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

School of Computer Science and Statistics

Practical Implementation of Named Data Networking for Internet of Things

Pratik Pathak

Supervisor: Dr. Stefan Weber

August 2022

A Dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Computer Science
(Future Networked Systems)

Acknowledgements

This project would not have been possible without the guidance of my supervisor Dr. Stefan Weber who patiently explained me even the most complex concepts and provided guidance throughout the course of my dissertation. I would also like to thank my family and all my friends for their continuous support during the course of my study at Trinity.

Pratik Pathak

University of Dublin, Trinity College
August 2022

Contents

Chapter 1 Introduction **1**
- Research Background 1
- Dissertation Map 3

Chapter 2 State of the Art **4**
- Internet of Things 4
  - Internet of Things Requirements 4
  - Communication Technologies 5
  - IoT Devices 6
  - IoT Platforms 7
    - Operating Systems 8
    - Development Frameworks 8
- Internet Protocol (IP) 9
  - Challenges with Internet Protocol in the Internet of Things 10
- Information-Centric Networking 11
  - Information-Centric Networking Features 11
  - ICN Architectures 12
  - Suitability for IoT 13
  - Comparision to Internet Protocol 14
  - How does NDN work? 15
  - NDN Deployment Alternatives 17
  - NDN Implementations 18
    - Emulators and Simulators 18
    - Core Implementations 19
    - IoT Implementations 19

Chapter 3 Design **22**
- Problem Description 22
- Proposal 23
- Architecture 23

Chapter 4 Technical Implementation **25**
- System Components 25
  - Internet of Things Devices 25
  - NDN Network 27
  - Gateway Node 29
- Project Steps 29
- Project Execution 30

Results & Evaluation 32
NDN IoT applications 32
NDN IP Inter-networking Solutions 33

Chapter 5 Conclusion and Future Work **35**
Conclusion 35
Future Work 36

Bibliography **37**

Appendix **39**

List of Tables

2.1	Comparison of Communication Technologies	7
2.2	Comparison of nRF52, ESP32 & Raspberry Pi	8
2.3	Comparison IP and ICN for IoT	14
2.4	Comparison of features of IP and NDN	15
2.5	Comparison NDN implementations for IoT devices	21
1	List of Abbreviations	39

List of Figures

1.1	Industries innovating in the Internet of Things	2
2.1	IP Packet Structure	9
2.2	IP vs NDN protocol stack	12
2.3	IP vs ICN content retrieval	14
2.4	NDN network Data Flow	15
2.5	NDN RIOT Architecture	19
3.1	Project Architecture	23
4.1	Esp Wifi module for IPv6 Address	25
4.2	Sensor as Data Producer in NDN network	25
4.3	IoT device sending Interest packets	26
4.4	NDN server sending back Data packets	26
4.5	IPv6 enabled NDN network with three containers	27
4.6	Publish & Fetch Data in the NDN network	28
4.7	The Gateway Node listening for incoming messages	28
4.8	IoT device notifies Gateway about new NDN name prefix	29
4.9	Propagating Gateway message in local NDN network	30
4.10	Package the NDN packet as UDP and forward to Gateway Node	30

Chapter 1

Introduction

In this section, we introduce the background necessary for the dissertation. It will describe the motivation for the choice of topic and the benefits of adopting a new Internet architecture for the Internet of Things domain. The chapter also explains the organisation of the dissertation document.

1.1 Research Background

The Internet of Things (IoT) consists of heterogeneous devices that can sense the environment and actuate based on events. They depend on various communication technologies like Bluetooth, Zigbee, Ethernet and Wi-Fi. The main goal of IoT by interconnecting devices, collecting and processing data is to understand it's surrounding continuously to make better decisions. The benefits of implementing IoT in sectors like healthcare, manufacturing, automation, frights, and smart cities are well known.

The Internet of Things (IoT) has not only been successfully implemented in various industries but the demand for this technology is also rapidly growing. The number of IoT devices has increased by around 18% this year [1] and it is expected that it would exceed 125 billion by 2030 [2]. On the contrary, IoT applications due to their nature face certain unique challenges [3] like naming, mobility, interoperability, security, energy efficiency and scalability. Hence, there is enormous research interest to solve these problems and make IoT networks more efficient.

