be sent back without interruption or delay.

It should be noted that in this case, the Content Object by the CCN daemon on N_1 making it unnecessary to propagate the Interest to the application running on that node. The prefix is matched directly to the Content Store and is sent back over the network without intervention from the application reducing the response time.

Conclusion

Throughout the simple experiments conducted with a single link connecting 2 nodes, it can be concluded that the CCN daemon does not submit Interest messages other than those expressed by the application driving the requests. When Interests are lost or not satisfied due to transmission errors, it is the responsibility of the application to send another Interest request until valid data is received. It is important to note that although the CCN daemon does not attempt to re-transmit Interests itself, it does provide the capability of validating data being received by matching it to the Interest information as well as originating Content Object.

In a Delay Tolerant environment, it will be the responsibility of the application to make sure that there is a continuous stream of Interest requests to recover from a loss of connectivity.

3.4.2 Experiment 2: Network with 3 nodes and 2 links

This experiment involves 3 nodes, N_1 and N_2 as well as R, which acts as a relay node. N_1 is the sender node, while N_2 is the receiver following the notation in experiment 1.

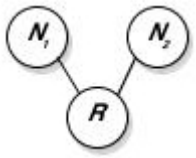

Figure 3.5 Illustration of the Experiment 2 Network Topology

It tests scenarios similar to the ones outlined in Experiment 1 with the addition of the Relay node. These three scenarios are:

(a) Both links are up.

(b) Link between N_1 and R is down before Interest reaches R.

(c) Link between N_2 and R is down just after Interest reaches R.

A number of paramters are defined for the scenarios in the experiment:

- *Interest timeout period:* 2 seconds
- *Interval between Interest retries:* 5 seconds
- *File size:* 512 bytes

Experiment (2a): No link disruptions

This scenario involves no link interruptions or disconnections. The receiver (N_2) requests information from the sender (N_1) and this data is promptly returned through the relay node.

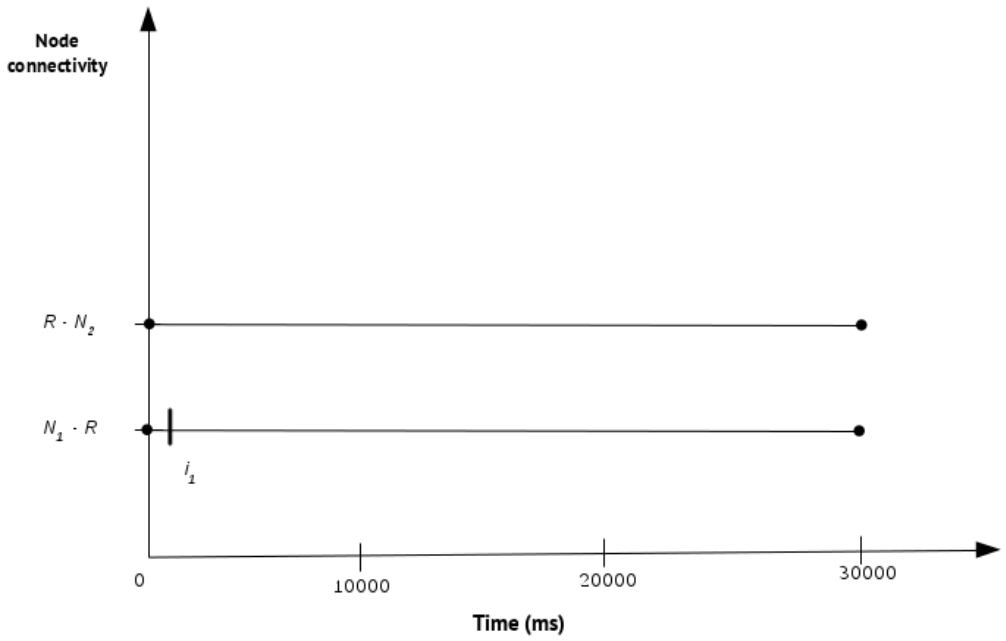

Figure 3.6 Experiment (2a) Links between nodes are always up

Observations:

Based on a number of 10 runs, the experiments yields the expected end result of the data being received based on the first Interest request. The following information shows response times for the requested data to be successfully received by node-3:

Minimum=32, Maximum=532, Average=176.3 (milliseconds)

In this case, N_2 sends an Interest which must be propagated to N_1. As an intermediate relay node, R relays the Interest to N_1. N_1 then sends data back to R, which relays it back to N_2.

Analysis:

The application running on N_2 requests the CCN URI `ccnx:/test/1`. The request in the form of an Interest message is sent to the local daemon instance running on the same node. The Interest is then added to the local Pending Interest Table (PIT). A look-up is then performed on the Forwarding Information Base (FIB) which returns a match for the prefix `ccnx:/test` on a face, UDP in this experiment, that connects to R. When the Interest message reaches R, it is added to the PIT. The FIB is then searched for the prefix which returns an entry pointing to N_1. As a result, the Interest message is then forwarded to N_1 where it is also added to the PIT. The prefix matches data which is locally served by the node, which is consequently retrieved from the application and sent back over the network. The data first arrives at R where it is verified, then cached in its Content Store. The PIT entry for that Interest is removed as it has been satisfied and the data is relayed back to N_2.

This process is very similar to experiment 1a when there are no disconnections or interruptions between 2 nodes. The only exception is the additional relaying operation that is performed by R in this case which does not appear to impact performance by a large amount when comparing response times.

Experiment (2b): Link disruption before request is sent

This scenario involves testing Interest re-transmission by the receiving node (N_2). The link between R and N_1 is down at the beginning of the experiment. This link is restored after the first Interest request expires. More than one Interest will be sent before the request is satisified.

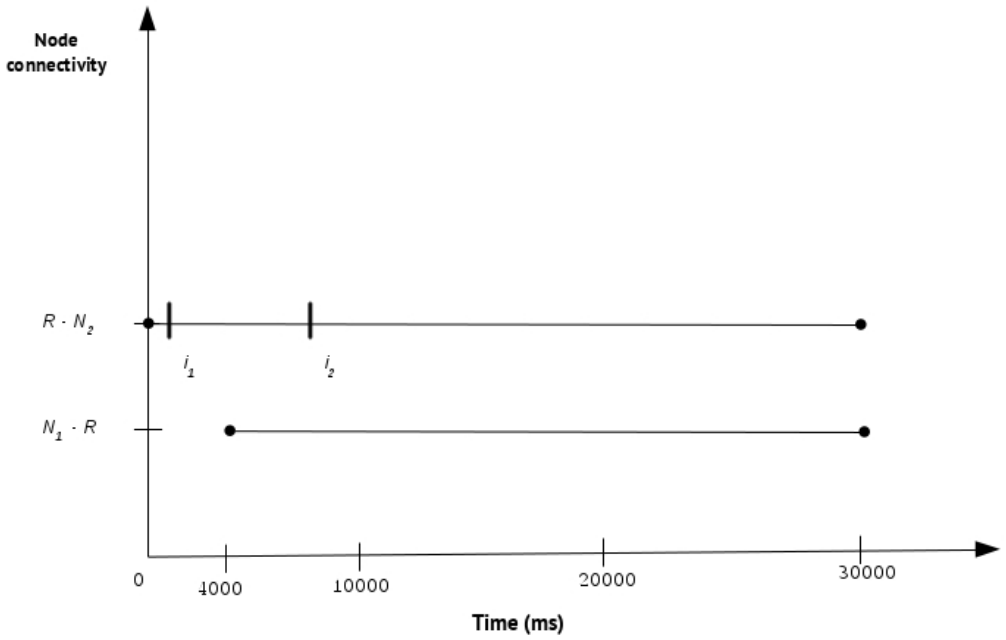

Figure 3.7 Experiment (2b) Link between N_1 and R is down before Interest reaches R

Observations:

Based on a number of 10 runs, the experiments show that 2 interests are required to successfully complete the request. The following information shows the total time for the requested data to be successfully received by N_2:

Minimum=7026, Maximum=7105, Average=7036.9 (milliseconds)

In addition, the following measurements identify the response time for the (second) successful Interest request:

Minimum=25, Maximum=104, Average=35.9 (milliseconds)

The total time it takes for Interest to be fulfilled is much longer than experiment 2a.

