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 24, 2019

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

TODOs

1 Abstract

2 Introduction

[1]

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 [2]. 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) [3]. 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

In NDN, as the name suggests, every piece of data 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, allowing them 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

As outlined in the NDN Packet Specification [4], Interest and Data packets consist of required and optional fields. Optional fields which are not present are interpreted as a predetermined default value. The packet structure for both Interest and Data packets are shown in figure 3.1, where red fields represent required fields and blue fields represent optional fields.

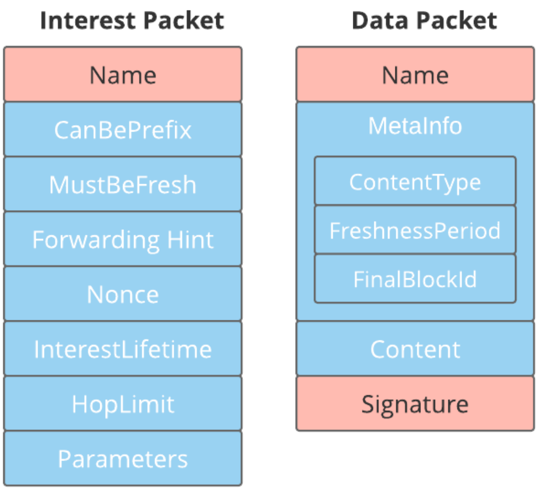

Figure 3.1: NDN Packet Structure [5]

The fields contained in NDN Interest packets are outlined below.

Name The name of the content the packet refers to.

CanBePrefix Indicates whether this Interest can be satisfied by a Data packet with a name such that the Interest packet's *Name* field is a prefix of the Data packet's *Name* field. This is useful when consumers do not know the exact name of a Data packet they require. If this field is omitted, the *Name* of the Data packet must **exactly** match the *Name* of the Interest packet.

MustBeFresh Indicates whether this Interest packet can be satisfied by a CS entry whose *FreshnessPeriod* has expired.

ForwardingHint This defines where the packet should be forwarded towards if there is no corresponding FIB entry. Due to limited capacity of the FIB, only a small number of name prefixes can be stored and the *ForwardingHint* field can aid in mapping application data name prefixes to sets of globally “reachable” names [6]. This field typically represents an ISP prefix and is used to tackle the routing scalability issues present in NDN [7].

Nonce A randomly generated 4-octet long byte string. The combination of the *Name* and *Nonce* should uniquely identify an Interest packet. This is used to detect looping Interests [4].

InterestLifetime The length of time in milliseconds before the Interest packet times out. This is defined on a hop-by-hop basis, meaning that that an Interest packet will time out at an intermediate node *InterestLifetime* milliseconds after reaching that node.

HopLimit The maximum number of times the Interest may be forwarded.

Parameters Arbitrary data to parameterize an Interest packet.

The fields contained in NDN Data packets are outlined below.

Name The name of the content the packet refers to.

ContentType Defines the type of the content in the packet. This field is an enumeration of four possible values: *BLOB, LINK, KEY* or *NACK*. *LINK* and *NACK* represent NDN implementation packets, while *BLOB* and *KEY* represent actual content packets and cryptographic keys respectively.

FreshnessPeriod This represents the length of time in milliseconds that the Data packet should be considered fresh for. As Data packets are cached in the CS, this field is used to approximately specify how long this packet should be considered the newest content available for the given *Name*. Consumers can use the *MustBeFresh* field of the Interest packets to specify whether they will accept potentially stale cached copies of a piece of Data and the *"staleness"* of the Data is defined using the *FreshnessPeriod* field.

FinalBlockId This is used to identify the ID of the final block of data which has been fragmented.

Content This is an arbitrary sequence of bytes which contains the actual data being transported.

Signature This contains the cryptographic signature of the Data packet

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.

3.1.3 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 (see section 3.1.5), enabling a single Interest to be sent upstream through multiple interfaces.

As discussed in section 3.1.5, NDN uses *stateful forwarding* and the PIT is the data structure which maintains the forwarding state. This table stores the names of Interests and the interface on which the Interest was received, for Interests which have been forwarded upstream, but not yet had any Data returned.