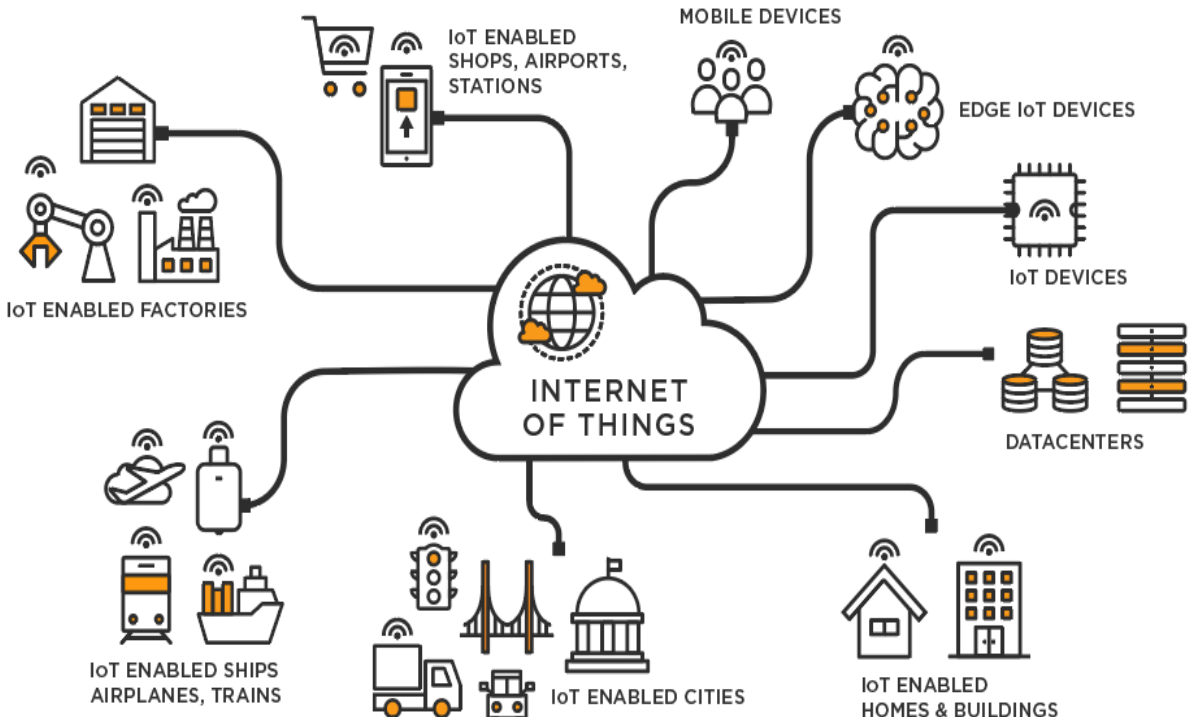

Figure 1.1: Industries innovating in the Internet of Things

The Internet Protocol (IP) was born in the 1970s as part of the monolithic Transmission Control Program developed by Vint Cerf and Bob Kahn. This was later broken down into Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) at the Transport layer and Internet Protocol at the Internet layer. TCP/IP later became the popular communication protocol for the internet as we know it today.

The domination of IP on the internet made it the default choice for developing IoT applications as well. As mentioned earlier, the current Internet architecture was developed many decades ago. Hence, it has issues adapting to the new requirements (as introduced in more detail in Chapter 2) that the Internet of Things era brings. Information-Centric Networking (ICN) is an alternative approach that looks to solve these problems by providing different features like the naming of data & services, distributed caching, object security, and decoupling of sender and receiver. Various ICN approaches are being researched that aim to overcome these challenges and improve the efficiency of these applications.

Named Data Networking (NDN) is the latest and most popular architecture from Information-Centric Networking that is being currently explored by the research community. Named Data Networking is a promising approach that addresses challenges intrinsic to IoT applications. This research will explore the state of the art of the current NDN implementations and will focus on developing a solution for current challenges in the application of Named Data Networking to low-powered IoT devices.

1.2 Dissertation Map

This dissertation is structured into 6 sections, each having its own focus area as described below.

