Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

An Investigation into Building a Multiplayer Online Game Using Named Data Networking

Stefano Lupo
14334933

March 12, 2019

An MAI Thesis submitted in partial fulfillment
of the requirements for the degree of
MAI Computer Engineering

TODOs

1 Abstract

2 Introduction

[?]

2.1 Background

Existing IP based internet: host abstraction, where it comes from, How it has scaled, What it supports / doesn't support (mutlicast etc), History of ICN (CCN –> NDN, Parc etc), Thin waist

2.2 Project Scope

talk about limitations etc

3 State of the Art (15 to 25)

3.1 Named Data Networking

Today, most networks make use of the so called Internet Protocol (IP) as the primary mechanism for global communication. The design of IP was heavily influenced by the success of the 20th century telephone networks, resulting in a protocol tailored towards point-to-point communication between two hosts. IP is the *universal network layer* of today's Internet, which implements the minimum functionality required for global interconnectivity. This represents the so called *thin waist* of the Internet, upon which many of the vital systems in use today are built‘ ‘[**?**]. The design of IP was paramount in the success of the modern day internet. However, in recent years, the Internet has become used in a variety of new non point-to-point contexts, rendering the inherent host based abstraction of IP less than ideal.

The Named Data Networking project is a continuation of an earlier project known as Content-Centric Networking (CCN) [**?**]. The CCN and NDN projects represent a shift in how networks are designed, from the host-centric approach of IP to a data centric approach. NDN provides an alternative to IP, maintaining many of they key features which made it so successful, while improving on the shortcomings uncovered after three decades of use. The design of NDN aligns with the *thin waist* ideology of today's Internet and NDN strives to be the universal network layer of tomorrow's Internet.

3.1.1 NDN Primitives

Interests, Data, how names are structured. Hierarchical naming to enable scalable routing analgous to IP

In NDN, as the name suggests, all data is given a name. The piece of data that a name refers to is entirely arbitrary and could represent a frame of a YouTube video, a message in a chat room, or a command given to a smart home device. Similarly, the meaning behind the names are entirely arbitrary from the point of view of routers. They key aspect is that data can be requested from the network by name, removing the requirement of knowing *where* the data is stored. NDN names consist of a set of "/" delimited values and the naming scheme used by an application is left up to the application developer. This provides flexibility to developers to structure the names for their data in a way which makes sense to the application.

NDN exposes two core primitives - *Interest* packets and *Data* packets. In order to request a piece of data from the network, an Interest packet is sent out with the name field set to the name of the required piece of Data. For example, one might request the 100th frame

of a video feed of a camera situated in a kitchen by expressing an Interest for the piece of data named */house/kitchen/videofeed/100* and this is done using the Interest primitive. In the simplest case, the producer of the data under this name, the camera in the kitchen for example, will receive this request and can respond by sending the data encapsulated in a Data packet, with the name field set to the name of the interest.

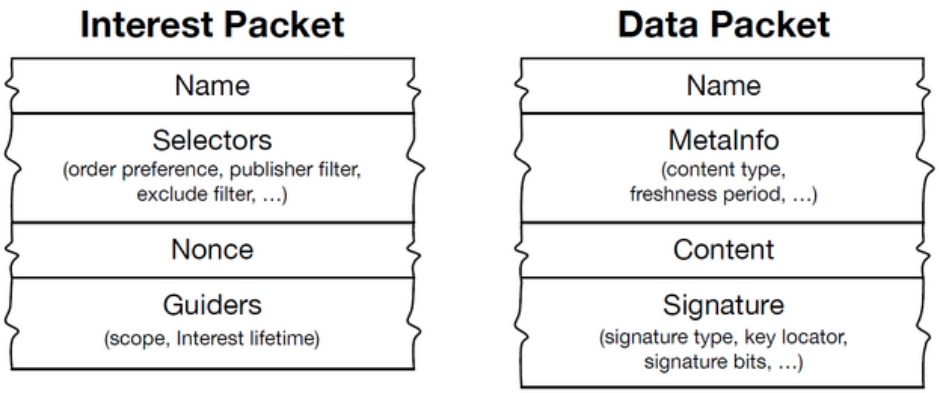
NDN communication is entirely driven by consumers who request data by sending interests and any unexpected data packets which reach NDN nodes are simply ignored.

3.1.2 NDN Packet Structures

Split primitives and this

In order to allow routers to differentiate between Interests for the same Data name, a random nonce value is added [**?**].