Finally, the CS is used to cache data received in response to Interests expressed. The CS allows any NDN node to satisfy an interest if it has the corresponding Data pakcet, even if it is not the producer itself. As with all caches, the CS is subject to a replacement policy, which is typically *Least Recently Used (LRU)*.

NDN also offers a *Face* abstraction. An NDN Face is a link over which NDN Interest and Data packets can flow. A Face can represent a physical interface such as a network card, or a logical interface such as an application running on a machine producing data under a certain namespace.

The operation of an NDN node on receipt of an Interest packet is shown in figure 3.2.

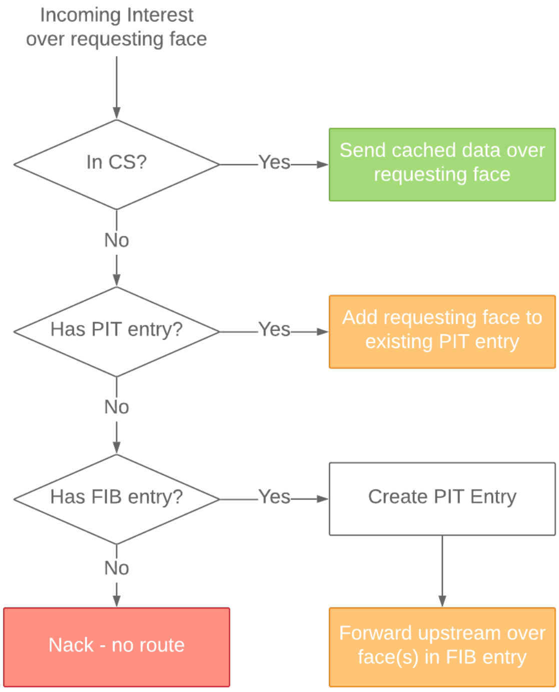
Incoming Interest
over requesting face
In CS?
Yes
Send cached data over requesting face
No
Has PIT entry?
Yes
Add requesting face to existing PIT entry
No
Has FIB entry?
Yes
Create PIT Entry
No
Nack - no route
Forward upstream over face(s) in FIB entry

Figure 3.2: NDN operation on receiving Interest

On receiving an Interest, the CS is checked to see if there is a cached copy of the Data corresponding to the name in the Interest. If a copy exists with the appropriate freshness, the Data packet can simply sent back over the requesting Face and the Interest packet is satisfied.

If there is no cached copy of the Data in the CS, the PIT is then checked. If a PIT entry containing the Interest name exists, this indicates that an equivalent Interest packet has already been requested and forwarded upstream. Thus, the Interest packet is **not**

forwarded upstream a second time. Instead, the requesting Face is added to the list of downstream faces in the PIT entry. This list of faces represents the downstream links which are interested in a copy of the Data.

If there is no PIT entry, the FIB is then queried to extract the next hop information for the given Interest. If there is no next hop information, a NACK is typically returned. In some implementations, the Interest could also be forwarded based on the *ForwardingHint* if one is present. If an FIB entry is present, a PIT entry for the given Interest is created and the packet is forwarded upstream.

The operation of an NDN node on receipt of a Data packet is shown in figure 3.3.

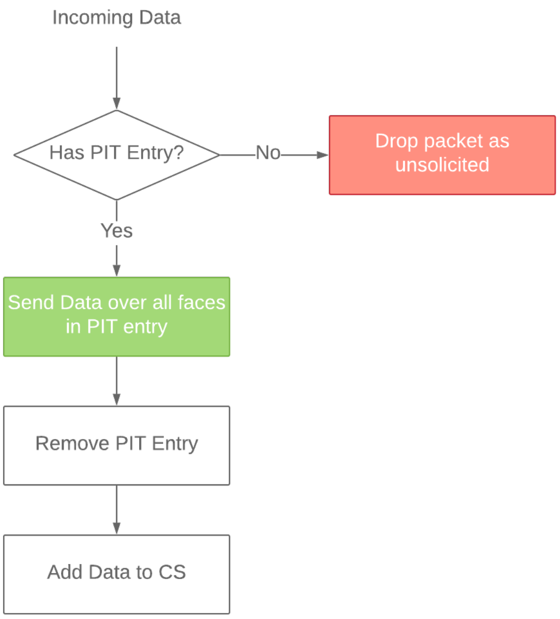
Incoming Data
Has PIT Entry?
No
Drop packet as unsolicited
Yes
Send Data over all faces in PIT entry
Remove PIT Entry
Add Data to CS