- Chapter 1: Introduction – This chapter gives an introduction to the Internet of Things application requirements and points out that the Internet Protocol faces a lot of challenges to adapt to these requirements. It also presents Named Data Networking which is a new proposed Internet Architecture as an alternative to the existing Internet Protocol.
- Chapter 2: State of the Art – This chapter introduces the requirements of the Internet of Things and the mismatch of Internet Protocol to satisfy them. We then look at Named Data Networking as an alternative and discussion the current implementations and deployment options.
- Chapter 3: Design – This chapter will list the problems for practical implementation and adoption of Named Data Networking and then propose a solution for the same. It will then describe the architecture of the project leading to the implementation in the next chapter.
- Chapter 4: Technical Implementation – This chapter will describe all the low-level components of the project and their setup in detail. It will then demonstrate the working of the project step-by-step and discuss the results by comparing them with various existing solutions.
- Chapter 5: Conclusion and Future Work – Lastly, this chapter will conclude the learnings and contributions from the prototype developed in the dissertation and discusses the future scope for improving the solution.

Chapter 2

State of the Art

In this section, we will detail the enablers of the Internet of Things technology and the current challenges in the implementation and deployment of IoT networks. We will then present a case for Named Data Networking which is a new research area focused on improving the existing Internet architecture.

2.1 Internet of Things

The Internet of Things has already proved its mettle and is adopted by large companies in various industries. These companies have realised utilizing the potential of IoT devices will not only make their existing processes efficient but also unleash new possibilities for business growth. Consumers have also experienced the benefits of using IoT in their daily life to make tasks simpler and thus are embracing this change. As a result, the number of IoT devices is growing rapidly which brings new challenges for implementing various kinds of IoT networks to the forefront. The growth of IoT technology also tests the current IP networking and highlights the mismatch between IP's host-centric design and the needs of IoT applications.

2.1.1 Internet of Things Requirements

Let us look at some requirements of IoT networks to understand the challenges [3] that need to be addressed to scale these networks for practical use.

Naming and addressing – The number of IoT devices connected to the network can be billions and still grow. This makes uniquely naming them an arduous task. The rate at which they produce data is also very high and addressing these data for routing purposes is another issue. This problem could partially be solved by a very large address space but is again challenging to implement without incurring a large overhead. Thus, a pragmatic, secure and scalable approach to naming is necessary.

Scalability – When billions of IoT devices must connect to the internet, the network must still be able to support operations with the desired latency. We need to think not only about the naming of the devices but also about managing and routing the huge amount of data

produced by these devices. The solution to this problem lies in improving the network efficiency to reduce the amount of work that needs to be done.

Mobility – In an IoT environment, the devices are mobile and require reliable connectivity even when the sensors are moving. For example, a vehicle-to-vehicle IoT application will be highly mobile and the network needs to adapt to this. To achieve this, the network should be able to serve data efficiently by making it available at different locations.

Heterogeneity and Interoperability – The IoT network is composed of different kinds of devices such as sensors, RFID tags, actuators and smartphones working together to solve a single problem. They are all heterogeneous in terms of processing power, storage, battery power, wireless support, etc and still have the requirement to communicate with each other seamlessly. This is difficult to achieve since there is no standard that could be implemented across these devices.

Security and Privacy – Some IoT applications may deal with a huge amount of sensitive data and thus need to be very careful about the security requirements and privacy concerns. IoT data is also vulnerable to new types of attacks not only because of their highly mobile nature but also lack of a standard method to provide and update security systems for all the heterogeneous devices. Network security if implemented correctly can prevent various common types of security attacks.

Energy Efficiency – Many IoT devices are low-powered because of mobility as well as budget constraints. They may be battery operated or need some continuous source of energy. Thus, it is critical to optimise the energy consumption of the devices. When a large amount of data flows through the network, it will increase the energy consumption of the devices. Thus, a network that requires less overhead for connectivity and data transfer is ideal for IoT environments.