Analysis:

This is similar to the scenario described in experiment 2b with the exception of the connection between R and N_1 being unavailable at the start of the run. The Interest sent from the application on N_2 reaches R, but cannot

be relayed to N_1 because the link is down. After the lifetime of the Interest expires, the PIT entries expire on both nodes and the Interest message is discarded. No Interests reach N_1 up to this point.

After the Interest retry interval (5 seconds) elapses, the application on N_2 resends the Interest which is then sent to R and this time relayed to N_1. N_1 identifies the URI in the request and replies with the requested data. The data is first cached in the Content Store on R, then once it arrives at N_2 is also cached and forwarded back to the application.

The total time required to fulfill the request is noticeably longer than experiment 2a due to the need for the second Interest message to be sent. This involves the time required for the Interest message to expire, the application wait time, the time for the second Interest to be sent, and finally the time it takes the data to be relayed back.

The time for the successful Interest to be fulfilled is similar to experiment 2a which is expected as after the first Interest message expires, the entire process must be repeated without knowledge of the prior attempt.

Experiment (2c): Link disruption after request is sent

This scenario also involves testing Interest retransmission by the receiving node N_2. However, in this case, the link between N_2 and R is down immediately after the Interest request is sent from N_2. The link between N_1 and R is operational throughout the experiment. It is expected that more than one Interest will need to be sent before the request is satisfied.

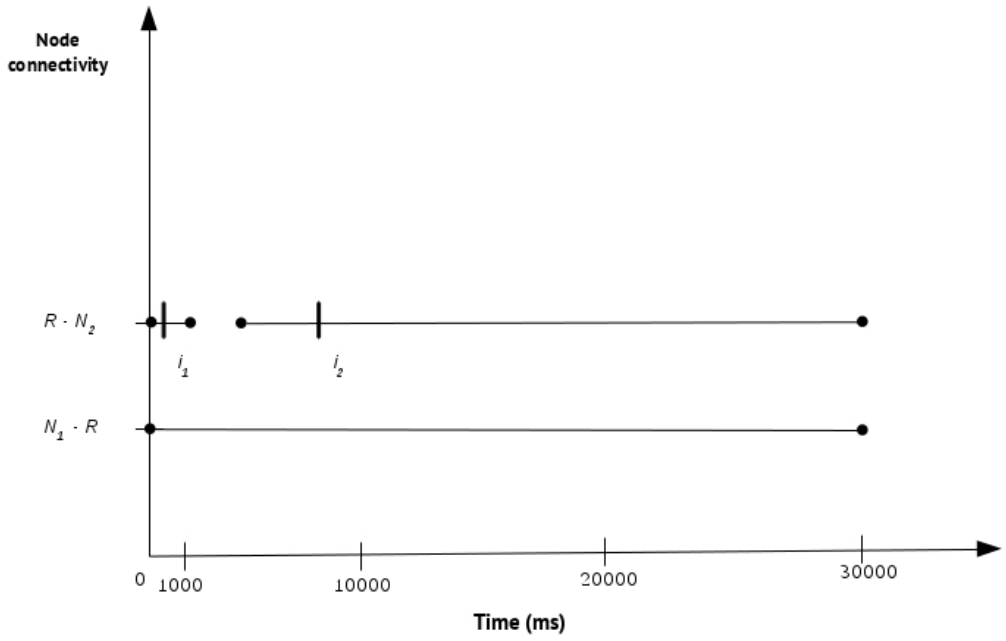

Figure 3.8 Link between N_2 and R is down just after Interest reaches R

Observations:

Based on a number of 5 runs, the results show that 2 Interests are required for the request to be satisfied. The following measurements show the total time for the request data to be successfully received by node-3:

Minimum=7006, Maximum=7008, Average=7007 (milliseconds)

In addition, the following times identify the response time for the (second) successful Interest request:

Minimum=5, Maximum=7, Average=6.2 (milliseconds)

The times for the successful (second) Interest to be fulfilled are noticeably lower than experiment 2a and 2b. In addition, while the total time required to fulfill the request is still higher than the one taken in experiment 2a, it is lower than experiment 2b.

Analysis:

This scenario is similar to experiment 2b, except that it forces Interest retransmission at a different point of time in the experiment. As opposed to N_1 not receiving the first Interest message in experiment 2b, the first Interest reaches N_1 through R. However, immediately after the Interest leaves N_2, the connection between N_2 and R is lost.

Because the Interest reaches R, it is added to its PIT and based on the FIB forwarded to N_1. N_1 then sends the data back to R, which attempts to send the data back to N_2 but fails due to the link being down. The PIT entry expires on all 3 nodes, but the Content Object remains cached in the Content Store of both R and N_1.

After the Interest retry interval (5 seconds) elapses, N_2 will send another Interest. When this message reaches R, the URI is looked up in the Content Store. Since the corresponding Content Object is still cached, a response is directly sent back to N_2. N_1 plays no role in satisfying the second Interest message.

The total time to satisfy the request is slightly shorter than experiment 2b due to the fact that the second Interest request does not need to be sent all the way back to N_1. Additionally, the response time for the second successful Interest message is noticeably shorter as well confirming that data is being retrieved from the relay node's Content Store rather than being forwarded on the network.

Conclusion

Similar to the conclusion from Experiment 1, the application is responsible to ensure that Interests are re-transmitted as required to fulfill requests. The behavior of the relay node confirms that CCNx does not interfere with Interest re-transmission. It also highlights the advantages of caching Content Objects in the local Content Store which would be an important feature in a Delay Tolerant environment. Additionally, the introduction of a relay node does not hugely impact performance when tested at similar connection speeds between the nodes.

It is expected that in a network with more relay nodes, the performance would be similar since re- transmitted Interests would be satisfied by the closest node on the network. In a Delay Tolerant setting, it would be beneficial to have Interests re-transmitted at shorter Intervals without explicit requests from the application. This would ensure that there is a larger window of opportunity around link outages for the data to be retrieved as well as a greater chance of requested data being cached on more nodes in the network resulting in faster overall retrieval times.

3.5 DTNx: Enhanced Re-transmission

From experiments 1 and 2, it was concluded that it would be beneficial to have re-transmit Interests at shorter intervals without the knowledge of the application. To test this, a separate application, DTNx, was added to each relay node that lies in the path of Interest messages being sent from the Receiver to the Sender. The application listens for Interests on the network and continuously re-transmits them until a corresponding Content Object is received. This mechanism would force the CCND daemon to cache the data quicker on nodes along the path the Interest requests are sent. The data itself is not read or used by DTNx.