Figure 3.1: NDN Packet Structure [**?**]



Discuss freshness period is on a hop by hop basis!!

3.1.3 NDN Basic Operation

NDN requires three key data structures to operate - a *Forwarding Information Base (FIB)*, a *Pending Interest Table (PIT)* and a *Content Store (CS)*.

The FIB is used to determine which interface(s) an incoming interest should be forwarded upstream through. This is similar to an FIB used on IP routers, however NDN supports multipath forwarding, enabling a single interest to be sent upstream through multiple interfaces.

The PIT is the data structure which provides the stateful forwarding. This table stores the names of interests that have been forwarded upstream but have not yet had any data returned and the interface on which the incoming interest was received from.

Finally the CS is used to cache data received in response to interests expressed. The CS is subject to a replacement policy and allows any NDN node to satisfy an interest if it has the corresponding data, even if it is not the producer itself.

Upon receiving an incoming interest, an NDN router performs the following:

1. Checks the CS to see if there is a cached copy of the data corresponding to the name in the Interest

2. Checks the PIT to see if any Interests under the same name have already been forwarded
    - If there is already an entry in the PIT, the router adds the incoming interface to the PIT entry and does **not** forward the Interest upstream a second time.
3. If there is no existing PIT entry, the router checks the FIB for the interface(s) which corresponds to the name of the Interest and forwards the Interest through the interface(s). It then creates an entry in the PIT using the name contained in the Interest and the interface which the Interest came through

Once the Interest reaches a node which contains the data in it's content store or is the producer of the data, the node can create a Data packet with the appropriate name and send it back through the interface that the Interest was received on. The Data packet then follows the same path the Interest took through the network, but in reverse. Upon receipt of a Data packet, an NDN router does the following:

1. Checks the PIT and ignores the packet if there are no pending interests waiting with this name.
2. Adds the Data packet to the CS
3. Sends the data through all of the interfaces listed in the PIT for this name.
    - If there is already an entry in the PIT, the router adds the incoming interface to the PIT entry and does **not** forward the Interest upstream a second time.
4. If there is no existing PIT entry, the router checks the FIB for the interface(s) which corresponds to the name of the Interest and forwards the Interest through the interface(s). It then creates an entry in the PIT using the name contained in the Interest and the interface which the Interest came through

An important note to make is that neither the Interest nor Data packets contain any source or destination address information. This is a key component of NDN as it allows a single Data packet to be reused by multiple consumers.

3.1.4 Names

As with IP addresses, NDN names are hierarchical. This can be beneficial to applications as it allows for naming schemes which model relationships between pieces of data. In order to support retrieval of dynamically generated data, NDN names must be deterministically constructable. This means there must be a mechanism or agreed upon convention between a data producer and consumer to allow consumers to fetch data [**?**].

The names of Data packets can be more specific than the names of the Interest packets which cause them. That is, the Interest name may be a prefix of the returned Data name. For example, a producer of sequenced data may respond to Interests of the form */ndn/test/<sequence-number>*. In this case, the producer would register the prefix */com/test* in order to receive all Interests, regardless of the sequence number requested. However a consumer may not know what the current sequence number is. Thus a convention could be agreed upon such that a consumer can express an interest for */com/test* and the Data packet that will be returned will be named */com/test/<next-producer-sequence-number>*, allowing the consumer to catch up to the current sequence number.

This method is used in the synchronization protocol outlined in

ref sync protocol

.

3.1.5 Routing

Longest prefix, hierarchical naming (ndn project has some stuff on p3 2.2.1 names), NLSR, stateful forwarding can allow routers to measure performance of routes and update things accordingly (they see packets going out AND COMING BACK unlike Implementations, also it is what enables multicast and in network caching)

IP routers use *stateful routing* and a *stateless forwarding plane*. This means that routers maintain some state on where to forward packets given their destination IP addresses (stateful routing), but when it comes to actually forwarding packets, the packets are sent over the chosen route and forgotten about (stateless forwarding). NDN on the other hand, uses both stateful forwarding and routing [**?**] in order to accomplish routing packets by name and not address, as seen by the usage of a PIT.

As discussed in section **??**, NDN names are hierarchical. This allows NDN routing to scale, in a similar manner to how routing scales by exploiting the hierarchical nature of IP addresses [**?**].

There are three key benefits offered by NDN's routing mechanisms - multipath forwarding, native multicast and adaptive forwarding.