2.1.2 Communication Technologies

The idea of the Internet of Things is based on connectivity between different devices and so various communication technologies [4] play an important role in any IoT application. We will take a look at the most popular options that are available and their suitable use cases. They can be divided into categories by the range of their communication as below:

1. Wired

Ethernet – Ethernet is based on the IEEE 802.3 standard and is a quick and easy way to connect devices to the internet. It uses fibre optic, co-axial or twisted pair cables in a Local Area Network (LAN) for connectivity. The advantages of Ethernet include low latency and less complexity. Hence, an IoT system which is co-located and works on a mission-critical

application can use this technology.

2. Short-range Wireless

Bluetooth Low Energy (BLE) – Bluetooth is a wireless technology meant for short-range data exchange. Bluetooth Low Energy (BLE) is the latest standard which is part of Bluetooth 4.0. As the name suggests, it dramatically lowers the consumption of energy by the devices as compared to previous Bluetooth versions. This is especially suitable for low-powered IoT devices that require limited mobility.

Zigbee – Zigbee is based on IEEE 802.15.4 standard and on a communication protocol that is mainly used for Personal Area Networks. It was created to suit the need for low-cost communication, especially in Personal Area Networks. IoT applications which require low data transfer rates, reduced power consumption and secure networking are the best use cases for Zigbee technology.

RFID – RFID is an automatic identification technology that allows for reading and writing data onto RFID tags with the tap of the reader. The RFID tag has a universally unique identification code that can be read in the vicinity of the reader. There are two types of RFID tags - active and passive. The active tags are more expensive, battery-powered and used for advanced applications. Passive tags are the most widely used form of RFID and have found their application in various sectors like retail for identification & checkout, home security, and road toll management, etc.

NFC – NFC is very similar to RFID technology and uses radio frequency for data transfer between devices in proximity. NFC is extremely simple to use and has found its application in smartphones, contactless payment systems and industrial applications. The requirement for the devices to be very close like only a few centimetres away acts as a shield and is advantageous for security reasons. Moreover, it also supports encryption which can be useful for sensitive data.

Wi-Fi – Wi-Fi is a wireless local area network (WLAN) protocol and is based on IEEE 802.11 standard. It is very convenient and the range can be customized with the use of a repeater. It can also provide very high network speeds but that comes at a cost of increased power consumption. Also, security is a concern when using Wi-Fi as connecting devices to unknown networks is like providing an easy way for hackers to obtain sensitive data.

3. Longe-range Wireless

Low Power Wide Area Network (LPWAN) – Wireless networks that are designed to allow long-range communication at a low data rate, reducing power and cost for transmission. Examples of these are SigfFox, NB-IoT, etc. SigFox is a low-power technology for wireless

communication of a diverse range of low-energy objects such as sensors and M2M applications. It allows the transportation of small amounts of data ranging up to 50 kilometres. Narrowband Internet of things (NB-IoT) is a radio technology standard developed for cellular devices and services.

Specification	Medium	Benefits	Use case	Drawbacks
Ethernet	Wired	Low Latency, Easy	Mission critical, Co-located	Cables needed
BLE	Short Range Wireless	Low power consumption	IoT apps with less mobility	Short range, Mobility constraint
Zigbee	Short Range Wireless	Low-cost	Personal Area Networks	Limited Speed
NFC	Short Range Wireless	Easy to use, Secure	Contactless Payment systems	Expensive, Low Speed
RFID	Short Range Wireless	Security	Retail, Home Security	High Cost
Wi-Fi	Short Range Wireless	Convienent, Extensible, High Speeds	Practically everywhere	Cost, Security
LPWAN	Long Range Wireless	Long Range, Low power consumption, Cost Efficient	Environmental monitoring, Smart Healthcare	Low data rate

Table 2.1: Comparison of Communication Technologies

2.1.3 IoT Devices

The increasing popularity of the Internet of Things is aided by the variety of IoT devices made available by manufacturers. In this section, we will look at different IoT devices and compare their features to select a suitable device for developing this project.

