MULTIPATH TRANSMISSION FOR CONTENT-CENTRIC NETWORKING IN VEHICULAR AD-HOC NETWORKS

Bachelorarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Thomas Kolonko
2017

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Contents

List of Figures

List of Tables

Acknowledgments

On this page I would like to thank everybody who supported me to write this bachelor thesis. First I would like to thank my coach Eirini Kalogeiton. She supported me trough the whole process of writing this thesis, spend many hours for pair programming sessions and gave valuable inputs when nothing seemed to work anymore. After that I would like to thank Prof. Dr. Torsten Braun who allowed me to write this thesis in his research group. I am also very grateful for the resources that were generously provided by the research group of Prof. Braun.

Abstract

Content-Centric Networking (CCN) is a new network approach based on the information-centric networking paradigm (ICN), which tries to provide a more secure, flexible and scalable network. CCN does not operate on a host-to-host basis but emphasizes the content itself by making it directly addressable and routable. Knowledge of the topology is therefore not required as the routing is based on the names of the content and happens only locally on the intermediate nodes. This allows much more efficient communication in highly mobile and dynamic environments, where no static topology is given, and devices enter and leave a given area very frequently.

In this thesis a new routing strategy for VANET's has been developed in ndnSIM version 2.0 within the ns-3 network simulator. The current implementation supports only routing based on incoming and outgoing faces. Since Wifi and other forms of wireless networking broadcast data by design, a new mechanism has been introduced for forwarding and discriminating by mac addresses. Two new fields have been added to the interest and data packages for original and target mac addresses. FIB tables were extended with the new mac address fields in order to forward the interests to the correct upstream nodes. PIT tables were also extended so that the data could follow the interests way back to the requesting consumer. A new strategy has been implemented for flooding the interest if the FIB entries have not yet been populated with known mac addresses and to forward specifically hop by hop if the FIB entries showed actual intermediate target nodes with lower costs. (TODO: add some info about the multiple netDevices when done)

The scenario was implemented and evaluated using the ns-3 network simulator. Simulation with 100 nodes were conducted for 21 times and ...... (TODO: finish this part as soon as you have some results)

(TODO: CHANGE WORDING from Amadeo14 BEGIN) -¿ Performance evaluation is carried by means of ndnSIM, the official NDN simulator, that is overhauled for use in realistic wireless ad-hoc environments. Results collected under variable traffic loads and topologies provide insights into the behaviors of both forwarding approaches and help to derive a set of recommendations that are crucial to the successful design of a forwarding strategy for named data ad-hoc wireless networking ¡- (TODO: CHANGE WORDING from Amadeo14 END)

Chapter 1

Introduction

In the past years content production and dissemination have both drastically increased. That has led among others to the wide use of Content Distribution Networks (CDN) that tackled the problem of distributing content more efficient and without slowing down the main servers. Not only the amount of data has changed but also the overall mobility has increased exponentially. A commuter expects to stream full HD movies on her cell phone while traveling in trains to work at high speeds. Mobile IP tried to solve this, but the main problem of an secure and always active host-to-host connection still remains.

1.1 Motivation

NDN is an implementation of ICN and tries to solve the above mentioned problems through a paradigm shift in networking. It is build upon the same innovative concepts as other ICN implementations. These are among others the named content and the routing by that name as in-network caching of data. The communication is also based on the Interest / Data model instead of maintaining an end-to-end connection at all time. This makes NDN with ndnSIM (it's simulator) very appealing for mobile ad-hoc environments. There are a few forwarding strategies already implemented within ndnSIM but as of ndnSIM version 2.0 they all work through faces and point to point wire connections. That makes it very difficult to use in a wireless fashion, since all net devices have the same face id's.

1.2 Study Subject

The goal of this theses is to implement a basic forwarding strategy that can be used in a mobile ad-hoc environment which then can be used as a baseline. The basic implementation of the strategy should then be reiterated into a improved strategy with better bandwidth efficiency. This goal can be divided into four subtasks. First a new forwarding mechanism should be implemented by not only using faces but also a further unique identifier, like a mac address. FIB and PIT entries need therefore to be extended in order to hold additional information about previous and next nodes of both interests and data. The second task of the thesis is around implementing a forwarding strategy that decides how to route the interests, once the flooding is done and the FIB and PIT entries have been populated with the correct mac addresses. The third part is applying both mentioned mechanisms into a mobile ad-hoc scenario where nodes move around leave and enter the observed premises. The fourth part is to achieve a multipath transmission where specific nodes simultaneously send out interests and receive data.

1.3 Outline

The remainder of this thesis is organized as follows. Chapter 2 explains shortly why a paradigm shift takes place from host-based networking to content-centric networking, explains what CCN is and why it could outperform the TCP/IP model. Chapter 3 explains ndnSIM and it's current implementation of the forwarding mechanism and the strategy as of version 2.0. Chapter 4 shows the steps taken to implement a new forwarding based on not only faces but also on macs and discusses the multipath approach taken. In Chapter 5 the results are evaluated and discussed. Finally a conclusion of the thesis is offered in Chapter 6. (TODO: take out the appendix or write here something).

Chapter 2

Related Work

Todays use of the Internet is heavily based on content dissemination of all kinds of media like audio and video. TV-Shows, clips and tutorials on Youtube, podcasting for educational purposes like on Coursera need to be distributed around the globe. Popular TV-Shows are being distributed globally within the first few hours after being telecast to the audience through p2p networks. When a video clip goes viral millions of requests for the content are made from all over the world. In 2008 alone 500 exabytes were created and today's number is a multiple of that [TODO: reference]. This became possible, since computers and computing devices got so inexpensive that almost anybody can afford them. Limited storage on clouds is given free to all users by many providers. When the Internet was invented, there were only a few computers and few resources like tape storage devices, archives or computing power distributed geographically. A client requested some specific information or resources from a specific destination which needed to be known to the client. The client was connected through TCP/IP to the server, and after the connection was established and possibly secured, the transfer of the information needed by the client could start. Host-centric networking was very reasonable at that time, but the Internet evolved towards content-dissemination and with it's evolution the need for new and more efficient solutions.

TCP/IP is no longer best suited for todays use of the Internet. One of the main reasons is that a TCP/IP connection is established between two machines and requires it to be active at all times. For content dissemination a mechanism that supports multipoint to multipoint connections that do not have to be active at all times might be much better. Another reason is that a client often does not know where to get some specific information from (and doesn't really care) but knows exactly what it wants. Therefore a paradigm shift from host-centric networks started to evolve towards content-centric networks.

2.1 Named Data Network / Content Centric Networks

Information-centric networking (ICN) is a possible answer to today's problems with TCP/IP architecture. ICN originated as a possible new paradigm for the future Internet to improve on scalability, reliability and efficient content distribution. One of the many ICN architectures is content-centric networking (CCN) originally introduced by Van Jacobson. It is being continuously researched around the globe and one initiative trying to implement CCN is Named Data Networking (NDN) project.

In CCN the content is made directly addressable and routable by name. An Interest (request by a client) is sent out and routed according to it's name until it reaches some endpoint that is able to respond with Content Objects to that Interest. Some of the main goals of CCN is better utilization of the bandwidth by increasing throughput and decreasing network traffic, better security, availability, flexibility and scalability of the networks. Better utilization of the bandwidth can be achieved by multicasting the same content to several endpoints and not re-unicasting the same content over and over again through the same channels near the content source. Also, content caching in intermediate nodes reduces the strain on bandwidth and increases overall efficiency. Better security is achieved by hashing and signing the content itself instead of the point-to-point connection between two hosts. Encrypting the data leads to more privacy within the Internet and could substitute many current access policy patterns used by servers and web pages. (TODO: don't understand why not) Better availability and flexibility are intrinsically given by content caching. Better scalability is achieved by not needing pre-planned structures like content delivery and P2P networks. Data is not only kept at the content producer but everywhere along the route if necessary.

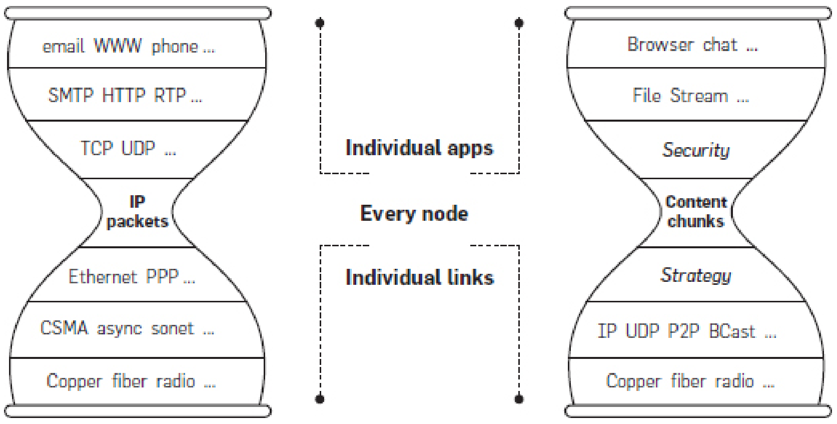


Figure 2.1: TCP/IP protocol stack on the left and CCN protocol stack on the right

In Figure **??** the IP and CCN protocol stacks are compared to each other. Both protocol stacks are built in a modular fashion that makes the architecture very flexible and scalable. The thin waste of the TCP/IP protocol stack consists of IP packages that have a source and a destination address. This Internet Layer is kept very simple and it makes only very weak demands on the lower Network Access Layer. This thin waist in TCP/IP protocol stack (*where*) is replaced in

CCN with a content layer (*what*) that describes what the package is and has even fewer demands on the lower layer, keeping much of the advantages from IP. Lower layers of the CCN protocol stack are responsible for the routing, encoding and decoding of the information, while the higher layers consist of security and interpretation of the information. Because of the modularity, CCN can be implemented on top of IP.