To test the effect of DTNx, 2 additional experiments were performed. Experiment 3 compares the impact on communication between two nodes, while experiment 4 incorporates three nodes. Each experiment includes two scenarios which are identical except for the inclusion of DTNx in the second. In both experiments, 3 Interest requests are sent by the application, once per minute. The status of links between the nodes changes throughout the timeline of the experiment. In scenarios where DTNx is introduced, it will re-transmit any Interest requests every five seconds. Re-transmission of a particular Interest stops when either three minutes have elapsed since it was first intercepted or if corresponding data is received. It should also be noted that due to limitations of the current ccnd implementation, each Interest expressions is canceled three seconds after it is sent on both the application and DTNx in order to avoid further automated retransmission from ccnd and ensure that only one Interest is sent at a time. This workaround has no effect on results as it is consistent throughout all experiments. Observations for experiments 3 and 4 will mainly focus on the effects of DTNx rather than delve into re-transmission details already described for experiments 1 and 2.

3.5.1 Experiment 3: Network with 2 nodes and 1 link

This experiment is similar to Experiment 1 with the additional introduction of the DTNx application. The objective is to study the effect of DTNx on Interest re-transmission and performance of the 2-node network. The first scenario uses an unmodified CCNx environment. The second scenario introduces DTNx to the nodes.

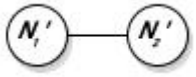

Figure 3.10 Illustration of the Experiment 3 Network Topology

The parameters used for this experiment are:

- *Interest timeout period:* 3 seconds
- *Application retry interval:* 60 seconds
- *DTNx retry interval:* 5 seconds
- *File size:* 512 bytes

Both scenarios 3a and 3b follow use the same Haggle trace file that dictates link disruptions as illustrated below:

Figure 3.10 Illustration of Link Disruption for scenarios 3a and 3b

Experiment 3a: Default CCNx behavior

The first Interest request from A reaches B before the link is dropped. The request is fulfilled by the application on B and cached in its local Content Store, however, the Content Object is not received by A. When the 3 second request timeout elapses, the application waits for 60 seconds before sending a second request. At that point in time, the link is still down and the Interest is lost. After a further retry interval, a third Interest is sent and the link is up for the Interest to reach B and Content Object to be returned directly to A. In this case, the Interest is satisfied directly from the Content Store on B and as such there is no intervention from the application on B. Once the data arrives on A, it is cached in the local Content Store and relayed to the application.

Experiment 3b: CCNx with DTNx on all nodes

Similar to the first scenario, the first Interest from A request reaches B and is cached locally. Between the first and second requests sent by the application on A, the DTNx instance on A is continuously sending additional requests. This results in the data being retrieved from B and cached locally on A. When the second Interest request is sent by the application, it can be retrieved directly from the local Content Store on A. This scenario requires ones less request to be sent by A than in experiment 3a because by the time the application sends a second request, the data is already locally cached and can be retrieved without relying on the network connection being available.

3.5.2 Experiment 4: Network with 3 nodes and 2 links

This experiment is similar to experiment 3 except it introduces an additional node which acts as a CCNx relay node, B between the sender, C, and receiver, A. The timeline is also slightly modified to accommodate the additional link.

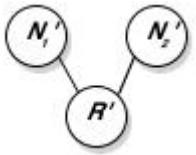

Figure 3.11 Illustration of the Experiment 4 Network Topology

The parameters used for this experiment are identical to experiment 3:

- *Interest timeout period:* 3 seconds
- *Application retry interval:* 60 seconds
- *DTNx retry interval:* 5 seconds
- *File size:* 512 bytes

Both scenarios 4a and 4b follow use the same Haggle trace file that dictates link disruptions as illustrated below:

Figure 3.11 Illustration of Link Disruption for scenarios 4a and 4b

Experiment 4a: Default CCNx behavior

The first Interest is sent from A and reaches B. Because Link2 is down, the Interest does not propagate to C. When the second Interest is sent, Link1 is again up, but Link2 is down, which means this Interest also reaches B, but not C. When the third Interest is sent from A, both links are up and the data can be retrieved successfully through B. In this case, the data is only cached locally on each node after the third request is sent.

Experiment 4b: CCNx with DTNx on all nodes

The first Interest is sent from A and reaches B. Because Link2 is down, the Interest does not propagate to C. When the second Interest is sent, Link1 is again up, but Link2 is down, which means this Interest also reaches B, but not C. When the third Interest is sent from A, both links are up and the data can be retrieved successfully through B. In this case, the data is only cached locally on each node after the third request is sent.

3.5.3 Discussion

Following the NDN architecture, it is the responsibility of each node to ensure that its requests are received, and this is what DTNx aims to focus on. It increases the rate at which those Interests are sent in order to improve the chances that a communication link is available within a shorter window of opportunity. The increase in frequency can also be more measured depending on other algorithms or calculations (i.e. Bus routes, etc)

move this to discussion? In this case, the data was retrieved with one less Interest than experiment 4a and much quicker as it is already cached locally. Despite there being no direct route between A and C when the first 2 Interests are sent by the application, it is still possible to retrieve the data through B because the increase in frequency of DTNx retransmission requests results in a higher probability for the window in which a link between 2 of the nodes is up can be taken advantage of. These experiments illustrate CCNx behavior in both DTN and opportunistic environments.

3.5.4 Conclusion

Chapter 4

A real application: Delay/Disruption Tolerant Game Platform (DTGP)

To complement the basic simulations run that test CCNx behavior in a Challenged Network setting, an implementation of TicTacToe was developed to evaluate its behavior and performance in a real application. The main requirements for designing such a game was that it must conform with the CCNx Interest-driven model and satisfy both conditions of location independence and disruption tolerance.

4.1 Framework Design

The game is built on top of a framework that utilizes CCNx and can potentially be adapted for other similar turn-based games. The framework, named DTGP, depends on two main generalization APIs. Firstly, the network API which is designed around two basic operations: get and put. These simple operations abstract the underlying network architecture while maintaining the basic principles of an ICN. Secondly, the application API which abstracts dependencies between the game and the network layer to basic functionality such as initialization, running, and terminating. It is up to the game to implement each of these functions by applying the game logic in conjunction with the network API.

The result is a game that runs on top of CCNx without requiring much understanding from the developer about the details of an ICN, yet can still function in a Challenged Network setting. It should also be noted that while the network API used may also theoretically be used with a TCP/IP network,

this has not been tested.

In the simple scenario of a game of TicTacToe, there are two players: A and B which have no prior knowledge of each other except for being on the CCNx network at some point in space and time. There are also two distinct types of nodes: 1) *Host* nodes that listen for new game requests, 2) *Initiator* nodes that send new game requests. An interest message is sent over the network through the get method presented by the framework, while a response or Content Object is delivered using the put method. A is a *Host* node that starts the game first, and continuously listens for Interest messages from interest parties in playing a new game. Player B, an *Initiator* node, then starts the game and starts to send Interest messages that request a new game with a unique game ID. The uniqueness of the game ID is crucial due to the nature of the CCN. This is discussed in some more detail in section 7.5. When the Interests from A reach B, B creates a new game object/instance and sends it back as a Content Object to A. When A receives the CO, the Interest for the new game is satisfied and the game can begin.

For simplicity, we assume that *Host* nodes always play the first move. After A receives the game CO, a new request is sent for move #1. Player B receives the Interest message and sends the response back followed by a new Interest for move #2. Player A receives the CO, updates its game state accordingly, and selects move #2. B then awaits a request for move #2 from A. The nodes continue to alternate the use of get and put methods for moves until the game reaches a final state at which the winning node will send a finalize Interest message to signal that the game has been completed. The response to this request is the final state of the game.

A number of aspects of the design relevant to the principles of the underlying architecture are visited below in more detail.