Arduino is one of the popular options for low-powered IoT projects and the price is also affordable. But, it lacks inbuilt features like Wifi and Bluetooth which are used in a lot of the projects. This can be overcome by an add-on Ethernet shield. On the other hand, Raspberry Pi is a powerful device with a lot more RAM and can also run operating systems like Linux smoothly. This makes it useful for advanced applications that cannot be run on low-powered devices. For most IoT projects, so many resources are overkill and can increase resource consumption as well as escalate costs. nRF52 is another low-powered device but again does not come with Wifi capabilities and is also costlier than other options. Lastly, the cheapest devices that support Wifi and Bluetooth come from the Espressif family like the ESP32 board. We summarize the specifications of all these devices in Table 2.2

Specification	**nRF52**	**ESP32**	**Raspberry Pi**	**Arduino Zero**
SOC/ MCU	ARM®Cortex®-M4 32-bit	Xtensa Dual Core 32-Bit LX6	Broadcom BCM2711 Quad-Core	32-bit ARM Cortex
RAM	512 kB flash/64 kB RAM	320 KiB RAM, 448 KiB ROM	2GB / 4GB / 8GB LPDDR4-3200 SDRAM	256 kB flash memory
Wi-Fi	NA	Enabled	Enabled	No
Bluetooth	Yes	Yes	Yes	No
Price Range	High	Very Low	High	Low

Table 2.2: Comparison of nRF52, ESP32, Raspberry Pi & Arduino

2.1.4 IoT Platforms

The development of IoT applications are supported by different IoT platforms like operating systems, development frameworks, and programming languages for microcontrollers.

2.1.4.1 Operating Systems

Most IoT devices cannot run full-fledged operating systems like Linux and Windows. Hence, several operating systems have been developed for these resource constraint devices that enable them to connect to the sensors for data collection, communicate with the outside world using the internet and actuate based upon environmental changes.

RIOT OS [5] – RIOT OS is a real-time operating system that supports a wide range of IoT devices and is often called the Linux of the IoT world. It is a microkernel-based OS and takes into consideration energy efficiency, real-time capabilities, small memory footprint, modularity, and uniform API access, independent of the underlying hardware necessary for IoT applications. It supports application development in C and C++ programming languages. It provides core functionalities like multi-threading, priority-based scheduling, Inter-Process Communication, and interruption handling. It also has built-in support for network protocols like IPv6, UDP CoAP, RPL, etc.

Mongoose OS – Mongoose OS is an open-source IoT development framework that was envisioned to be a complete environment for prototyping, development and managing connected devices. It supports the development of IoT applications using C as well as Javascript which is a simple and widely used programming language. It has good cloud integration support and uses the mos tool for various tasks: building firmware, flashing firmware, connecting to wifi and managing devices.

FreeRTOS – FreeRTOS is an open-source OS developed by Amazon that has grown greatly in popularity. It is a microcontroller-based operating system that supports the development of applications with small memory requirements and is thus suitable for IoT devices. It has developed good community support with libraries for cloud connectivity, security, etc. The languages supported for development include C and C++

2.1.4.2 Development Frameworks

Arduino – Arduino is a company that develops hardware as well as software for the Internet of Things. It has an open-source Integrated Development Environment (IDE) that makes it easy to write, build and flash code to IoT boards and hence is popular for everyone getting started to build IoT applications. It majorly supports writing code in C and C++. While it is possible to use it on different devices it works best with only Arduino boards.

IP LIVES ON LAYER 3 NETWORK LAYER FOR END TO END TRANSFER

ESP-IDF – ESP-IDF is an IoT development framework developed by Espressif and works for developing IoT applications on the ESP series of system on a chip (SOCs). The SDK is self-sufficient for building various applications that exploit the board features like Wi-Fi, and Bluetooth among others. It supports languages like C and C++. RIOT OS uses this framework for building applications.

MicroPython – MicroPython is a lean and efficient implementation of the popular language Python for IoT devices. This makes it easy for people who are already familiar with Python to quickly build IoT applications. While it is a full compiler for the language, it again uses ESP-IDF for building the application for ESP devices. Micropython is optimised to run on constrained devices while still being compatible with Python for code reuse.

2.2 Internet Protocol (IP)

Data packets are the most fundamental unit for transferring data across the internet. They are created by breaking a large data file into small chunks as a manageable unit to share them across the network reliably. Sending the data packets from one computer to another will require a logical name for each packet and also a route that can be followed by the packet from the source to the destination.