Two big differences of TCP/IP and CNN are the strategy layer and the security layer. The strategy layer is responsible for all dynamic routing decisions based on the name and the strategy. The strategy can be a different one for different namespaces. E.g. an emergency message could be always broadcast according to it's name. The Security layer differs from the TCP/IP protocol stack, since the content chunks are signed and encrypted themselves, contrary to TCP/IP, where the connection is secured.

2.1.1 CCN/NDN Node Model

In CCN there are no clients and servers anymore but **consumers** and **producers**. Consumers request some information by sending out an **interest**. This interest packet consists of a content name, some selectors and a nonce. The interest is being forwarded according to the node's strategy until it reaches a node that can satisfy it. If a node can satisfy the received interest, it will respond with a **data** package consisting of the same content name as the interest, a signature, signed info and the data. The data will be sent back towards the consumer. The node having the requested information is called producer (it generates the data). Interests and data are received and sent out through interfaces which can be network or application interfaces.

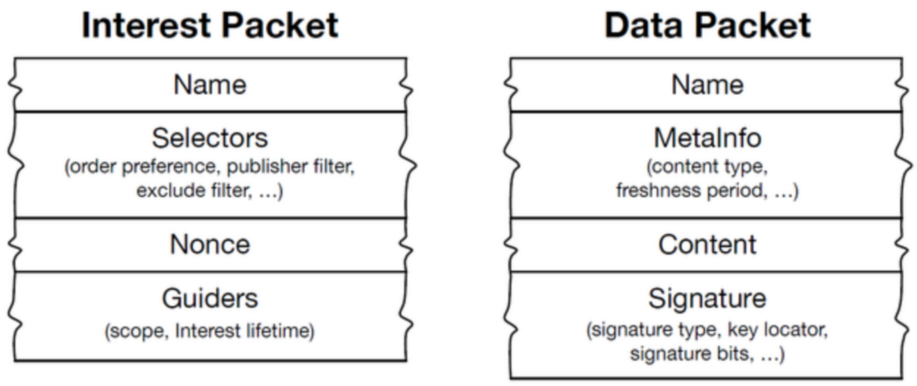

Interest Packet
Name
Selectors
(order preference, publisher filter, exclude filter, ...)
Nonce
Guiders
(scope, Interest lifetime)
Data Packet
Name
MetaInfo
(content type, freshness period, ...)
Content
Signature
(signature type, key locator, signature bits, ...)

Figure 2.2: Overview of the packet structure for interests and data messages in NDN

The most important data structures for routing the interest to the producer and the data back to the consumer are called the pending interest table (PIT), the forwarding information base (FIB), and the content store (CS). A PIT entry is also needed to detect loops among other things. If an interest arrives at the node having the same content name as a previous interest, it's nonce will be checked against the PIT entry's nonce. If both nonces are the same the interest has been looped otherwise another consumer requested the same data.

PIT

The Pending Interest Table (PIT) keeps track of all the interests that have been forwarded towards potential content sources. It also keeps track of all incoming and outgoing faces of the specific interests (multiple in-faces and out-faces reflect the multipoint to multipoint characteristics of CCN). If a second interest with the same content name but different nonce arrives at the node, the incoming face will be added to the already existing PIT entry of the previously forwarded interest. The interest will not be re-forwarded and shortly after dropped. It won't be deleted right away since a looped interest is identified by it's nonce. When an interest reaches a content source or a producer, data is send back. This data message follows the breadcrumbs (faces) that are left in the PIT entries in order to find it's way back to the consumer. The PIT entries are deleted shortly after the requested data has been sent downstream.

CS

The Content Store (CS) is located within the intermediate nodes. It is a cache of data packages that have passed this node and have been saved for later use. That is a critical advantage over TCP/IP where data packages are meant only for point to point delivery and are not cached for other possible content requester. It depends on the implementation of the CS to decide which packages should be saved to the cache and how they should be replaced if the cache is full. Data packages can be solicited or unsolicited. If a data package is solicited then the data was requested by forwarding an interest. Solicited data commonly is stored into the CS. Unsolicited data packages were not requested by the node but overheard. They could be pro-actively cached for later use and flagged for immediate removal if the cache fills up. Current implementation focus mainly on Least Recently Used (LRU) and Least Frequently Used (LFU) replacement strategies. The CS needs to be searched for data that could satisfy the received interest before the interest is handed off to the forwarding strategy.

FIB

The Forwarding Information Base (FIB) is used by the strategy to forward interests upstream towards potential producers or intermediate nodes, that have cached the requested data. Every interest that needs to be forwarded will be matched against the FIB entries by the longest prefix match algorithm. It basically checks the content name of the interest with the FIB entries and chooses the entry with the largest number of leading name components, since the routing is done by content name. If an entry is found the interest will be send upstream to the outgoing faces. If there is no match the interest can be broadcast or dropped according to the network implementation strategy. The FIB is always checked last if there is no PIT entry (it has not yet been forwarded or it has been already satisfied) and there is no CS data that can satisfy the interest directly.

2.1.2 Forwarding of an Interest

Interests are forwarded based on the content name and the implemented strategy on all intermediate nodes. The above discussed tables are used for deciding if and how to further process the interest. The strategy is only responsible for forwarding the interests towards content sources or producers. The data coming back follows the path of the interest back to the consumer.

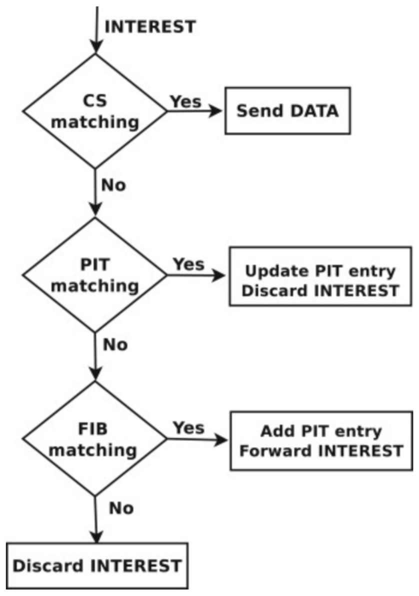


Figure 2.3: Shows how the data structures are used in order to forward or discard the interest

When an interest arrives at an intermediate node the content store is checked first if the requested data has been cached already. If the interest can be satisfied by some cached data, the data message could be sent downstream towards the consumer and the interest is dropped (not further processed). If the CS has no data with the same content name as in the interest, the PIT entries are checked. If a PIT entry already exists for the interest, the node has already requested the data and is awaiting it. In that case, the incoming face(s) are added to the existing PIT entry and the interest is dropped. If a PIT entry does not exist yet and no cached data can satisfy the interest, the node checks the FIB entries for a longest prefix match, in order for the interest to be forwarded upstream towards a potential content source. If an entry is found, a new PIT entry needs to be created with the name of the interest, it's incoming face and outgoing face (from the FIB). The interest is then forwarded according to the FIB and the strategy.

Sending Data downstream to the original requester is straightforward since no special routing is required. The Data follows the breadcrumbs of the interest left the PIT entries. These breadcrumbs are the incoming faces the interest and serve as the outgoing faces for the data message.

2.1.3 Transport and Routing

As mentioned above the ”content chunks” layer in the CCN protocol stack makes even weaker demands on the lower layers than the IP layer makes its lower layer, the Network Access Layer. It operates on unreliable, best-effort packet delivery services in potentially highly dynamic and mobile environments. Interests and Data packages are expected to get lost and/or corrupted. In CNN the strategy layer of the intermediate nodes is responsible to retransmit the interest if within the timeout no data has been received. The strategy layer knows which outgoing faces were used and what the timeout was, therefore it is able to adjust the parameters for a retransmission. The same is true for the strategy layer of the consumer. If it does not receive any data back within a given timeframe, it too, will retransmit the interest. Flow control can be managed by the consumer in terms of how many interests can be sent out before receiving the first data packages back. It is also managed on a hop-by-hop basis. Each intermediate node decides when to retransmit an interest due to loss or corrupted data coming back. The buffer for the interests is the PIT whereas the buffer for the data passing downstream to the consumers is the CS. There is no need for special congestion control techniques. (TODO: find the reference for that)

2.1.4 Sequencing

One of the big advantages in CCN over the host-to-host based TCP/IP approach is that the data in transition can be used many times by many different consumers. That leads to the problem of uniquely identifying the data in a self explanatory way. Consumers must be able to deterministically construct a name for the data without having previously seen it. Hierarchical names that reflect the content and the organizational structure of their origin are very well suited to solve that problem. Since the naming is absolutely irrelevant for the network (TODO: find the reference for that), applications can choose a naming that fits their need best. For example, to get the segment 34 of a video version 2 by group A of the University of Bern the name could be:
/unibe.ch/group/A/videos/introduction.mpg/_v2/_s32
This name can be aggregated with more components if needed by the application or network. It only needs to adhere to previously specified rules.