4.1.1 Bootstrapping: Discovery and Initialization

As is the case with any decentralized design, bootstrapping is always a challenge. In this case, the characteristics of the underlying ICN architecture are taken advantage of and an opportunistic approach is followed for nodes to locate others. Following the CCNx protocol, Interest messages are used as a request mechanism when a node wants to join or start a new game. *Initiator* nodes send out requests in the form of `ccnx:/uu/core/games/ttt/new/<gameid>`, where ¡gameid¿ represents a unique pseudo- random number. The uniqueness of the gameid important to avoid potential conflicts between other games on the network. Nodes active on the network that want to play TicTacToe passively listen for Interest requests with a prefix of ccnx:/uu/core/games/ttt/new. These nodes are referred to as *Host* nodes. The ¡gameid¿ portion of the URI

is parsed and used to label and a new game instance, which is then sent back to the *Initiator* node. While it is theoretically possible allow all nodes to perform both *Initiator* and *Host* roles, this was restricted to a single role per node for the purposes of this design.

This initialization process guarantees that opponents will eventually be located because all *Initiator* nodes are actively advertising their requests and that the underlying architecture provides assurances that these requests get routed through reachable intermediate CCNx nodes despite there not necessarily being a direct path to active *Host* nodes on the network.

4.1.2 Addressing and Routing

An assumption is made that addressing, including ccnx URIs, is grouped by the expected location of that data and not by content classification. This means that during the bootstrapping process, Interests are only sent using `ccnx:/uu/core/games/ttt/` as opposed to a `ccnx:/games/ttt/` prefix. This simplifies the routing and forwarding mechanism as that is not the focus of this work. That prefix is added to the FIB on each CCNx node so that TicTacToe related Interests are propagated throughout to reachable nodes.

Despite the use of this addressing scheme, nodes on the network have no prior knowledge of other nodes and communication is not end-to-end. Only content is requested and sent back as a response using the basic Interest and Data messages. Those messages do not specify senders or recipients, yet are forwarded based on demand. This may result in the network being flooded with requests before it is fulfilled, however, such a side effect is minimized due to the way the intermediate nodes do not send out multiple instances of the same Interest that already exists in their PIT. This is particularly true for situation where there may be more *Initiator* nodes than *Host* nodes resulting in endless re-transmission of Interest requests for new games that never get satisfied.

As demonstrated in chapter 3, interval and timers that control re-transmission can be configured as appropriate for the state of the network. Although not required for the chosen design, these timers may also be exploited to expire content on the network sooner than required to avoid Interest message duplication and conflicts.

4.1.3 Game Logic and State

The framework is designed to support a variety of games, however, given the nature of the underlying ICN, turn-based games are best suited to demonstrate how tolerant the implementation is to network delays or disruptions.

The game logic that runs on the platform will vary from one game to another, but still relies on basic put and get methods presented by the API.

The TicTacToe implementation is based on a standard 3x3 grid version. Each player is prompted for their move on a turn basis. Each time an Interest request is received in the form of `ccnx:/uu/core/games/ttt/<gameid>/move/<moveid>`, a Content Object is sent back with the game state encoded along with the new move requested. There is local input validation for a players move, but also validation on the remote node. If player A sends back a new game state with an invalid move #2, then player B will reject that state and resend a request for the same move while incrementing the id to move #3. Despite the case that player A will make the assumption that the move is valid and proceed to request the next move, this request is ignored. Because the players take alternating turns making moves, until player B is satisfied with the new game state, a request for a new move from player A will not be fulfilled. Player B will keep requesting moves until a valid game state is returned.

When the game state is evaluated to be final, a *fin* request is sent which results in the game ending for both players. A byproduct of this is that the *new* and *fin* requests can be used as markers for a game to retrieve and possibly replay it after it has ended. Apart from being useful for analysis, this may also be used by observers who are able to retrieve the cached copy of the game on the network on a move-by- move basis.

4.2 Design alternatives

There are a number of other designs that were considered, but not implemented or tested. These are provided below for reference as they try to take advantage of the CCNx protocol differently. These alternatives are presented but were not part of the experimentation as they either do not strictly adhere to the CCNx principles of using Interest messages for requesting information and Content Objects for supplying responses or introduced complexity that made it difficult impractical for experimentation purposes.

4.2.1 Discovery through a centralized directory

As an alternative mechanism for discovery, another proposed mechanism uses an intermediate node to act as a game directory that acts as a repository for all available games listed by participants on the network.

Player A wants to let the world know that they are available to play a game. The player creates a Content Object, called a game content objecct, under the name ccnx:/uu/core/games/ttt/open- games/¡random-game-id¿.

This Content Object contains the name under which player A wants to receive interests related to the game. For example, ContentObject name: `/uu/core/games/ttt/open-games/349057804` Content: `Player prefix=/vodafone/de/user1/` Player A listens for interests for ccnx:/uu/core/games/ttt/open-games/349057804, and for interests to ccnx:/vodafone/de/user1/ttt/349057804. In an Internet or large network setting, player A may never receive interests for that prefix, because there are no routes that point to player A for ccnx:/uu/core/games/, therefore a "game directory" is used. This directory can be thought of as a database server of open games. Player A sends a pull request to the game directory by sending an interest such as: `ccnx:/uu/core/games/ttt/pull-request/#/vodafone/d`

This interest is forwarded to the game directory. The game directory sees that this is a registration request, and identifies the part after the hash sign. It sends a dummy ACK content object back in response to the interest to signal receipt. The game directory then pulls the game content object from player A by sending an interest: `ccnx:/vodafone/de/user1/ttt/349057804/game-co`

Player A receives this interest and sends back the game content object. Note that player A cannot send it back directly, because the game content object is called ccnx:/uu/core/games/ttt/open- games/349057804, and the request was for a prefix of ccnx:/vodafone/de/frederik-hermans/. However, player A can still send a content object that encapsulates the game content object.

When the game directory receives the content object, it imports it to its own namespace. From then onward, any node on the Internet can now retrieve the game content object `ccnx:/uu/core/games/ttt/open-games/349057804`.

In a Challenged Network setting, the registration request may never be received, because player A may not have connectivity to the game directory. In that case, player A can still receive interests directly for the game content object on the `ccnx:/uu/core/games/ttt/open-games/349057804` URI.

Player Matching and Initialization

Player B joins the network and wants to play a game of TicTacToe. Player B sends an Interest for ccnx:/uu/core/games/ttt/open-games/ and gets back a game content object. It unpacks that content object and sends out an interest for the player prefix specified in the game content object. In the above example, Player B sends out an interest as follows:

`ccnx:/vodafone/de/user1/ttt/349057804/start/#/twodegrees/nz/user2/ttt/3490578`

Player A receives this interest, and sends back either a NACK (if it has already started the game with someone else), or an ACK, in which case player A and player B are considered to be active participants of game 349057804. Note that from the Interest message received, player A learns player B's

prefix which follows the hash sign.

While this approach is more complex, it augments the ad-hoc approach with a solution for situations where Interest message forwarding in a large network is impractical for each node to route based on published content as opposed to a well defined hierarchy.

4.2.2 Interests as a notification mechanism

Assume that two players have successfully found one another through a discovery mechanism and have established a game with player A being the one to make the first move. Player A sends an Interest request to player B when it has decided on its first move using ccnx:/playerB/¡gameid¿/¡gamestate¿. Player B replies with an ACK-type message to acknowledgment receipt of the game state. Afterwards, player B makes a move, updates the game state and sends out an interest as ccnx:/playerA/¡gameid¿/¡updatedgamestate¿ addressed to playerA as a notification of the next move. Player A replies with an ACK for this Interest and the cycle continues until the game reaches a final state.