Internal Protocol (IP) [6] is a set of rules developed to reliably address and route the data packets in the network so that they can reach their labelled destination. It is a network layer protocol and most commonly uses Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) at the transport layer depending on the requirement of reliability of the data packets. Each device that connects to the internet receives an IP address which is a unique identifier for the host. IP packets are created by adding the IP header containing information such as the source & destination address, header and packet length, time to live, transport protocol to use, etc. A sample IP packet is shown in Figure 2.1. The routers make use of this information for forwarding the packet to the destination host.

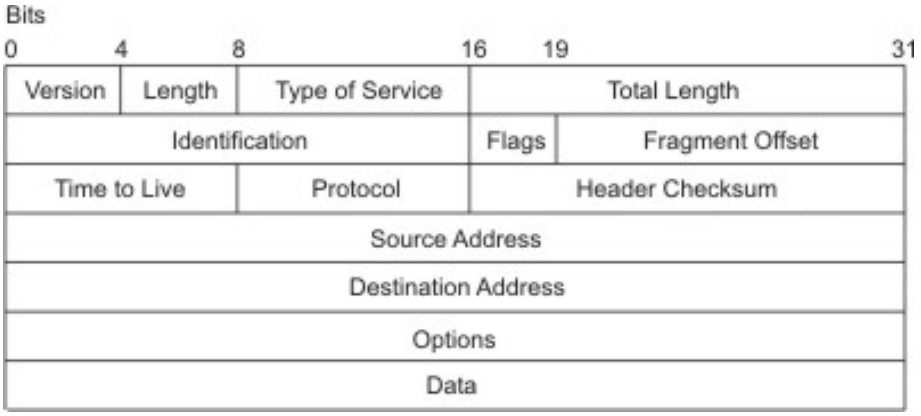

Figure 2.1: IP Packet Structure

The IP versions from 1 to 3 were experimental before the modern IPv4 was released and

TCP/UDP LIVE ON LAYER 4 TRANSPORT LAYER FOR SERVICE TO SERVICE TRANSFER

it has become the most widely used version today. The Internet protocol also faced a major challenge and realised that the available address space of the current IP is going to run out in the future. This issue was addressed in IPv6 by increasing the number of characters in the IP address and thus increasing the permutations available. IPv6 is still not fully adopted and most devices are still using the IPv4 addressing. Hence, it was necessary to operate a dual stack of IPv4 and IPv6 on the internet to support the incremental adoption of IPv6.

2.2.1 Challenges with Internet Protocol in the Internet of Things

Internet Protocol was envisioned decades ago and decided to follow a host-to-host communication model. On the other hand, the Internet of Things is a new technology that did not exist at that time and thus the challenges for an IoT network were not taken into consideration while designing the protocol. Internet of Things applications use the IP for communication since it has become the standard for today's internet and is an easy way for IoT devices to connect to the Internet. The challenges for IoT systems with the current Internet architecture [7] are analyzed below:

Connectivity of IoT devices – As discussed previously, the current internet architecture is based on a host-to-host paradigm and with the number of IoT devices connected growing to billions of devices it creates a bottleneck for connectivity. With this, the amount of data produced in the network is also increasing and handling this data efficiently will be difficult for the current communication model.

Mobility – The mobility of IoT devices brings another challenge of managing the data efficiently and making it available at the correct location when needed. The internet is not equipped to handle such a challenge and thus becomes a bottleneck for communication in such situations.

Resource constraints – The current internet model was not designed with energy efficiency in mind and thus causes a lot of overhead for battery-operated devices. It also causes a burden on other resources like storage, processing power, etc. A more efficient network which can optimize data flow in the network and reduce the use of valuable resources for IoT devices is necessary.

Security – The Internet protocol does not provide an in-built mechanism for end-to-end device security and thus it is up to the system to adopt a suitable security model. This makes the system complex and affects communication performance. This problem could be solved by a standard security mechanism at the network layer making it suitable for sensitive applications out of the box.

The ongoing research has focussed on new approaches to communication model that addresses the limitations of the existing design to better overcome these challenges. One

future architecture for solving these problems is Information-Centric Networking (ICN), which addresses the data instead of host-to-host communication.