2.1.5 Network Security

CCN digitally sings and encrypts the content itself and not the connection over which it travels. TCP/IP needs to secure the connections, which in turn must link the content to the server infrastructure. To trust a content the user must fetch it from it's original source making it very difficult to cache popular content and make it available to other users. For a rich and robust content-based security model the consumer must be able to assess the integrity (content is not corrupted), pertinence (what question does it answer) and provenance (who claims this is an answer). TCP/IP can only provide weak integrity through a checksum and only implicit pertinence and provenance through securing the host to host channel and therefore trusting the source and destination addresses. CNN, on the other hand, transparently provides then content name and therefore the meaning of the content which satisfies pertinence. Through public key signatures the consumer and any intermediate node can check the content's authenticity, therefore verify integrity and satisfy provenance. Content Protection and Access Control could be solved solely by encrypting the content with different keys. No trusted servers or directories would need to enforce complicated access policies on the filesystem. Expensive authentication services like SWITCH AAI could be saved. If for example certain documents within a database should be only readable by a certain group of people these documents could be simply encrypted while still accessible to everyone. Only the people with the correct key could encrypt it and use it. Network security is improved against many classes of network attacks. Every node can possibly (if the resources allow it) check the integrity of the data and cache it for further use. There is no single host that provides the data, therefore hiding content from consumers is very difficult. DDoS attacks with data packages are not really a problem since every node can ignore unsolicited data coming in. If resources allow it, the node can simply store it in CS and mark it as unsolicited. If the space runs low on the CS these data packages will be removed first. No propagation of unsolicited data takes place since no PIT entries exist for the data. DDoS attacks therefore would have to be done through interests. If the prefix stays the same the PIT entry get's updated for every new interest, but no forwarding takes place. If the prefix changes constantly the strategy has many means to take action like limiting the rate of the interests with certain prefix patterns or lower prioritization of interests that result in data coming back.

2.2 VANETs

Vehicular ad hoc network's (VANETs) operate in very dynamic and mobile environments under possibly poor and intermittent connectivity. To the few static roadside unites (RSU) there are many devices ranging from trackers and phones on pedestrians other vehicles like cars, motorcycles or drones to airplanes and satellites as shown in Figure **??**.

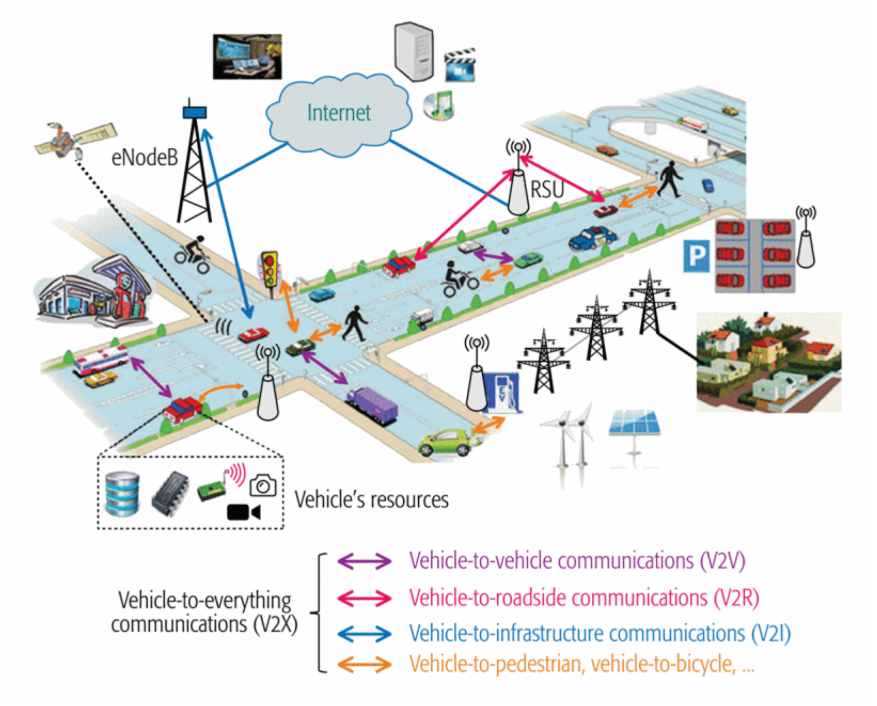

Internet
eNodeB
RSU
P
Vehicle's resources
Vehicle-to-everything communications (V2X)
Vehicle-to-vehicle communications (V2V)
Vehicle-to-roadside communications (V2R)
Vehicle-to-infrastructure communications (V2I)
Vehicle-to-pedestrian, vehicle-to-bicycle, ...

Figure 2.4: Different communication agents interacting with each other through different channels

In such dynamic and mobile environments the TCP/IP based approach that focuses on the source and destination and their secured connection quickly becomes a burden. Dynamic name-to-IP resolutions are difficult to do and infer a high management overhead. Keeping the connection up in urban areas where signal propagation is obstructed frequently can quickly become a challenge. Mobile IP is a workaround for this problem but does not solve the issue really. The CCN approach on the other hand seems to be very well suited for such ecosystems where the focus lies on the information itself (trusted road safety information for a specific area) and not on the identity of the host (other vehicles, RSU, Internet) the data might have originated from. With in-network caching the CCN approach also tackles the problem of poor signal strength and intermitted connectivity within a very heterogenous network system. A vehicle can store information and propagate it to an otherwise disconnected area through a *store-carry-and-forward* mechanism. The multipoint to multipoint characteristics already mentioned before allows to aggregate the same interests (maps, safety warnings, road condition, congestion warnings....) and multicast the arriving data through different faces and different channels back to the consumers simultaneously.

There are two main routing schemes for VANETs. In the *proactiv* scheme the content providers advertise periodically in order to keep the FIB entries updated with fresh routing information. In the *reactive* scheme no advertisement by the content providers is done and the FIB's are populated based on interest flooding. Flooding-based discovery seems to be better suited for VANET's since periodical FIB updates on all intermediate nodes incur a high and mostly unnecessary cost on the network. An interest is able to find it's way quickly to some node having a copy of the data or the producer itself. Collision avoidance and packet suppression is done on lower layers, although packet suppression will be done in the forwarder module in ndnSIM (see chapter 3). Caching policies in VANET's differ slightly from regular CCN use since the vehicles are moving fast into and out of regions relevant to the data. Data about possible congestion gets outdated pretty quickly so do often warnings of any kind. Further the question arises if unsolicited data should be cached into CS and if yes, under which conditions and for how long. As mentioned before vehicles could link otherwise disconnected areas, but in that case the data naming should be clear or that specific intent.

While the vehicles and devices are getting smarter and are being equipped with ever more sensors, it is expected that they will produce a huge amount of data. Most of it will be aggregated and logged, some of it will be available for distribution some of it should remain private. There are many open questions how to best support the information flow given the networks capacity. Although not originally intended by ICN the intermediate nodes could be included into in-network processing of the data like filtering useless data or aggregating redundant information.

Chapter 3

ndnSIM

NDN is a newly proposed Internet communication paradigm that tries to keep most of the well accepted and tested TCP/IP Internet architecture (as shown in **??**), while evolving the thin waist introducing many new features. These features differ in fundamental ways from a point-to-point communication architecture and need to be simulated extensively under constantly changing parameters and implementations.

NdnSIM is an open source simulator for NDN networks within the existing NS-3 simulator framework. Its main goals are to facilitate experimentation inside the research community and make all basic NDN protocol operations accessible and therefore changeable like routing, data caching, packet forwarding and congestion management. Packet-level interoperability with CCNx implementation is given in order to support traffic measurements, traffic traces and analysis tools between CCNx and ndnSIM. Large-scale simulations should be supported and made easy to set up through helper classes. Helper classes automate the repetitive creation of single entities like nodes and set them up in a standardized way. Therefore, the simulator has been implemented in a very modular fashion making it very easy to modify, replace or re-implement specific components like the FIB, PIT or forwarding strategy. Replacing components have minimal or no impact on other components, as long as they adhere to the modules API's and other components they interact with.

3.1 Design overview

NS-3 and ndnSIM both follow a philosophy of maximum abstraction for all modelled components making experimentation on one hand very fine-grained and on the other hand very isolated and decoupled from the rest. The NDN core protocol stack can be installed on every simulated network node in a consistent manner through helpers, that take care of all the parts that need to play together like the inter-node communication with installed applications through their respective application faces.