Multipath forwarding

One of the challenges of routing IP packets using a stateless forwarding plane is ensuring that there are no loops. Otherwise a single packet could loop endlessly throughout the network. The typical approach to solving this problem is to use the *Spanning Tree Protocol (STP)* [**?**] to build a loop free topology. This results in a single optimal path between any two nodes in a network and disables all other paths.

However as NDN uses a stateful forwarding plane in which Interests for the same Data name are aggregated at the router level, Interest packets cannot loop. That is, the usage of a PIT prevents looping. Similarly, as Data packets take the reverse path of the Interest packets, they also cannot loop. This means NDN can natively support multipath forwarding. This is done by allowing multiple next hops for a given entry in the FIB. This provides flexibility in the routing protocols which can be used with NDN and offers several benefits such as load balancing across entries in the FIB. Thus, to take advantage of the native multipath forwarding capabilities, a NDN specific routing protocol was developed (see section **??**)

Native Multicast

As discussed in **??**, when a router receives an Interest for an already outstanding Interest, it does not forward the second Interest upstream. Instead it adds the incoming Interest interface to the PIT entry. Once the data for the Interest reaches the router, it forwards the Data packet to **all** of the interfaces listed in the PIT entry. Thus, NDN natively

supports multicast as a producer may produce a single Data packet and have it reach many consumers.

Adaptive Forwarding

As NDN's forwarding plane is stateful, routers can dynamically adapt their forwarding strategies as the needs arise. Routers can track performance of certain upstream connections (e.g. round-trip-time) and can use this information to detect failed temporary link failures and route around them.

Hop by hop flow control? However there is also some drawbacks "Living in a PIT-LESS World: A Case Against Stateful Forwarding in Content-Centric Networking"

3.1.6 In-Network Storage

As discussed in section **??**, a Data packet is entirely independent of who requested it or where it was obtained from, allowing a single Data packet to be reused for multiple consumers [**?**]. The CS of routers provides a mechanism for opurtunistic in-network caching, which can help reduce traffic load for popular content.

NDN also supports larger volume, persistent in-network storage in the form of *repo-ng* [**?**], which supports typical remote dataset operations such as reading, inserting into and deleting data objects [**?**]. This mechanism provides native network level support for Content Delivery Networks (CDN) [**?**] and can allow applications to go offline for longer periods of time while their content is served from in-network repositories.

3.1.7 Security

Digests, trust anchors etc

3.1.8 NFD

3.1.9 NLSR

3.1.10 NDN Tools, Libraries and Frameworks etc

Implementations of CCNx or w/e, useful tools etc

3.1.11 NDN Benefits in a MOG Context

Interest aggregation, in network caching, native multicast, multipath forwarding.

3.2 Real Time Applications using NDN

3.3 Distributed Dataset Syncrhonization

Summarize the survey paper

3.3.1 ChronoSync

3.3.2 PSync

3.3.3 CCNx Sync

3.4 Video Game Architectures

3.4.1 Taxonomy of data

3.4.2 Architectures (C/S, P2P, Hybrid)

3.4.3 Dead Reckoning

3.4.4 Interest Zone Filtering

Read up on lag compensation etc

4 Problem Statement

5 Design

Explain the things MOGs must do

5.1 Player Discovery

5.2 Sync Protocol

Broadcast names (for discovery etc) need multicast forwarding strategy

5.2.1 Differences from existing protocols

outstanding interest catchup, only one writer

5.2.2 Naming Design

5.3 Interaction

This is not handled in Egal Car

5.4 Optimizations to Sync Protocol for Data types

5.4.1 PLayer Status

5.4.2 Projectile

5.5 Game Design

5.5.1 Protos

5.5.2 Very brief game functionality

5.5.3 Linkage between game and backend

diagrams of pubs / subs / game engine

5.5.4 Optimizations

Interest Zone Filtering

Dead Reckoning

Dead reckoning impacted by caching? E.g. getting someone else's dead reckoned packet? Stefan mentioned this I cant decide if it matters right now

5.5.5 Automation Script

Must be repeatable, used fixed seed for RNG

6 Implementation

6.1 Building Game with LibGdx / Ashley

Remote update reconciliation

6.2 Java Backend

6.2.1 Sequence Numbered Cache

6.3 Testing

6.3.1 Docker

6.3.2 NLSR

Building topologies Automating players, simulators, INCREDIBLES

6.3.3 Latency Calculations

7 Evaluation

Key things to examine: scale, overhead (packet size vs app data), latency Evaluation Matrix Push to breaking point

8 Conclusion

Appendices

App. A Source Code for DNS Management System