2.3 Information-Centric Networking

The major task of a network is to design a way to deliver data from an existing source to the requestor of that data. For this, we need to decide on a naming scheme for routing the data. There are two main options [8] for this –

1. Naming the locations -

The Internet Protocol works on this design which names the hosts using an IP address and routes the data based on source and destination IP addresses. This is the point-to-point model that requires the requestor of the data to make a connection with the server before any sharing of data can take place.

2. Naming the data itself -

Information-Centric Networking (ICN) proposes that we name the data instead of the hosts. The requester of the data will directly address his request to the network with the name of the data. The network will route the data from the nearest producer of the data or the network cache if it is found along the path.

2.3.1 Information-Centric Networking Features

Information-Centric Networking provides some unique features as a result of its design choices which will be helpful in deploying fast and efficient IoT applications. The ICN features [3] are summarized below:

Content Naming – As discussed previously, ICN is based on naming the data instead of locations and thus content naming is a critical part. The content name should uniquely identify the data while also providing other features like scalability, compactness and readability. Different naming schemes like Hierarchical naming, flat naming, Attribute-Value based naming, and Hybrid naming have been proposed and they all have some benefits as well as disadvantages.

Routing and Forwarding – The naming of content brings about another change in the routing of the data packets. The requestor of the data only specifies the content name and the network is responsible for looking up the content and forwarding the request to the destination. This is achieved by the forwarding and routing tables. After the request reaches its destination, the content travels back by the same path to the requestor.

In-network Caching – The ICN networks can cache data packets at each node since they are location independent and self-sufficient. This has an advantage since partial content that

is popular can also be cached in the network and served when requested. Overall improves the performance of the network and reduces latency. There are a few challenges to determine the content that needs to be cached based on traffic patterns and the availability of the storage as well as the freshness of the content.

Mobility – The content is identified only using naming and hence it is independent of the location where it was produced or stored. This makes location mobility of the data easier without the requestor needing to know about the storage location. Moreover, the content can be cached in the network making it available at various locations closer to the requestor of the data.

Content-based Security – Information-Centric Networking introduces a content-based security model where each data packet is self-authenticated reducing the overhead for the application to adopt external security mechanisms. The producer of the data signs the content and it is verifiable by any consumer using a public key. It protects the network from common attacks such as sniffing, spoofing, replay and message modification. Researchers have identified new types of attacks like cache pollution and content poisoning to which the NDN network is vulnerable and looking for solutions to avoid them.

2.3.2 ICN Architectures

Information-Centric Networking is able to solve complex problems faced by the current Internet and hence the research community has focussed its attention to develop different architectures [10] based on the ICN paradigm. We will look at the most popular architectures below and choose the one that suits the need of this project.

NetInf – The NetInf architecture features a Centralized Name Resolution Service (NRS). It converts content names into locations For eg. IP addresses and thus is compatible with current infrastructure. When some content is published, the name for the same is registered in the NRS. The content naming scheme for this architecture is flat and hence it follows hierarchical structure for the infrastructure consisting of global and local NRS. This architecture is easy to implement with minimal changes to existing hardware. But, the major disadvantage would be that the scalability of the network would be limited.

NDN / CCN – Named Data Networking features two types of packets - Interest packet to request some content and a data packet containing the data as a response. The user only requests for data using the name of the content and the network is responsible for routing the interest packet in the direction of the requested data object. This maybe at one of the producer locations or a cache within the nodes of the network. The data packets travel back by the same path where the request originated and is cached in some or all nodes in this path. Named Data Networking is an enhanced version of the Content-Centric Network (CCN).

2.3.3 Suitability for IoT

We observe that ICN features satisfy many requirements of Internet of Things applications mentioned in section 2.1.1. Thus, Information-Centric Networking will make the development of these applications less complex while also improving communication performance. Table 2.2 compares the features provided by Information-Centric Networking for IoT and directly compares them to Internet Protocol [9].