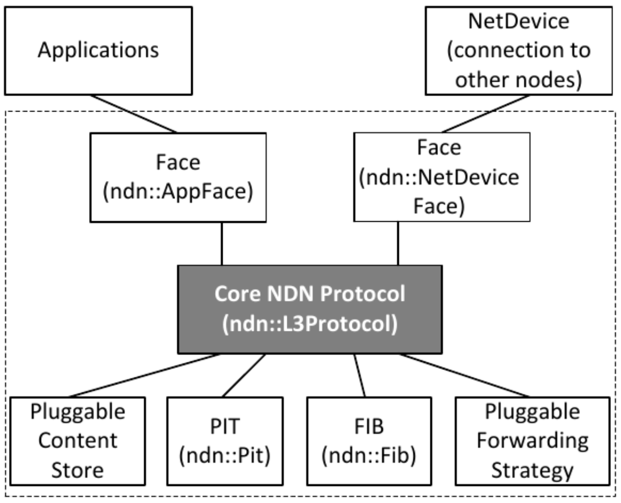

Applications
NetDevice (connection to other nodes)
Face (ndn::AppFace)
Face (ndn::NetDevice Face)
Core NDN Protocol (ndn::L3Protocol)
Pluggable Content Store
PIT (ndn::Pit)
FIB (ndn::Fib)
Pluggable Forwarding Strategy

Figure 3.1: Basic component abstractions in ndnSIM

The basic component abstractions are seen in figure **??**. The core NDN Protocol is the ndn::L3Protocol that receives interest and data packages from upper and lower layers through the corresponding faces. As of ndnSIM version 2.0 there are two distinct faces: the application face (ndn::AppFace) is responsible for inter-node communication between the application and the node itself while the net device face (ndn::NetDeviceFace) is responsible for inter-network communication with different nodes. Other faces for different purposes are expected to be added by the core developer or the community as needed. The CS (ndn::ContentStore) is an abstraction for in-network caching of data and can easily be omitted or replaced by another implementation of a different storing policy. The PIT (ndn::Pit) abstracts the data structure that is responsible to log all received interests with their nonce and incoming faces while the FIB (ndn::Fib) abstracts the data structure to guide the strategy in interest forwarding. The Strategy (ndn::ForwadingStrategy) is responsible to implement how interests and data are forwarded. That includes lookups in the content store for cached data, in PIT for already forwarded interests and in FIB if both previous searches didn't yield any matches. Each action in the forwarder of the strategy is represented as a virtual function in the forwarder header class and can be overwritten.

3.1.1 Face abstraction

The face abstraction plays an important role in the overall modularity of the NDN simulator by acting as an interface therefore making ndnSIM design independent from any underlying transport layers. All communication between application, the NDN core protocol and other nodes with the L3 protocol happens through faces that take care of any needed conversion.

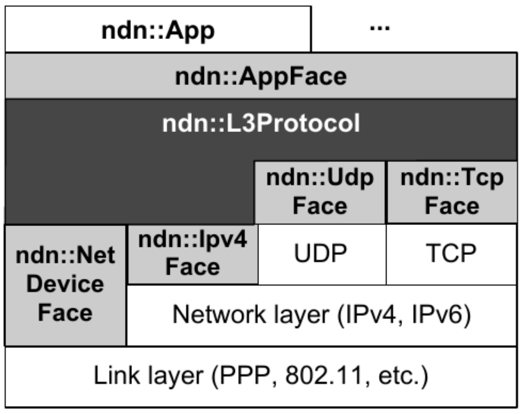


Figure 3.2: Face abstractions around ndn::l3Protocol

As shown in **??** the application can communicate with ndn::L3Protocol through the AppFace whereas for inter-node transmission it depends on the transmission protocols which faces need to be used. TCP and UDP have their respective ndn::UdpFace and ndn::TcpFace. IPv4 and IPv6 also have their own ndn::Ipv4Face making it particularly easy to implement the NDN architecture on top of IP or even TCP/IP. If NDN needs to be implemented without TCP/IP protocol it can communicate directly with the link layer through the ndn::NetDeviceFace.

3.1.2 Content Store abstraction

The CS is crucial for the NDN Internet architecture as it caches data for later use in potentially all the intermediate nodes. It can do rudimentary error recovery and multicast the data asynchronously downstream to the requesters. The replacement policy determines what data is saved into cache and how it gets replaced or deleted after a certain time. Currently implemented versions of the CS support Least Recently Used (ndn::cs::Lru), First In First Out (ndn::cs::Fifo) and a Random Replacement Policy (ndn::cs::Random). Each implementation is based on a dynamic trie-based data structure with hash-based indexing as are the PIT and FIB implementations.

3.1.3 Pending Interest Table (PIT) abstraction

The PIT data structure keeps information about each forwarded interest. Each PIT entry is uniquely identified by the content name of the interest. It holds a list of all incoming faces on which the interest was received and a list of all outgoing faces that the interest has been forwarded to. The arrival and expiration time are also kept in order to retransmit a lost interest. The nonce of an interest is a randomly generated number that is attached to the interest and identifies the interest in order to avoid loops. Nonce and Interest name uniquely identify the interest. Different consumers issuing the same interest will very likely have different nonces. In that case, no loop is detected and the incoming face is added to the already existing PIT entry. If the same nonce and interest are received again, the interest has looped and should be dropped.

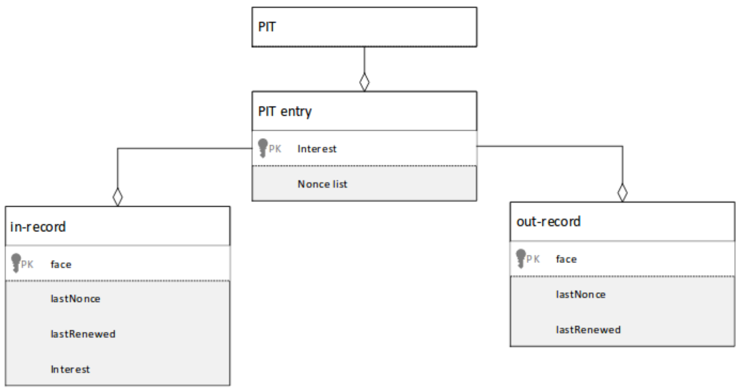


Figure 3.3: PIT data structure

Figure **??** shows the different PIT classes and how they relate to each other. There are also two additional timers on every PIT entry that are not explicitly shown here. The *unsatisfy timer* is the lifetime of an interest and counts down. If it reaches 0 the PIT entry has expired and needs to be forwarded again. The *straggler timer* starts counting down as soon a interest got rejected or satisfied. The straggler timer gives the node the time to detect further loops by the satisfied or rejected interest still floating within the network. Also, data plane measurements of returning data might require to still finish obtaining data points just after that event. Deleting the PIT entry immediately, there would be no means to detect loops and acquire valuable measurements [**?**]. After the straggler timer runs out the PIT entry is deleted. The PIT abstraction provides basic functions to insert, lookup, delete PIT entries and get measurements if necessary.

3.1.4 Forwarding Information Base (FIB) abstraction

The FIB guides the forwarding strategy in making decisions about Interest forwarding. It is similar to an IP's FIB but contains name prefixes instead of IP prefixes and holds several interfaces (out-faces) therefore enabling multicast forwarding. The faces are ordered according to their cost (routing metric) putting the cheapest connections at the beginning of the list. The lookup is performed on the content name prefixes. The longest prefix match yields the requested FIB entry with its outgoing face(es).

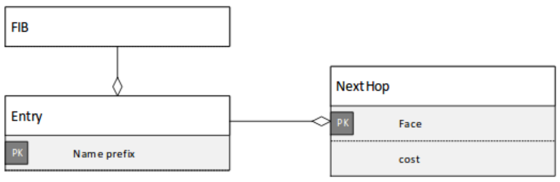


Figure 3.4: FIB data structure

As shown in Figure **??** each FIB entry that has been identified by longest-prefix match has an aggregated collection of NextHops. The NextHop collection must be non-empty and for every

NextHop there is one outgoing face towards a possible content source. There may only be one NextHop with a specific face id. The FIB abstraction provides basic insertions, deletions and exact match operations.

3.1.5 Forwarding Strategy abstraction

NdnSIM's modular architecture allows to experiment with different types of forwarding strategies without having to adapt any core components. The forwarder class interacts with the strategy and has the following important functions:

- *onIncomingInterest*: checks for localhost violations and if the interests have looped. Then the function inserts or updates the PIT, resets its timers and looks if the interest can be satisfied by cached data.
- *onContenStoreMiss*: if there is no data to satisfy the interest, the strategy is called.
- *onOutgoingInterest*: this function is called by the strategy and forwards the interest to the outgoing faces.
- *onIncomingData*: checks for localhost violations and if there are any PIT entries for that data. If there are no PIT entries the data is dropped, otherwise the data will be handed on to the onOutgoingData function.
- *onOutgoingData*: forwards the data downstream towards the content requester.