This approach uses Interest messages mainly as a notification mechanism. Each Interest has its corresponding Content Object whose purpose is to serve as an acknowledgment for receiving that move. The game state is serialized and communicated as a part of the Interest message syntax making each request unique for every move as the game progresses. In this case, each player is responsible for ensuring that its own move it is received by the opponent. While this makes clever use of the messaging mechanism allowed by CCNx, it does not confirm to the basic principle of Interests acting as requests and Content Objects carrying responses to such requests. Continously using Content Objects to encapsulate an ACK does not make efficient use of the protocol.

4.2.3 Poll driven communication

Assume that a discovery mechanism is used for two players to identify one another. Player A acts as a host for the game and player B makes the first move. Player B sends out an Interest message in the form of `ccnx:/playerA/<gameid>/move/<i>/<seri`
This message specifies the move only as opposed to the entire game state. Player A replies with a Content Object that includes the game state with Player B's move applied to it as well the subsequent move made by Player A. When Player B receives that Content Object, a decision is made based on the new state and the next move is then sent in a new Interest message in the form of `ccnx:/playerA/<gameid>/move/<i+1>/<serialized-move>`.

Player A will again apply this move to the game state, decide on a new move, and send a Content Object back to Player B with the new game state. This continues until the game reaches a final state.

This design is differentiated by the point that one player is entrusted to host the game and maintain the game state throughout its duration. Each Interest message acts as as both as a request for a new move and a vessel for the current move. Content Objects carry the updated game state which contains moves made by the opponent. It is the responsibility of the non-hosting player to ensure that their move is sent and that a valid game state is received in response.

This approach exploits the messaging mechanism to send data within an Interest request and uses Content Objects as an acknowledgment, ableit carrying the game state. This method again inefficiently utilises the network and storage space as there is duplication of game state data in the Interest request and the Content Object. It puts more load on the nodes to decode each move and apply it to the game state before the logic can be analysed. It would also make it difficult for observer nodes to replay the game because requesting a cached move would require knowledge about the move itself.

4.3 Experimentation using Haggle

A set of scenarios were run to study the performance of CCNx in a challenged network environment. In all such scenarios, moves for each game are predetermined. This means that each game can be repeated while ensuring that it progresses and terminates in the same way for all runs. The *Host* node wins in every run of the experiment.
There are 2 ways to study the data collected from the experiments:

Application perspective: This approach focuses on a list of *unique game messages* composed from all Interests transmitted by either the *Initiator* nodes or the *Host* nodes. If all interests related to that game are satisfied, then the game is considered complete.

An Interest is considered satisfied if the original request from an *Initiator* node receives a response even though outstanding Interests on relay nodes have not received the a response. The list of *unique game messages* is traversed and checked for each *Initiator* node or *Host* node that there exists a `content_from` message *(inbound response)* for every `interest_to` *(outbound request)* message on the same face on the same node.

Network perspective: This approach focuses on Interests within a game regardless of which node they are associated with on the network. Each Interest request seen on the network is matched to a corresponding response.

An interest is considered satisfied if there is a corresponding `content_from` (inbound response) event for an `interest_to` (outbound request) event on the same face for the same node, regardless of the type of node. On each node, the list of all messages sent and received are traversed and sorted by timestamp. Each `interest_to` (outbound request) event is then matched with a `content_from` (inbound response) response event on the same face. Due to the nature of the experiment, it is common to have a one to many relationship between `content_from` (inbound response) and `interest_to` (outbound request) events. This is because it is likely that some of the outbound requests are lost. Consequently, the last matched `interest_to` (outbound request) occurrence from the sorted list is considered to be the actual match for the response and all other matches are re-transmissions. This is important to both count the number of re-transmissions for each Interest as well as to calculate the Interest satisfaction time.

4.3.1 Experimentation Parameters

Based on a network of 5 nodes, 21 different scenarios were run 3 times each on the Haggle testbed using a trace file (see Appendix A) that simulates link disruption over an experiment run duration of 1 minute. The number of *Initiator* and *Host* nodes vary throughout the different experiment runs, however, there are always more *Host* nodes than *Initiator* nodes. This is to avoid ending up in a situation where any one new game request is never addressed as the resulting number of high Interest counts add unnecessary noise to the data being collected.
For each experiment run, the ccnd log is analysed for a number of metrics:

- **Number of Interests per Game:** This metric helps understand the load on the network and the effects of the disruption on Interest re-transmission by analysing the number of Interest requests required to complete a game. Note that maintaining a high *Host* to *Initiator* node ratio ensures that all new game requests are fulfilled successfully.

- **Interest Satisfaction Rate:** The rate is defined as the number of responses for an Interest divided by the total number occurrences of that Interest in a run of an experiment.

- **Interest Satisfaction Time:** This metric provides insight into the efficiency of the network in the face of link disruption by looking at the time it takes to satisfy Interest requests.

4.4 Discussion

In this analysis, the focus is made on the network perspective. This allows us to study the data as a whole for all nodes and understand the load on the network created by a game as well as the behavior of CCNx on an Interest level.

Because all games reach a final state, there are no unanswered *unique game messages* that are generated from the principle nodes of each scenario, whether an *Initiator* node or an active *Host* node. However, there may potentially be some unanswered requests from relay nodes.

Number of Interests per Game:

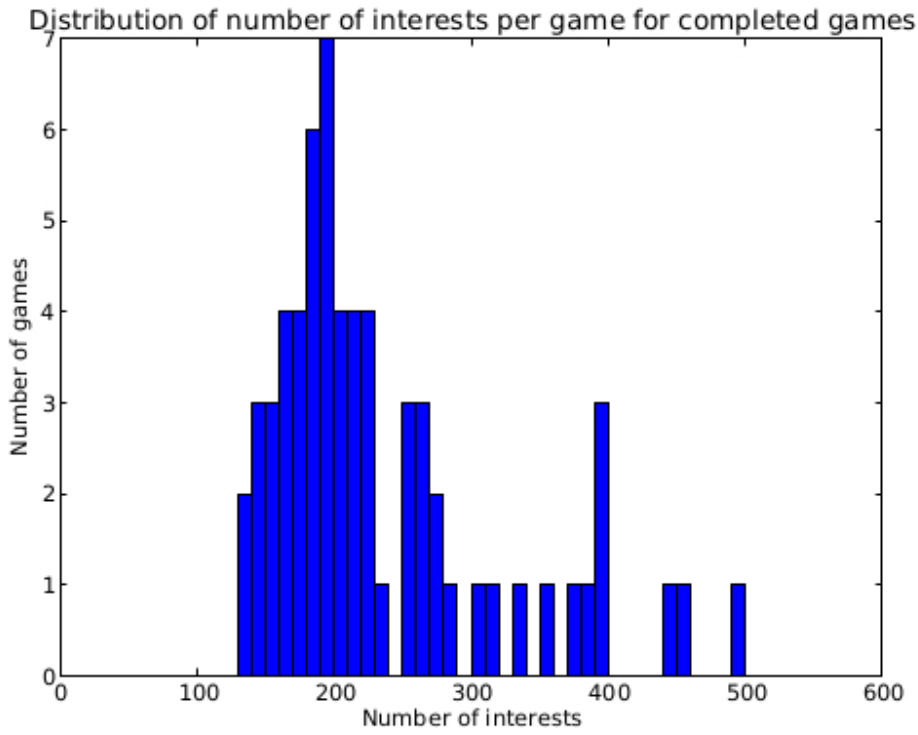

Figure 4.1 Histogram illustrating the distribution of Interests per game

This histogram shows the distribution of the number of Interests required to complete a game. As expected, there are 0 games that do not end/complete. The number of Interests required to complete a game for most cases is approximately 200 Interests.