Figure 3.3: NDN operation on receiving Data

On receiving a Data packet, the PIT is checked to ensure that the Data packet had actually been requested. If there is no PIT entry, the node never expressed an Interest for this piece of Data. This means the Data is unsolicited and is typically dropped.

Otherwise, the Data packet is sent over all of the requesting faces contained in the PIT entry, the PIT entry is removed and the Data is added to the CS.

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 [8].

The names of Data packets can be more specific than the names of the Interest packets which trigger 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

and Forwarding 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 [9] in order to accomplish routing packets by name and not address, as seen by the usage of a PIT.

As discussed in section 3.1.4, 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 [9].

The use of a stateful forwarding plane is NDN has some drawbacks such as added router operation complexity and the addition of a new attack vector through router state exhaustion attacks, due to the limited size of the PIT [10]. However, there are three key benefits offered by NDN's stateful forwarding plane - 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 forwarding 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)* [11] 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, Interest packets cannot loop. As discussed in section 3.1.2, Interest's contain a *Nonce* field, allowing Interests to be uniquely identified. If an NDN router sees an Interest which is identical to an Interest in the PIT, the Interest is ignored as a loop has been detected. 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 3.1.10)

Native Multicast

As discussed in section 3.1.3, when a router receives an Interest which matches an entry in the PIT, it does not forward the second Interest upstream. Instead it adds the Face over which the incoming Interest was received to the PIT entry. Once the data for the Interest reaches the router, it forwards the Data packet to **all** of the faces 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 where they forward packets as the needs arise. Routers can track performance metrics such as round-trip-times of upstream connections and can use this information to detect temporary link failures, or poorly performing links and route around them.

3.1.6 In-Network Storage

As discussed in section 3.1.3, 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 [5]. 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* [12], which supports typical remote dataset operations such as reading, inserting into and deleting data objects [13]. This mechanism provides native network level support for Content Delivery Networks (CDN) [5] and can allow applications to go offline for longer periods of time while their content is served from in-network repositories.

3.1.7 Forwarding Strategies

The choice of how to forward packets in NDN is defined by a *forwarding strategy*. Several strategies have been designed for NDN such as *Best Route*, *NCC*, *Multicast* and *Client Control* [14]. However, *Best Route* and *Multicast* are the most common. To forward packets, a list of possible next-hops is obtained from the FIB for a given Interest. For the *Best Route* strategy, the Interest is forwarded over the best performing Face, ranked by a certain metric such as link cost or round trip time. For the *Multicast* strategy, Interests are forwarded over all Faces which are obtained from the FIB for a given Interest.

As one would expect, Forwarding strategies play a major role in the performance of an application using NDN. For example the *Multicast* strategy should be used only in scenarios where multicast is beneficial or required as it can cause a major increase in the number of Interests which must be sent across the network. However application's correctness can also be affected by the forwarding strategy [15]. For example, if a *Best Route* strategy is used in a distributed dataset synchronization context, it is possible that only a subset of participants will see published updates and thus *Multicast* should be used in this context. This is outlined further in .

Reference Player Discovery multicast

3.1.8 Security

Digests, trust anchors etc

The aspect of security in NDN is the shift from attempting to secure communication channels to focusing on securing the data itself at production time.

Typically, the main form of secure communication in the Internet today is using the Transport Layer Security (TLS) protocol [16] along with the Transmission Control Protocol (TCP) over IP (TCP/IP). As discussed in section 3.1, TCP/IP is a mechanism for allowing communication between two nodes in a network. TCP/IP sets up a communication channel between the hosts and TLS is used to secure that channel.