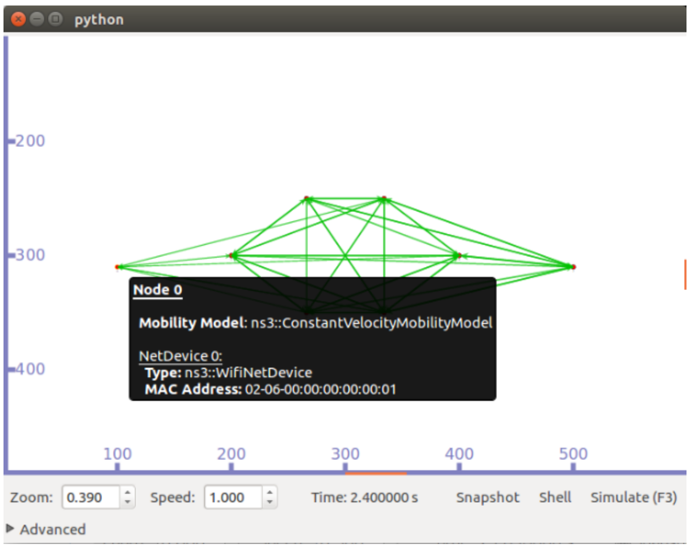
The strategy is called on three occasions. The first time from the forwarder::onContentStoreMiss(), when the decision has to be made how to forward the interest upstream. The second time from the forwarder::onContentStoreHit() to decide how to proceed with an satisfied interest. And the last time from the forwarder::onIncomingData()

The strategy has one important function:

- *afterReceiveInterest*: it receives the interest and the corresponding FIB and PIT entries from the forwarder and decides how to forward them further and through which faces.

3.2 Simulation Environment

The NS-3 simulator supports several simulation environments. In this thesis, NS-3 PyViz was used as a live simulator. PyViz also has an interactive console that visualizes the connections between the nodes of the simulation. It also can be used to debug the state of the running object, show PIT and FIB entries in live and where packages are being dropped.



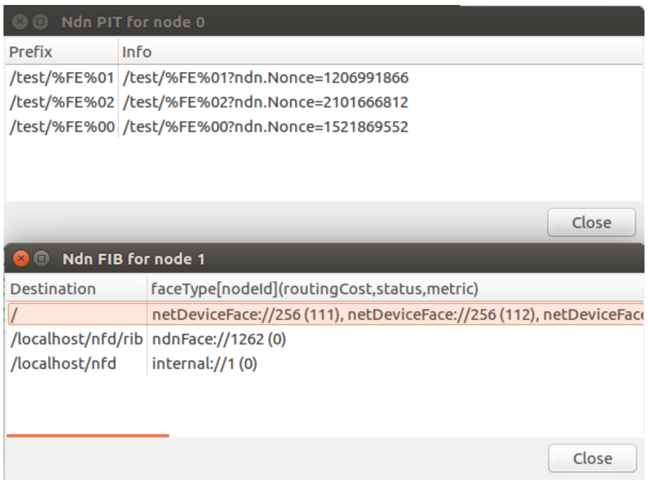
Node 0

Mobility Model: ns3::ConstantVelocityMobilityModel

NetDevice 0:
Type: ns3::WifiNetDevice
MAC Address: 02-06-00:00:00:00:00:01

Figure 3.5: Simulation of a possible scenario

Figure **??** shows a current scenario in development. Zoom and speed can interactively be adjusted. The simulation can be started and paused at any time. Node specific information can be shown while hovering over the node. Traffic between the nodes is visualized by green lines. The current utilized bandwidth is mentioned just above the traffic lines in a live simulation.



Ndn PIT for node 0

Prefix	Info
/test/%FE%01	/test/%FE%01?ndn.Nonce=1206991866
/test/%FE%02	/test/%FE%02?ndn.Nonce=2101666812
/test/%FE%00	/test/%FE%00?ndn.Nonce=1521869552

Close

Ndn FIB for node 1

Destination	faceType[nodeId](routingCost,status,metric)
/	netDeviceFace://256 (111), netDeviceFace://256 (112), netDeviceFac
/localhost/nfd/rib	ndnFace://1262 (0)
/localhost/nfd	internal://1 (0)

Close

Figure 3.6: PIT and FIB entries in real time

Figure **??** shows all the PIT and FIB entries in real time. Face id's with the corresponding face type are also shown for easier debugging purposes.

3.3 Forwarding by the NDN Forwarding Daemon (NFD)

All the above mentioned modular components and abstractions are implemented and coordinated by the NDN Forwarding Daemon (NFD), which is a network forwarder that implements and evolves with the NDN protocol. NFD is responsible to correctly forward interest and data packages, maintain all basic data structures like CS, PIT and FIB, and implement the packet processing logic. Management interfaces are used by NFD to configure, control and monitor the different components.

Packet processing in NFD is accomplished through forwarding pipelines and forwarding strategies. A forwarding pipeline is an aggregation of different steps that are applied on a packet or a PIT entry. Forwarding pipelines are triggered by specific events like the reception of an Interest, detection of a loop, of when an interest is ready to be forwarded further, etc. A forwarding strategy is attached at the beginning or the end of a pipeline and supports it with information about how and when to forward the interest. Figure **??** shows the interactions between forwarding pipelines and strategies. For the thesis relevant pipelines will be shortly described.

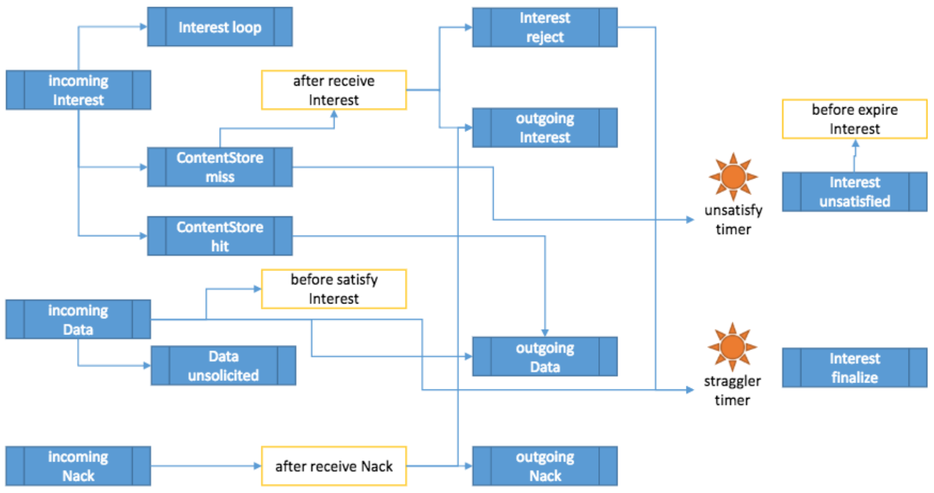


Figure 3.7: Pipelines and Strategies, white boxes are decision points inside the strategy while blue boxes represent the different pipelines

Forwarding pipelines operate on interests, data and nacks (negative acknowledgments send downstream to inform about failure to satisfy a particular interest). After a pipeline has finished it's processing the package is handed off to another pipeline or to a decision point inside the strategy until all processing is finished. NFD uses three main forwarding paths:

- *Interest Processing Path*
- *Data Processing Path*

- *Nack Processing Path*

The first two paths will be explained further. The nack processing path has not yet been implemented in ndnSIM 2.0 and is under active investigation, therefore it will not be explained.

3.3.1 Interest Processing Path

The interest processing path consists of the following pipelines:

- *incoming interest*: Interest has been received through a face and is ready to be processed by the forwarder
- *interest loop*: loop has been detected and interest needs to be dealt with
- *content store miss*: incoming interest can't be satisfied by the content store
- *content store hit*: incoming interest can be satisfied by CS and does not need further forwarding
- *outgoing interest*: interest is made ready and sent out
- *interest reject*: strategy rejects interests according to PIT entries
- *interest unsatisfied*: processing unsatisfied PIT entries before timeout
- *interest finalize*: deletion of PIT entries

Incoming Interest Pipeline

After receiving an interest package through a face implemented in `Face::onReceiveInterest` the interest is forwarded to the incoming interest pipeline that is implemented in `Forwarder::onIncomingInterest` method. The input parameters to this method are the interest itself and a reference to the face it was received on. Figure **??** shows all the steps taken within the pipeline.

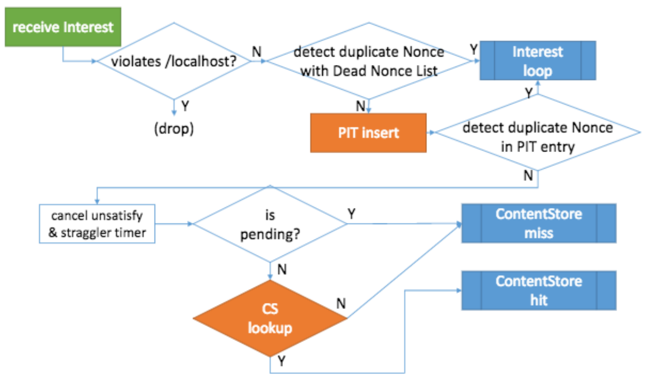


Figure 3.8: Pipelines and Strategies, white boxes are decision points inside the strategy while blue boxes represent the different pipelines

The following is a short step by step description of the pipeline in figure **??**:

1. An interest reaching the node through a non-local face is not allowed to have the `localhost` prefix since this prefix is reserved for localhost communication only. This check is necessary because interests with the `localhost` prefix can be used to configure the node and make unwanted changes. The compliant forwarder has no reason to send such an interest to a non-local face.