Interest Satisfaction Rate:

The Interest satisfaction rate provides insight into the amount of overhead required for games to reach a final state in a Challenged Network environment. In the implementation, the total number of responses is divided by the the occurrences for a specific Interest. In the case where there are 0 responses for a particular Interest, then the satisfaction time is considered to be nil. This is not to be confused with the idea that the Interest is instantaneously satisfied, as in this case it is simply ignored. For example; there are 10 occurrences of `ccnx:/test/1` and 2 occurrences of `ccnx:/test/1/%jhASH@#`, then the satisfaction rate is (20%) across the entire network.

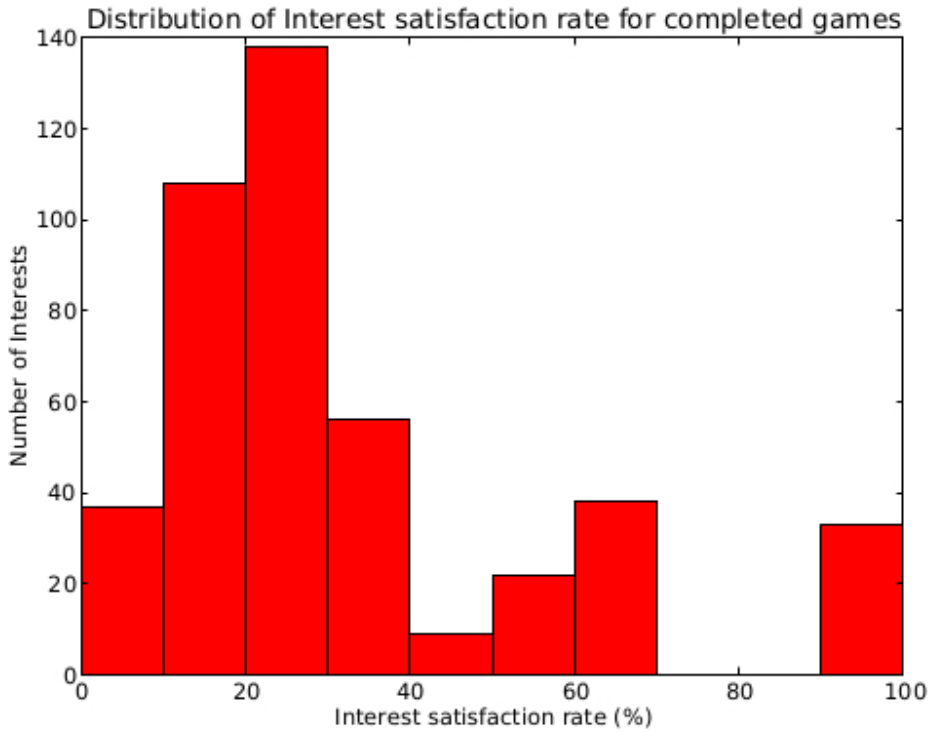

Figure 4.2 Histogram illustrating the distribution of Interest satisfaction rate

This histogram shows the distribution of the rate at which an interest is satisfied. This is on a per- Interest level irrespective of how that Interest

may influence a game. It appears that the general case is that individual interests are rarely satisfied, which correlates to the disruptive nature of the link configuration in the experiment trace file.

While there appears to be a large amount of *Interest loss* (even at 65%), this is not to be unexpected considering the nature of the challenged network environments being simulated.

Interest Satisfaction Time:

This is defined by the time it takes for a node on the network to receive a response for an Interest that it sent. There are 3 different methods to calculate satisfaction time for an Interest. As discussed earlier, Interests have no inherent state or sequence identification which makes it difficult to differentiate or track the interests. Note that an interest can only be identified by the message itself, the node it was seen on, and which face it was transmitted on.

1. Use the first occurrence of an Interest on a node as the baseline for response time calculation. This works under the assumption that one wants to measure how long it took to get a response from the first request transmitted by a node on the network.

2. Use the last occurrence of an interest on a node as the baseline for response time calculation. This would work in the case that one assumes that the latest Interest sent by a node is the one that triggered a response, even though it may have not been the initial trigger. To elaborate, a previous interest cachedthe response on a relay node nearby and this final interest pulled the cached copy.

3. Attempt a rather complex solution which involves tracking a Interest through every single *hop* and use the hop aggregates to calculate the response time. This would probably provide the most accurate response time for the interest that was actually satisfied, however, this approach runs into ambiguity in terms of which Interest actually triggered the response. If it takes 3 separate Interest retransmissions to get the data cached to an adjacent node on the network and a final fourth Interest to actually pull the response, it becomes difficult to identify which of those interests would be the baseline to use for the calculation.

The first approach is chosen as it provides a simple and straight-forward approach to identify the response time for an Interest on the network. As a result, this metric only deals with unique game messages on the network

rather than treat Interest requests transmitted on the network on an individual basis. While this provides an arguably narrower data set, it removes any ambiguity from approach #3 related to matching Interests and their corresponding responses which cannot be uniquely identified being a basic principle of a CCN.

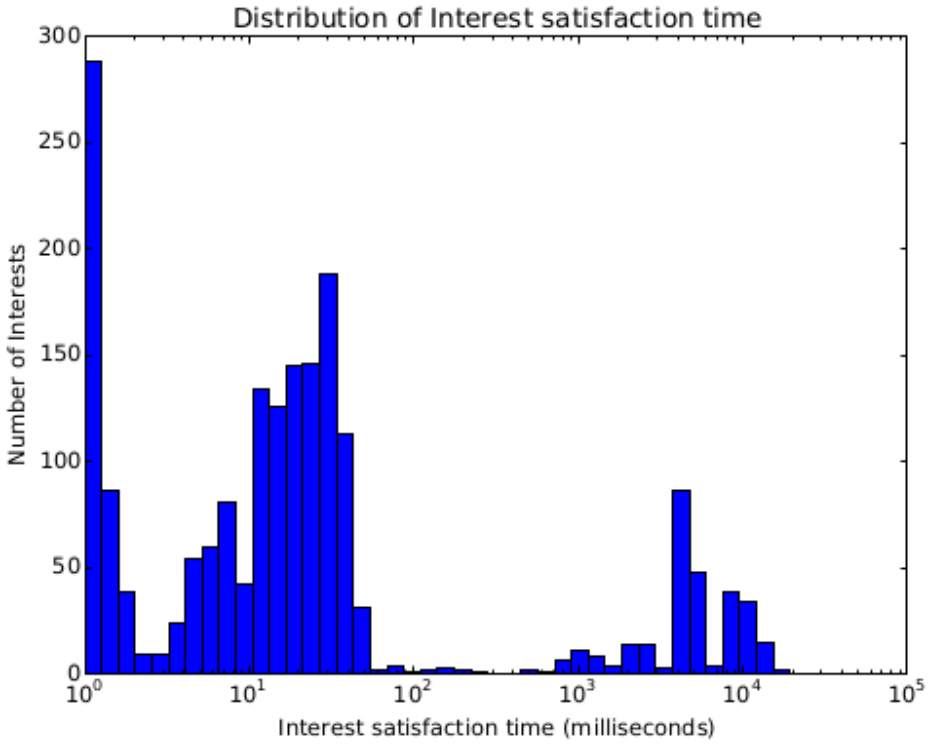

Figure 4.3 Histogram illustrating the distribution of individual Interest satisfaction time on a logarithmic scale

The histogram shows that the majority of Interests on the network are satisfied within 200 milliseconds and most just under a few milliseconds. This is likely to be due to the constant caching of responses throughout the network when a window of opportunity arises for communication between nodes when the links are up. This highlights efficiency in the network despite the disruptions that regularly take place.