NDN on the other hand focuses on securing the Data packets produced in response to Interests. As shown in section 3.1.2, NDN Data packets must contain a *Signature* field. A cryptographic signature is generated using the producers public key, binding the producer's name to the content. [17].

As NDN uses public key cryptography, all applications and nodes must thus have their own set of keys and a means for determining which keys can legitimately sign which pieces of data. NDN uses three key components in this regard - *NDN Certificates*, *Trust Anchors* and *Trust Policies*.

NDN Certificates bind a user's name to its key and certifies the ownership of this key [17]. Trust Anchors are the certificate authority for a given NDN namespace. NDN nodes can then verify published certificates by backtracking along the trust chain until a Trust Anchor is reaced. Finally, Trust Policies are used by applications to define whether or not they will accept certain packets based on naming rules and Trust Anchors.

3.1.9 NFD

In order to provide the NDN functionality, the *NDN Forwarding Daemon (NFD)* was developed. NFD is a network forwarder that implements the NDN protocol [18]. The NFD thus implements all of the features described in section 3.1.3 such as the CS, PIT and FIB.

As NDN strives to replace IP as the universal network layer, NDN can run over a variety of lower level protocols such as Ethernet, TCP/IP and UDP/IP. The NFD provides this functionality by abstracting communication to dealing with *Faces*. *Faces* can be backed by a variety of transport mechanisms such as UDP/TCP tunnels or Unix sockets. This allows applications using the NDN Common Client Libraries (see section 3.1.11) to communicate with the NFD through the Face abstraction.

The API of the NFD provides a means for creating *Faces*, adding *Routes* and specifying *Forwarding Strategies*.

A typical set up of applications using the NFD is shown in figure 3.4. The NFD requires faces to be created before operation. In this case, as part of the setup procedure, node A would create a *Face* towards node B, backed by a UDP tunnel towards node B's IP address. Node A would also create a *Route* towards node B by specifying the prefix node B is responsible for, along with the ID of the Face previously created. In this case the route would map */ndn/nodeB* to face 2. Note that this process is somewhat automated by using the prefix discovery protocol of NLSR (see section 3.1.10). Finally, node A can specify the *Forwarding Strategy*, or use the default of *Best Route*.

As nodeB is a producer, it only needs to create a *Face* towards the local NFD (face 7 in this case) and inform the NFD that it will be producing data under the prefix */ndn/nodeB*. This is done using the *registerPrefix* call provided by the NDN client library. This will create the a route in node B's NFD which maps */ndn/nodeB* to face 7.