2. Next the name of the interest with its nonce is checked against the dead nonce list. That happens before the PIT entry is created for the incoming interest (supplements the regular nonce checking through PIT entries). If the tuple of interest and nonce have been processed already, the incoming interest has been looped and will be handed off to the interest loop pipeline.

3. Using the name and the selectors of the interest the PIT is searched for matches. If no match can be found a new PIT entry is created, otherwise an existing PIT entry is updated with the new information mainly the inFace and timers. In Figure **??** of chapter 2 the CS was checked first, then the PIT was searched and created if necessary. NFD has a slightly different approach. It is expected that the CS will be much larger than the PIT. A search in the CS may take much longer than a PIT lookup, and if a pending interest is present the CS lookup can be omitted saving time and resources.

4. The nonce of the interest is checked against the nonces in the PIT and if a loop or multi-path arrival is detected the interest will be handed off to the interest loop pipeline. If no loop or multi-path arrival is detected the forwarding process is continued.

5. Since a new valid interest arrived and the PIT entries got updated, the current unsatisfy timer (fires when the PIT entry expires) and the straggler timer (fires when PIT entry is no longer needed after it has been satisfied or rejected) are cancelled.

6. This step checks if the interest is pending (if the PIT entry has already another in-record from the same or different face).

7. If there is no PIT entry a CS lookup is needed in order to satisfy the interest (ContentStore hit) or forward it (ContentStore miss). If there is already a pending interest the CS lookup has been done already on a previous interest that led to a PIT entry since no match could be found.

ContentStore Miss Pipeline

If the ContentStore lookup in `Forwarder::onIncomingInterest` doesn't yield any data that can satisfy the interest, the content store miss pipeline is called. This pipeline is implemented in `Forwarder::onContentStoreMiss` and the parameters for this method are the interest packet, the incoming face and the PIT entry. The validity of the interest has been checked already and it needs to be forwarded by taking the following steps:

1. In the incoming interest pipeline a PIT entry was created already. Now the in-record with its incoming face needs to be added to the PIT entry and if it already exists then the in-record with its face is updated. The expiration counter is set to the value of the interest's field `InterestLifetime`.

2. The PIT entry's unsatisfy timer is set. It will expire as soon as all in-records of this entry have expired and hand control off to the interest unsatisfied pipeline.

3. The content store miss pipeline then decides which strategy to use and forwards the interest with its incoming face and PIT entry to the chosen strategy for further processing.

The incoming interest pipeline has finished.

ContentStore Hit Pipeline

If the ContentStore lookup in `Forwarder::onIncomingInterest` does yield a match, the content store hit pipeline is called through its implementation in `Forwarder::onContentStoreHit`. The parameters include the interest packet, incoming face, the PIT entry and the data. The straggler timer is being set since the interest is about to be satisfied. After that the data is handed off to the outgoing data pipeline.

The incoming interest pipeline has finished.

Outgoing Interest Pipeline

The `Forwarder::onOutgoingInterest` method implements the outgoing interest pipeline. It is entered from the `Strategy::sendInterest` or a child of the Strategy class that overrides the method. The parameters are the PIT entry, the outgoing face and a `wantNewNonce` boolean that signalizes that a new nonce is needed for the interest. As mentioned above the PIT entry holds a reference to the interest and therefore the interest packet is not in the parameters list. Figure **??** illustrates the outgoing interest pipeline.

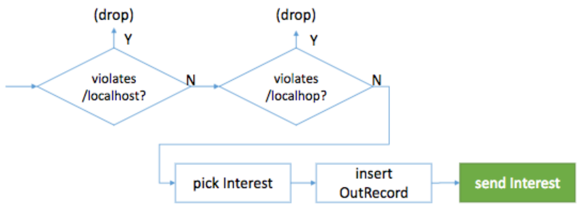


Figure 3.9: Outgoing interest pipeline

1. First localhost and localhop violation is checked. Interests with a /localhost scope are not permitted to be sent out through non-local faces. Interests with a /localhop scope are permitted to be sent out through a non-local face only if the PIT entry has one or more in-records with that has a local face stored in it.

2. The interest is picked from the PIT entry and saved to a local variable.

3. If the `wantNewNonce` flag is set, the interest is copied and a new nonce is attached to it. This is necessary when the node needs to retransmit the interest. If the interest wouldn't receive a new nonce it would be recognized as a looped interest and dropped immediately.

4. An out-record is created in the PIT entry and add an out-face to it. If an out-record with the same out-face already exists, the entry gets refreshed by the `InterestLifetime` value.

5. The interest is forwarded upstream via out-face.

3.3.2 Data Processing Path

The data processing path consists of the following pipelines:

- *incoming data*: incoming data are processed
- *data unsolicited*: unsolicited (not requested) data are processed
- *outgoing data*: data is forwarded downstream

Incoming Data Pipeline

`Face::onReceiveData` fires when an data arrives at a face. It triggers the `Forwarder::onData` which calls `Forwarder::onIncomingData` with the incoming face and the data as parameters. The following figure **??** summarizes the steps taken by this pipeline:

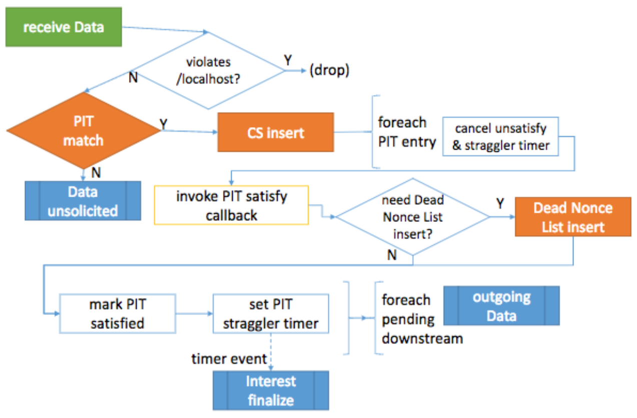


Figure 3.10: Incoming data pipeline

1. Like with the incoming interest pipeline the /localhost scope needs to be checked. Data may not contain the prefix /localhost when coming from a non-local face.

2. The incoming data needs to be checked against the PIT entries to find out if it is solicited or unsolicited. The data is unsolicited if no matching PIT entry is found. In this case, it is forwarded to the next pipeline: data unsolicited.

3. If one or more PIT entries have been found the data is solicited and therefore inserted into the CS. How the data is processed in the CS is up to the policies defined in the content store.

4. The unsatisfy and straggler timers for all matched PIT entries need to be cancelled since the interest is being satisfied.

5. The responsible strategy is determined and triggers the method `Strategy::beforeSatisfyInterest`. By default, it is empty in ndnSIM 2.0 and can be implemented by a custom strategy overriding the method.

6. The nonce of the data is added to the dead nonce list for further loop detection, since interest and data could still be floating inside the network. This is necessary because the next steps delete the corresponding in- and out-records of the PIT entry and the nonce would be lost.

7. The PIT entry is marked satisfied. All in-records and out-records (corresponding to the incoming face of the data) are being deleted.

8. For every pending downstream the data is handed off to the next pipeline, the outgoing data pipeline, with its corresponding face. NFD takes care that the data is handed off only once for every distinct downstream, even if several PIT entries are found.

Outgoing Data Pipeline

The outgoing data pipeline is implemented in `Forwarder::onOutgoingData` and can be called from two different pipelines. If the incoming interest pipeline found a CS match, the data is fetched and passed to `Forwarder::onOutgoingData` with the corresponding out-face (the interest's in-face in the PIT entry). If the incoming data pipeline found one or more PIT matches, the data and out-face are passed to this method.

This pipeline has three steps:

1. The localhost scope is checked. Data with the prefix /localhost are not permitted to be send out through a non-local face.

2. This step is reserved for traffic management and traffic shaping but not implemented in ndnSIM 2.0 yet.

3. The data packet is send out through it's out-face.

Chapter 4

Design and Implementation

The main goal of this thesis is to extend the current ndnSIM 2.0 implementation in order to allow efficient communication between mobile agents. A first interest packet is broadcast by the consumer on one channel and forwarded by all intermediate nodes in a broadcast manner. Default routes are set by the NFD at the beginning and need to be overridden as soon as more information is obtained. (TODO: need to check how it is on base) Looping interests need to be identified and dropped in order to reduce bandwidth waste. After being forwarded by the intermediate nodes the interest arrives at the producer and a corresponding data packet is created. This packet returns to the consumer and will add routes by updating the FIB entries.

4.1 Problem Description

The current implementation relies only on face id's of the application or net device (`ndn::NetDevice`) which is sufficient for point to point connections but will not work on wireless communication like in the mobile ad-hoc scenarios investigated in this thesis. Wireless communication is inherently broadcasting and the nodes are responsible to accept the package if it was meant for them and drop it if it was only overheard. There cannot exist a ”route” by defining the in-faces of an interest and forwarding it to out-faces obtained from the FIB entries. The same goes for data being forwarded through faces downstream towards a consumer. The topology is not known and changing fast so there needs to be a way to identify specific nodes in a consistent manner.