There are some outliers in the data for which the satisfaction time may be up to 19 seconds. This is arguably a considerable amount of time when the duration of the experiment is 60 seconds. However, it should be noted that some of the Interests which have a longer response time may have been ones triggered by relay nodes which would not necessarily have hindered game progression.

Interest Per Node Ratio:

??

4.5 Conclusion

There is usually 200 interests per game. The satisfaction rate for each interest is usually low. Despite that, the statisfaction time is still quite low which is a good trade off especially when in an Challenged Network environment.

Chapter 5

Related Work

The term Information-Centric Networking defines a rather large subset of protocols that share the some fundamental characteristics. While separating information retrieval from being location dependent is the fundamental pillar of ICNs, some approach the problem differently. This may influence the characteristics of the network and how it behaves in certain conditions, such as Challenged Networks.

This section looks at the broader family of Content-Centric Network protocols, of which CCNx is one implementation, and compares them to how other Information-Centric protocols perform in a Challenged Network environment. The differences discussed include data distribution mechanisms, naming and addressing schemes, routing, caching, and tolerance to mobility or disruption.[1]

5.1 Publish-Subscribe Based Models

The **Publish-Subscribe Internet Routing Paradigm (PSIRP)** is a publish-subscribe based protocol that involves source nodes publishing content on the network. Nodes that want to receive the data must subscribe to specific content. A Rendezvous system matches the publishers with the subscribers and produce identifiers that form communication channels and forwarding routes that route data between the nodes.

Network of Information (NetInf) is another publish-subscribe based protocol in which source nodes publish objects to the network through a Name Resolution Service which also stores an associated list of network locators for that object. A request can then be sent to the NRS to retrieve the network locators through which the object can be retrieved.

Data-Oriented Network Architecture (DONA) uses a hierarchical

resolution infrastructure to authorize data that is published by nodes. A registered object is considered valid for a certain period of time and has a route associated with it to the source node. A request is propagated through the resolution infrastructure in an attempt to fulfill it through the closest node that has a copy of the data. Registrations can be using wildcards to force requests to a specific node without specifying one object a time. Data responses will generally be routed back along the same path the request was received.

The aforementioned models differ from CCNs as they depend on a registration or publish-subscribe mechanisms. They are based on a flat naming scheme used for their addressing mechanism. With the exception of NetInf which is purely flat, the other models use a hierarchical name resolution system similar to the Internet's DNS architecture and make use of Bloom filters for improved look-up performance. The CCN hierarchical prefix aggregation model is inherent in the architecture of the protocol which offers advantages in terms of routing performance scalability on larger scaled networks.

In-network caching is an inherent characteristic shared between the models. The request driven models follow an opportunistic method for caching data as it is seen on the network, usually as a response to a data request. Some models such as NetInf have the added ability to cache data through direct requests to the name resolution system if the requested data is registered with it. PSIRP restricts the caching to the scope of an object's rendezvous point. CCNs are unique in the sense that it could support a finergranularity of caching based on smaller parts of a data objects that are fulfilled from different nodes on the network. Space in caches can be freed by deleting objects based on available capacity based on various factors, such as aging. These caching mechanisms are more effective than edge network caching as they focus on active data on the network based on demand among a group of nodes as opposed to needlessly replicating data to all nodes on the network potentially wasting storage and bandwidth capacity.

Disruption tolerance and mobility are two of the areas that CCNs stand out compared to the other publish-subscribe models. In a Challenged Network environment where connectivity is sparse, location transparency with respect to other nodes on the network is a crucial factor. CCNs inherently support this behavior through the strategy layer that allows communication over any of the available interfaces on a node regardless of the connectivity state with the network. On the other hand, the publish- subscribe models discussed all require some form of registration with a name resolution system or rely on a common rendezvous service. These protocols suffer greatly in this regard because direct connectivity can not guaranteed. This is especially true for the relocation of source nodes that must re- register before

their information objects can be retrieved on the network.[1]

5.2 NDN Protocols

Being a subset of ICNs, NDN based protocols share a lot of common properties with CCNs. **BOND** [8] is a broadcast protocol which is based on named data and is independent of the level of connectivity or mobility between nodes on the network. This is similar to CCNx in that it attempts to utilize any available communications link to reach other nodes.

Like CCNx, BOND is also requester initiated. Requests, which are similar to Interests, are sent for named data on the network which are then forwarded to reachable nodes until the request reaches a node on the network which holds the data. The response is used to both learn the route back to the requester as well as cache the data locally on each node along the route to fulfill future requests. Prefix matching is used to identify which requests can be satisfied or forwarded. Additionally, BOND maintains data structures to keep track of messages including a Distance Table, Pending Send Index, and a Cache Store, which are synonymous to the FIB, PIT, and Information Store used by CCNx, respectively.

When processing request packets, BOND considers 2 main factors on a per node basis: 1) *Can a node forward the request?* 2) *How long to wait before forwarding the data back to the sender?* The decision on whether to forward or not is based on a distance metric between the sender and the requesting node. Nodes that are too far away are ineligible. Eligible nodes compete on whether to send the data based on a randomly based delay metric to avoid collision. Each packet sent on the network is identified by a nonce which is used to decide on whether it should respond to that request packet seen on the network or ignore it. This is similar to strategy and suppression rules that CCNx employs to control how responses are sent by nodes on the network and avoid response duplication. One difference with flow control is how BOND nodes will explicitly acknowledge receipt of data packets to avoid unnecessary data being sent to the requesting node after it's request is satisfied. CCNx avoids this mechanism and replies on the requesting node to ignore duplicate packets.

BOND classifies connectivity into connected and disconnected networks which affect its mode of operation. Nodes will react differently if they detect that they are in a disconnected network. When this mode is enabled, nodes will automatically resend packets using what is named a replay flooding technique. CCNx once again takes the simpler approach and relies on the requester to ensure data is successfully retrieved with no guarantees provided

by other nodes on the network.

While the BOND design does not mention the use of multiple interfaces in a way that CCNx does, this functionality may be inherent in the way the broadcast mechanism works. However, BOND explicitly uses layer 2 MAC transmission with custom collision avoidance mechanisms and cannot run on top of higher level (TCP/UDP) protocols like CCNx can. While this is theoretically not a disadvantage, it limits the practical use of the protocol as it cannot be used in hybrid connected and disconnected networks that run over IP.

Overall, both implementations approach the problem in a very similar fashion and as a result would be expected to perform as such. The disconnected mode BOND uses will provide assurances that data delivery will continue in a challenged network environment where there are delays or link disruptions. It is possible that the more complex BOND approach for data delivery may improve performance and reduce packet redundancy, however, it is likely that it will not be much of an improvement over CCNx.

5.3 Opportunistic Network Protocols

Another architecture commonly associated with Challenged Networks is Opportunistic Networking. In Opportunistic Networks, nodes use their locality to determine the best route throughout the network. When a message or packet is to be sent across the network, a node will independently determine the next hop based on the final destination. If such a hop is not immediately available, the forwarding decision is delayed until a later time. While this architecture does not guarantee speedy delivery, its flexible nature is well suited for environments where connectivity is unreliable, either due to delays or disruption.[5]

Chapter 6

Conclusion

Chapter 7

Future Work

It may be possible to introduce some adaptive (learning) transmission mechanism over time to avoid overly retransmitting Interests when we have no connectivity. This may improve the Interest satisfaction rate considerably. Time would not be expected to be increased dramatically? Perhaps just the outliers that take too long.

Bibliography

[1] B. Ahlgren, C Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *Communications Magazine, IEEE (Volume 50, Issue: 7)*, July 2012.