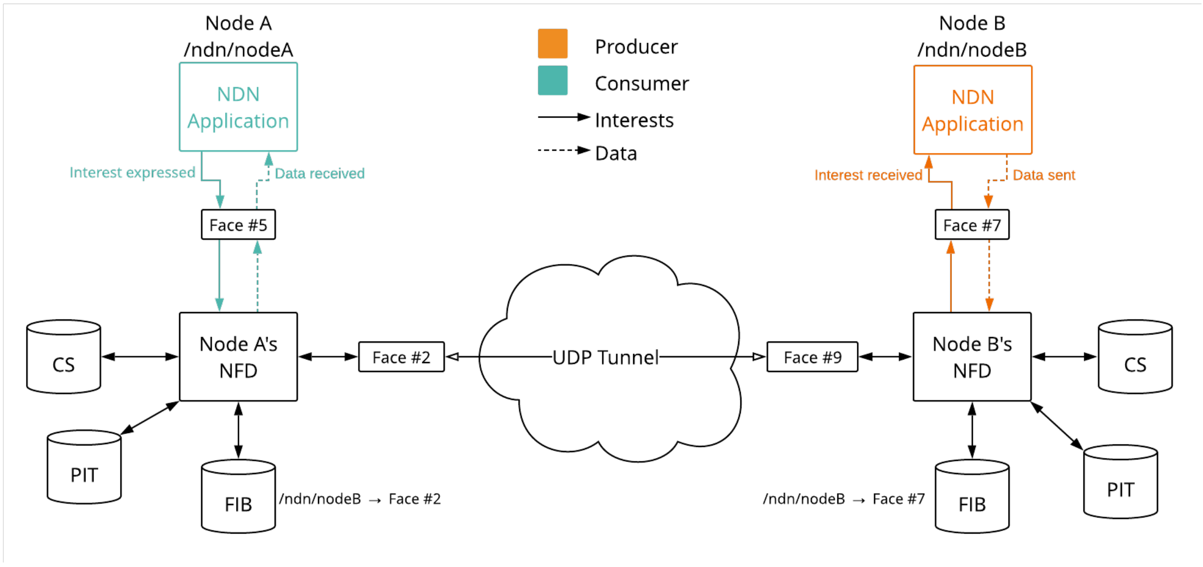

Figure 3.4: An example NFD setup

With the NFDs configured, an example operation would be the following (with some of the basic operations of NDN omitted for brevity):

1. Node A's application creates a face towards the local NFD (face 5 in this case).
2. Node A requests node B's status by expressing an Interest for */ndn/nodeB/status* through face 5.
3. Node A's NFD checks the CS and PIT which are empty and finally determines the next-hop for the Interest is through face 2.
4. Node A's NFD sends the Interest through the UDP tunnel towards node B's NFD which accepts UDP connections on NFD's default port of 6363. This creates face 9 on node B's NFD in the process.
5. Node B's NFD then finds the FIB entry created for */ndn/nodeB* when nodeB registered the prefix and forwards the Interest over face 7

6. Node B's application will then create the corresponding Data packet and send it over face 7
7. Node B's NFD will check the PIT for a list of faces to forward this Data packet over and will find face 9
8. The Data packet will reach Node A's NFD via the UDP tunnel
9. Node A's NFD will extract the list of downstream faces for this Data packet from the PIT and will send the packet over face 5 to node A's application.

3.1.10 NLSR

NLSR

3.1.11 NDN Tools, Libraries and Frameworks etc

Implementations of CCNx or w/e, useful tools etc

3.1.12 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)

This is particularly evident in applications built on a peer-to-peer architecture, where a lack of a single authority presents additional challenges in synchronising the state of shared objects while presenting a responsive simulation [19]

3.4.3 Dead Reckoning

Reconciliation between dead reckoned position and remote update methods. Dead reckoning (DR) is a short-term linear extrapolation algorithm which utilises information relating to the dynamics of an entity's state and motion, such as position and velocity, to model and predict future behaviour. [19]

3.4.4 Area of Interest Management

PubSub paper has interesting stuff on IZF

Read up on lag compensation etc

3.5 Closely Related Projects

3 of em

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

Things can't be parametrized in NDN without forming a gross name schema - e.g. interaction API can't send parameters. Potential solution is to have publisher's maintain outstanding interests towards consumers for interactions?

6 Implementation

6.1 Building Game with LibGdx / Ashley

Remote update reconciliation

6.2 Java Backend

6.2.1 Sequence Numbered Cache

6.2.2 Concurrency

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

In order to examine the performance of the game, . . .

Need to decide what game objects to use

7.0.1 Round Trip Times

As a bench mark, no caching, no DR, no IZF RTT for each topology

7.0.2 Effects of Enabling Caching

Discuss difficulties of maintaining fresh cache in MOG scenario where data changes are not predictable (freshness period etc), tree topology makes G should never really be getting any cache hits cause any data that is cached at G should have been forwarded to one of the intermediate routers who would then cache the data. For Dumbbell the intermediate routers should only be caching data on the OPPOSITE SIDE? If an interest arrives at F for C/D, it will be cached at F on the way back. However once it reaches E it will be cached there too. If A or B then request the interest it should be cached and served from E not from F. As in-network caching is one of the main benefits of NDN for typical use cases such a serving content, the impacts of using caching for the MOG were studied. Theoretically, enabling caching would directly impact the following

isher Interest Rate

Round Trip Times

ıote Update Deltas

IT KIND OF MAKES SENSE THAT CACHE RATES ARE LOW AS ITS ALMOST ALL INTEREST AG. BUT ENABLING IZF AND DR COULD CUASE CACHE HITS ROUND TRIP TIMES WITHOUT DEAD RECKONING ARE ALMOST ENTIRELY DEPENDENT ON LOCALPLAYERSTAUTS PUBLISH RATE!!!!