If one route is found and the FIB entries updated, interests will follow this path to the producer and data will follow the breadcrumbs back to the consumer. That is likely to lead to congestions and incur unnecessary overhead on some of the intermediate nodes while leaving others out of the information flow all together. Having the possibility to chose from different channels and different paths will spread the load onto different intermediate nodes making congestions and retransmissions less likely.

4.2 Multipath Approach

In order to support paths the nodes must know where the packet did come from and for what node is was meant. The faces of the net device cannot be used since they are equal on each node. Every node assigns number id's from 256 to each face. All the nodes will have at least face id 256. Also the application faces will be the same on the consumer and the producer. That can be achieved by using the NDN's net device (`ndn::NetDevice`) mac addresses that are unique for each simulation.

4.2.1 Changes to the Interest

In order to have routes and the FIB entries populated, the information about the topology must be communicated to all the surrounding neighbours. The information about previous nodes that have been encountered while forwarding the interest towards a potential content source can be added to the interest. That makes it possible to forward the data downstream towards the requester by following the breadcrumbs left in the PIT entries. These breadcrumbs consist in our implementation of the mac addresses of the net device from the previous node. The interest has three new fields:

- *std::string m_interestOriginMacAddress*: This field has the mac address of the previous node.
- *std::string m_interestTargetMacAddress*: This field has the mac address of the next node if there is a FIB entry to get the target mac from.
- *std::string m_macInterestRoute*: This field shows the the way of the interest so far.

The following figure **??** illustrated the mac address relevant steps in the incoming interest pipeline `Forwarder::onIncomingInterest`.

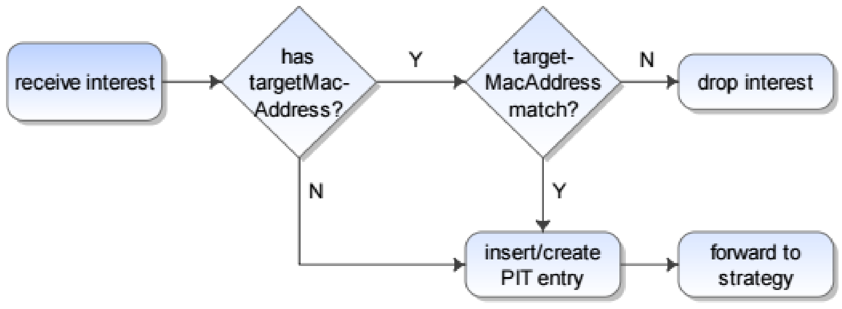


Figure 4.1: This figure shows how mac addresses are added and checked within the incoming interest pipeline

At the beginning of the simulation no knowledge about the topology is present. The first interest is broadcast. Since the FIB entries have not yet been populated with information on neighbouring nodes and their mac address, the first interest cannot have a valid targetMacAddress. The missing mac address signals to all receiving nodes that the interest is being flooded through the network and needs to be broadcast further. After information about the topology is present the interest will have a valid mac address of the target it is supposed to reach.

The steps in figure **??** are the following:

1. After the node receives the interest, the *Interest::m_interestTargetMacAddress* is checked for a valid mac address through regular expressions.

2. In the case the interest has no valid target mac address the interest is accepted as broadcast the interest will be inserted into the PIT table with it's origin mac address.

3. If a valid mac address was found, it is checked against the receiving net device's mac address.

4. If the interest's target mac address equals the receiving net device's mac address the interest was meant for this node (in particular to this net device) and will be inserted into the PIT table with it's origin mac address. If the two mac addresses (target mac address and the current device's mac address) do not match the interest was not meant for this node and will be dropped immediately to save resources. No overhearing occurs at any time.

5. The interest will be forwarded to the strategy for further processing.

All interests have valid *std::string m_interestOriginMacAddress* fields at all time. They are needed to leave the breadcrumbs for the returning data and are inserted into the PIT entries at time of arriving at a new node. If the incoming interest had a valid target mac address, it will become the new (current) mac address of the outgoing interest. If no valid target mac was given to the incoming interest a new current mac address must be chosen from all active net devices. This is done through a static counter and the modulo expresion, making sure, that differenent paths are taken. Adding the new origin mac address happens in `Forwarder::onOutgoingInterest` or in `Strategy::afterReceiveInterest`.

4.2.2 Changes to the Data

In order to have routes and the FIB entries populated, the information about the topology must be communicated to all the surrounding neighbours. The data is responsible to populate the FIB entries with this new information and adding mac addresses. Existing FIB entries can be updated or new routes can be added with the help of the `ns3::ndn::FibHelper` helper class by calling the `(`AddRoute`)` method.

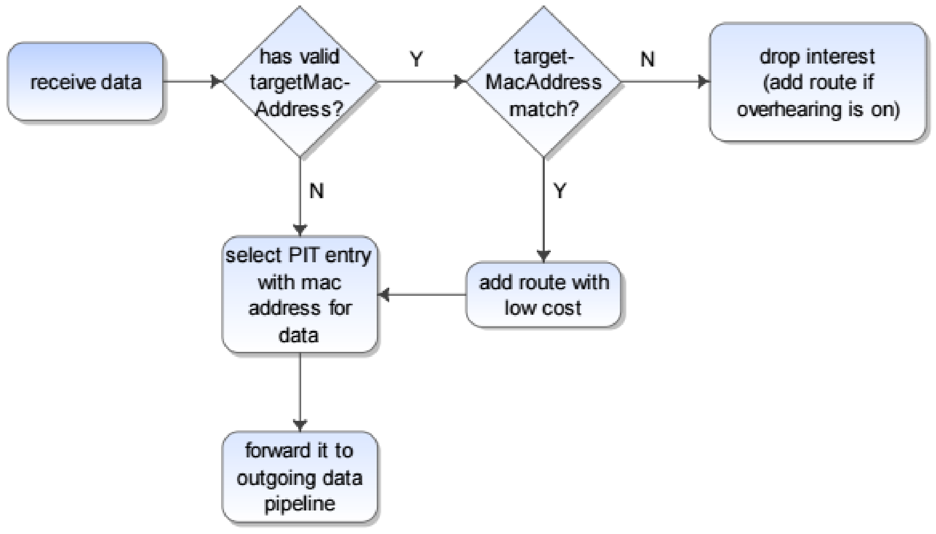
The data is also extended with three new member variables:

- *std::string m_dataOriginMacAddress*: This field has the mac address of the previous node.

- *std::string m_dataTargetMacAddress*: This field has the mac address of the next node if a mac address was left in the PIT entry.

- *std::string m_macDataRoute*: This field shows the the way of the data so far.

The following figure **??** illustrated the mac address relevant steps in the incoming data pipeline `Forwarder::onIncomingData`.



receive data
has valid targetMac-Address?
Y
target-MacAddress match?
N
drop interest (add route if overhearing is on)
N
Y
select PIT entry with mac address for data
add route with low cost
forward it to outgoing data pipeline

Figure 4.2: This figure shows how mac addresses are added and checked within the incoming data pipeline

The steps are explained as follows:

1. After the node receives the data, the *Data::m_dataTargetMacAddress* is checked for a valid mac address through regular expressions.

2. If the target mac address of the data is equal to the current net device's mac address the data was meant for this node. If the mac addresses do not match, the data was overheard.

3. It depends on the implementation if overheard data shall be further processed, cached into CS or just dropped. If it is saved, the route from where it came will be added to the FIB entries for later use.

4. If the data was meant for this node a route is added with low cost, in order to prfioritize the route.

5. From the corresponding PIT entry mac address and face information are retrieved and the added to a copy of the data.

6. The updated copy of the data is then handed off to the outgoing data pipeline.

The outgoing data pipeline is implemented in `Forwarder::onOutgoingData`. It receives the data with additional parameters. If the target mac address of the data is valid and the data is solicited, the origin mac address field of the data is updated with the correct mac address. If the data was unsolicited a new mac address of the possible net devices is selected in an alternating manner and added to the data. The route is updated and the data is send through it's out-face to the next node.

The data encoding of the interest and data packets had to be extended by the new fields in order to be able to be send over the network and receive by other nodes.

4.2.3 Changes to the FIB

The FIB entries hold information about how to forward the interest upstream. Every FIB entry is uniquely identified by the prefix of the incoming interest. It is found by longest prefix match. The entry aggregates NextHop objects with the corresponding faces and cost information as shown in chapter 3 figure **??**. In this thesis it is not done by faces like initially implemented, but by mac addresses. These mac addresses are updated inside existing FIB entries or new FIB entries are created with this additional information. Therefore the new FIB entries have been extended with functions to fetch the new information from the NextHop entries. And the NextHop entries have been extended by a new mac address field.