[2] V. Cerf, S. Burleigh, A. Torgerson, L. Hooke, Durst R., K. Scott, K. Fall, and H. Weiss. Delay- tolerant networking architecture. IETF RFC 4838 - Network Working Group, 2007.

[3] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM 03 Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27 – 34, 2003.

[4] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox. Information-centric networking: seeing the forest for the trees. In *HotNets-X Proceedings of the 10th ACM Workshop on Hot Topics in Networks, SIGCOMM*, 11 2011.

[5] C. Huang, K. Lan, and C. Tsai. A survey of opportunistic networks. In *22nd International Conference on Advanced Information Networking and Applications - Workshops*, 2008.

[6] V. Jacobson, D. Smetters, J. Thronton, M. Plass, N. Briggs, and R. Braynard. Networking named content. In *CoNEXT '09 Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1 – 12, 2009.

[7] M. Karaliopoulos and C. Rohner. Trace-based performance analysis of opportunistic forwarding under imperfect node cooperation. In *The 31st Annual IEEE International Conference on Computer Communications: Mini-Conference*, 2012.

[8] Michael Richard Meisel. *BOND: Unifying Mobile Networks with Named Data*. PhD thesis, University of California, Los Angeles, 2011.

[9] Palo Alto Research Center (PARC). CCNx Interest Message. http://www.ccnx.org/releases/latest/doc/technical/InterestMessage.html, July 2013.

[10] Palo Alto Research Center (PARC). CCNx Protocol. http://www.ccnx.org/releases/latest/doc/technical/CCNxProtocol.html, July 2013.

[11] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thornton, D. Smletters, Zhang. B., G. Tsudik, K. Claffy, D. Krioukov, D. Massey, C. Papadoupoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh. Named data networking (ndn) project. NDN, Technical Report NDN-0001, http://named-data.net/techreports.html, 10 2010.

Appendix A: TicTacToe Haggle Scenario

Haggle scenario trace file that specifies duration for link up-time between nodes for 5 nodes and duration of 1 minute.

```
---- Start of File ---------
5 1 84 97
1 5 84 97
3 2 71 100
2 3 71 100
5 1 113 114
1 5 113 114
5 2 147 252
2 5 147 252
5 4 243 278
4 5 243 278
3 1 189 319
1 3 189 319
3 1 333 365
1 3 333 365
4 3 380 393
3 4 380 393
5 2 388 395
2 5 388 395
5 3 342 403
3 5 342 403
3 2 336 418
2 3 336 418
4 1 350 436
1 4 350 436
5 4 374 480
4 5 374 480
5 1 522 537
```

```
1 5 522 537
5 3 577 579
3 5 577 579
4 3 565 645
3 4 565 645
5 1 615 659
1 5 615 659
3 2 679 684
2 3 679 684
5 4 684 696
4 5 684 696
4 2 720 736
2 4 720 736
3 1 658 750
1 3 658 750
5 3 641 771
3 5 641 771
5 1 718 776
1 5 718 776
3 2 765 777
2 3 765 777
4 2 847 864
2 4 847 864
5 3 841 880
3 5 841 880
5 1 878 884
1 5 878 884
5 4 858 888
4 5 858 888
5 2 841 889
2 5 841 889
4 3 1007 1038
3 4 1007 1038
5 4 1007 1111
4 5 1007 1111
4 3 1140 1141
3 4 1140 1141
3 1 971 1165
1 3 971 1165
5 3 1118 1208
3 5 1118 1208
4 2 1209 1223
2 4 1209 1223
```

```
3 2 1150 1243
2 3 1150 1243
4 3 1197 1251
3 4 1197 1251
5 3 1242 1280
3 5 1242 1280
3 2 1269 1307
2 3 1269 1307
3 2 1339 1340
2 3 1339 1340
3 2 1357 1368
2 3 1357 1368
5 1 1230 1375
1 5 1230 1375
5 2 1377 1420
2 5 1377 1420
4 2 1382 1426
2 4 1382 1426
4 3 1449 1513
3 4 1449 1513
5 1 1498 1567
1 5 1498 1567
5 4 1552 1571
4 5 1552 1571
5 3 1560 1573
3 5 1560 1573
5 4 1707 1712
4 5 1707 1712
4 2 1711 1720
2 4 1711 1720
4 3 1628 1743
3 4 1628 1743
4 1 1696 1744
1 4 1696 1744
4 1 1764 1794
1 4 1764 1794
3 1 1783 1795
1 3 1783 1795
5 2 1820 1823
2 5 1820 1823
5 2 1843 1918
2 5 1843 1918
5 3 1903 1930
```

```
3 5 1903 1930
4 1 1815 1937
1 4 1815 1937
5 2 1960 1969
2 5 1960 1969
5 3 2018 2030
3 5 2018 2030
5 3 2031 2043
3 5 2031 2043
3 2 1976 2046
2 3 1976 2046
3 2 2058 2075
2 3 2058 2075
4 2 2064 2109
2 4 2064 2109
2 1 2026 2114
1 2 2026 2114
4 2 2119 2168
2 4 2119 2168
4 1 2149 2179
1 4 2149 2179
5 2 2217 2223
2 5 2217 2223
5 3 2218 2275
3 5 2218 2275
3 1 2076 2302
1 3 2076 2302
3 1 2309 2321
1 3 2309 2321
4 3 2319 2363
3 4 2319 2363
3 2 2330 2371
2 3 2330 2371
3 1 2457 2468
1 3 2457 2468
3 2 2379 2486
2 3 2379 2486
5 2 2431 2493
2 5 2431 2493
4 1 2382 2534
1 4 2382 2534
3 1 2522 2580
1 3 2522 2580
```

```
4 2 2535 2592
2 4 2535 2592
5 4 2586 2609
4 5 2586 2609
4 3 2615 2618
3 4 2615 2618
4 1 2558 2626
1 4 2558 2626
5 1 2664 2718
1 5 2664 2718
5 2 2548 2727
2 5 2548 2727
4 2 2774 2784
2 4 2774 2784
5 2 2755 2814
2 5 2755 2814
5 4 2845 2852
4 5 2845 2852
5 3 2811 2865
3 5 2811 2865
5 1 2786 2873
1 5 2786 2873
4 3 2878 2894
3 4 2878 2894
4 2 2977 2984
2 4 2977 2984
5 3 2980 2997
3 5 2980 2997
3 2 3004 3006
2 3 3004 3006
4 3 2927 3006
3 4 2927 3006
4 3 3049 3093
3 4 3049 3093
5 2 3033 3100
2 5 3033 3100
5 4 3081 3118
4 5 3081 3118
4 1 3175 3184
1 4 3175 3184
5 4 3139 3194
4 5 3139 3194
3 1 3142 3196
```

```
1 3 3142 3196
4 3 3217 3225
3 4 3217 3225
4 2 3226 3227
2 4 3226 3227
4 1 3205 3228
1 4 3205 3228
5 1 3051 3249
1 5 3051 3249
5 4 3208 3262
4 5 3208 3262
4 2 3263 3270
2 4 3263 3270
3 1 3206 3314
1 3 3206 3314
5 1 3276 3325
1 5 3276 3325
5 2 3315 3343
2 5 3315 3343
3 2 3308 3361
2 3 3308 3361
3 1 3349 3380
1 3 3349 3380
4 1 3430 3441
1 4 3430 3441
5 2 3418 3457
2 5 3418 3457
5 3 3430 3516
3 5 3430 3516
4 3 3506 3524
3 4 3506 3524
2 1 3548 3600
1 2 3548 3600
------ End of File ---------
```