7.0.3 Effects of Interest Aggregation

This is really not working the way I thought it would be :/ Compare interests received for each node's status to sum of interests expressed towards that node by all other nodes

7.0.4 Effects of Forwarding Strategy

Multicast shouldn't make a difference in tree like topology if interests are aggregated as there is only one upstream node from each route to a data source as defined by NLSR. On square however, producers should see all of the interests provided their local NFD's don't aggregate the interest which they dont seem to be.

8 Conclusion

8.0.1 Further Work

More rhobust interaction API, concurrent issues (e.g. two people shoot block at same time?)

Things can't be parametrized in NDN without forming a gross name schema - e.g. interaction API can't send parameters. Potential solution is to have publisher's maintain outstanding interests towards consumers for interactions?

Bibliography

[1] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, Lixia Zhang, Ilya Moiseenko, Yingdi Yu, Wentao Shang, Yi Huang, Jerald Paul Abraham, and Steve DiBenedetto. Nfd developer's guide. *Dept. Comput. Sci., Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0021*, 2014.

[2] Ndn executive summary, . URL `https://named-data.net/project/execsummary/`.

[3] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael Plass, Nick Briggs, and Rebecca Braynard. Networking named content. *Communications of the ACM*, 55(1):117–124, 2012. ISSN 00010782. doi: 10.1145/2063176.2063204.

[4] Ndn packet specification, . URL `https://named-data.net/doc/NDN-packet-spec/current/`.

[5] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[6] Alexander Afanasyev, Xiaoke Jiang, Yingdi Yu, Jiewen Tan, Yumin Xia, Allison Mankin, and Lixia Zhang. Ndns: A dns-like name service for ndn. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9. IEEE, 2017.

[7] Xiaoke Jiang, Jun Bi, and You Wang. What benefits does ndn have in supporting mobility. In *2014 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2014.

[8] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D Thornton, Diana K Smetters, Beichuan Zhang, Gene Tsudik, Dan Massey, and Christos Papadopoulos. Named data networking project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 157:158, 2010.

[9] Cheng Yi, Alexander Afanasyev, Ilya Moiseenko, Lan Wang, Beichuan Zhang, and Lixia Zhang. A case for stateful forwarding plane. *Computer Communications*, 36 (7):779–791, 2013.

[10] Cesar Ghali, Gene Tsudik, Ersin Uzun, and Christopher A Wood. Living in a pitless world: A case against stateful forwarding in content-centric networking. *arXiv preprint arXiv:1512.07755*, 2015.

[11] Radia Perlman. An algorithm for distributed computation of a spanningtree in an extended lan. In *ACM SIGCOMM Computer Communication Review*, volume 15, pages 44–53. ACM, 1985.

[12] Ndn repo-ng github page, . URL `https://github.com/named-data/repo-ng`.

[13] Ndn repo-ng homepage, . URL `https://redmine.named-data.net/projects/repo-ng/wiki`.

[14] ndnsim forwarding strategies, . URL `https://ndnsim.net/2.1/fw.html`.

[15] Hila Ben Abraham and Patrick Crowley. Forwarding strategies for applications in named data networking. In *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems*, pages 111–112. ACM, 2016.

[16] E. Rescorla T. Dierks. Rfc 5246, the transport layer security (tls) protocol version 1.2. URL `https://tools.ietf.org/html/rfc5246`.

[17] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. An overview of security support in named data networking. *IEEE Communications Magazine*, 56(11):62–68, 2018. ISSN 0163-6804.

[18] Nfd github page. URL `https://github.com/named-data/NFD`.

[19] Patrick J Walsh, Tomás E Ward, and Séamus C McLoone. A physics-aware dead reckoning technique for entity state updates in distributed interactive applications. 2012.

Appendices

App. A Source Code for DNS Management System