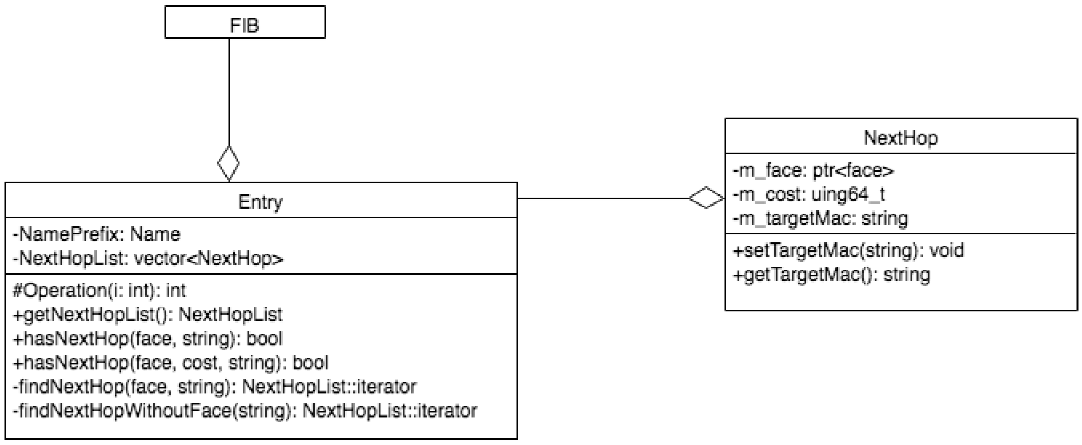

FIB
Entry
-NamePrefix: Name
-NextHopList: vector<NextHop>
#Operation(i: int): int
+getNextHopList(): NextHopList
+hasNextHop(face, string): bool
+hasNextHop(face, cost, string): bool
-findNextHop(face, string): NextHopList::iterator
-findNextHopWithoutFace(string): NextHopList::iterator
NextHop
-m_face: ptr<face>
-m_cost: uing64_t
-m_targetMac: string
+setTargetMac(string): void
+getTargetMac(): string

Figure 4.3: This figure shows the new FIB data structure

Figure **??** shows only the relevant and new member variables and methods added to the data structure. The FIB entries need new methods that find next hops according to the mac addresses or a combination of mac addresses and faces. A new method returns a NextHopList that is passed by reference and not constant anymore so incrementing on the cost can be done directly on the data structure. The NextHop class has been extended by the target mac field, setters and getters.

4.2.4 Changes to the PIT

The PIT entries hold information about how to forward the data downstream. Every PIT entry is uniquely identified by the prefix of the incoming data. It is found by longest prefix match. The entry aggregates in-record and out-record objects with the corresponding faces lastNonce and lastRenewed information as shown in chapter 3 figure **??**. The faces need to be extended by mac addresses since the data is broadcast and the receiving node needs to identify if it is the intended recipient.

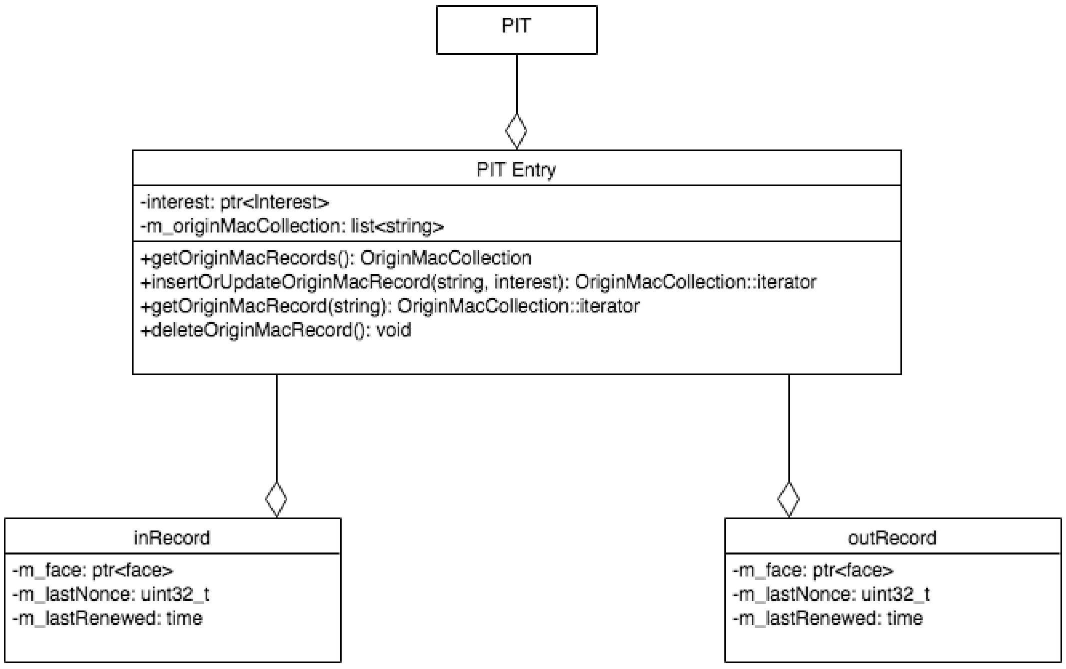

PIT
PIT Entry
-interest: ptr<Interest>
-m_originMacCollection: list<string>
+getOriginMacRecords(): OriginMacCollection
+insertOrUpdateOriginMacRecord(string, interest): OriginMacCollection::iterator
+getOriginMacRecord(string): OriginMacCollection::iterator
+deleteOriginMacRecord(): void
inRecord
-m_face: ptr<face>
-m_lastNonce: uint32_t
-m_lastRenewed: time
outRecord
-m_face: ptr<face>
-m_lastNonce: uint32_t
-m_lastRenewed: time

Figure 4.4: This figure shows the new PIT data structure

Figure **??** shows only the relevant and new member variables and methods added to the data structure. A new list member variable `m_originMacCollection` has been added to the PIT entry. Every interest forwarded correctly leaves it's origin mac address in this list. When requested data comes it checks the origin mac collection in the PIT entry and attaches it to the data's field. Methods, setters and getters encapsulate the member variable within it's class.

4.3 Retransmissions

(TODO: every retransmission should be broadcast in order to tackle the ad-hoc component)

4.4 Mobility

explain trace-files....

Chapter 5

Evaluation

Chapter 6

Conclusion

6.1 Summary and Conclusion

6.2 Future Work

Chapter 7

Appendix

Bibliography

[1] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael Plass, Nick Briggs, Rebecca Baynard, *Networking Named Content*, Parc, Paolo Alto, 2009

[2] Dnyanada P. Arjunwadkar *Introduction of NDN with Comparison to Current Interet Architecture based on TCP/IP*, In International Journal of Computer Applications, 2014

[3] Marica Amadeo, Claudia Campolo, Antonella Molinaro, *Forwarding Strategies in Named Data Wireless Ad hoc Networks: Design and Evaluation*, In Journal of Network and Computer Applications, 2014

[4] Giuseppe Rossini, Dario Rossi *Evaluating CCN multi-path interest forwarding strategies*, Technical Report, Telecom ParisTech, 2012

[5] ”NFD Developer’s Guide”, October 04, 2016. [Online]. Available: https://named-data.net/wp-content/uploads/2016/10/ndn-0021-7-nfd-developer-guide.pdf

[6] Spyridon Mastorakis and Alexander Afanasyev and Ilya Moiseenko and Lixia Zhang *”ndnSIM 2: An updateded NDN simulator for NS-3”* Technical Report, NDN, November, 2016

[7] Fan Li, Yu Wang, *Routing in Vehicular Ad Hoc Networks: A Survey*, IEEE Vehicular Technology Magazine, June 2007

[8] Alexander Afanasyev and Ilya Moiseenko and Lixia Zhang *”ndnSIM: NDN simulator for NS-3”* Technical Report, NDN, October, 2012

[9] Marica Amadeo, Claudia Campolo, Antonella Molinaro, *Enhancing content-centric networking for vehicular environments*, University ??Mediterranea?? of Reggio Calabria, Italy, 2013

[10] Marica Amadeo, Claudia Campolo, Antonella Molinaro, *CRoWN: Content-Centric Networking in Vehicular Ad Hoc Networks*, IEEE Communications Letters, vol. 16, no. 9, September 2012

[11] Carlos Anastasiades, Jürg Weber, Torsten Braun *Dynamic Unicast: Information-centric multi-hop routing for mobile ad-hoc networks, Computer Networks*, Institute of Computer Science, University of Bern, Switzerland, 2015

[12] Cheng-Shiun Wu, Shuo-Cheng Hu, Chih-Shun Hsu *Design of Fast Restoration Multipath Routing in VANETs*, Department of Information Management, Taipei, Taiwan, 2011

[13] Lucas Wang, Ryuji Wakikawa, Romain Kuntz, Rama Vuyyuru, Lixia Zhang, *Data Naming in Vehicle-to-Vehicle Communications*, Computer Science Department, University of California, Los Angeles, 2012

[14] Lucas Wang, Alexander Afanasyev, Romain Kuntz, *Rapid Traffic Information Dissemination Using Named Data*, Toyota Info Technology Centre Mountain View, California, 2012

[15] Yu-Ting Yu, Yuanjie Li, Xingyu Ma, Wentao Shang, M. Y. Sanadidi, Mario Gerla *Scalable Opportunistic VANET Content Routing With Encounter Information*, University of California, Los Angeles, 2013

[16] "PyViz," https://www.nsnam.org/wiki/PyViz, 2017. [Online]. Available: https://www.nsnam.org/wiki/PyViz