IoT requirements	***IP-based efforts***	***NDN native features***
Resource naming	DNS, URI	Named content
Application data security	Object-based security	Self-secured packets
Request - response model	CoAP, REST	Consumer driven model
Caching	At application layer	In-network caching

Table 2.3: Comparison IP and ICN for IoT

The IP as layer is unable to fulfil the needs of IoT applications at the network layer and hence various issues like performance degradation and interoperability arise. This could be overcome by adopting Named Data Networking which will provide all the functionalities at network layer as highlighted in Table 2.3

ONLY LAYER 3 CHANGES

2.3.4 Comparision to Internet Protocol

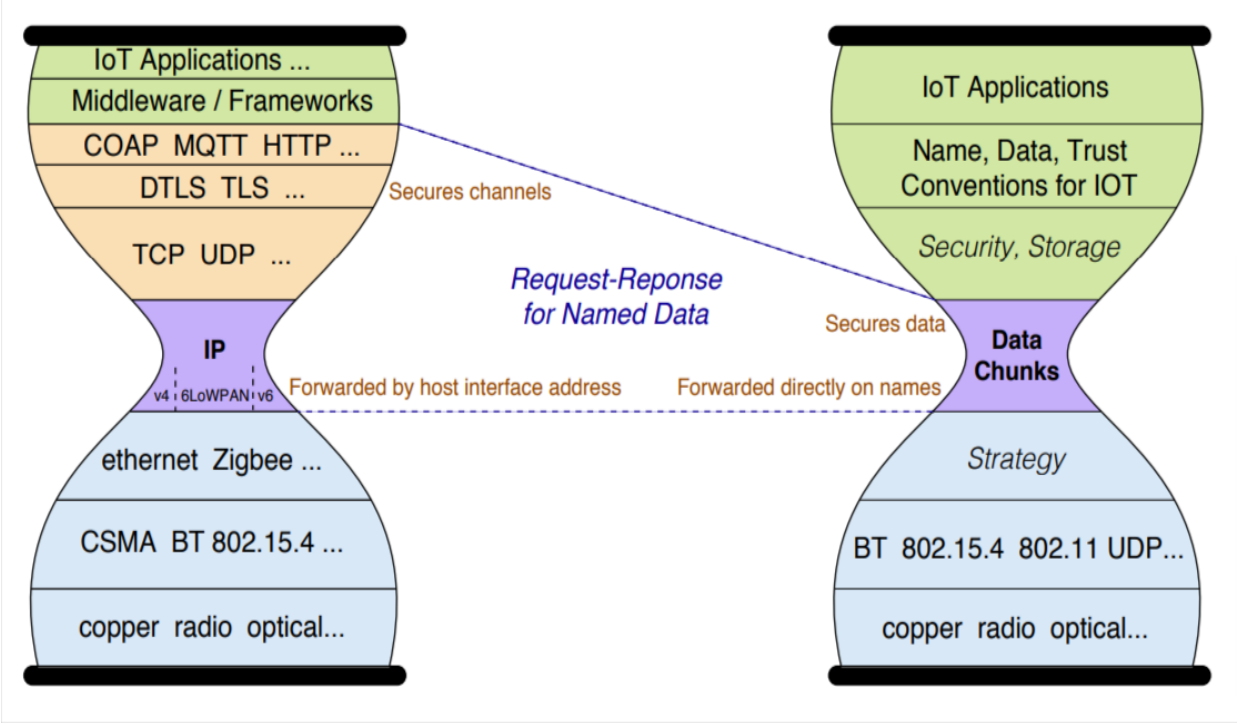

Figure 2.2: IP vs NDN protocol stack

The protocol stack [8] for Named Data Networking has a lot of similarities with the IP stack but also has some fundamental differences that are highlighted in Figure 2.2. We can see that NDN retains the same hourglass shape of the IP protocol stack and supports the use of existing transport layer protocols to perform datagram delivery. The structure of the protocol stack changes at the thin waist where NDN replaces IP packets having source and destination addresses for routing with named content chunks. Based on this fundamental change, we summarize how Named Data Networking compares to Internet Protocol in Table 2.4 below.

Internet Protocol (IP)	*Named Data Networking (NDN)*
Namespace consists of IP addresses for each node in the network	The content is named using a hierarchical structure independent of location
No inherent security mechanism is present, making it complex to implement at the application layer	It provides security at the network layer by signing each data packet at the time of production

Transfers data packets from source to destination IP address	The data packets are fetched based on interests and served from closest location
Data is directly served from the server unless a caching mechanism like CDN is setup	Inherently caches data in the network thus improving latency and increasing efficiency

Table 2.4: Comparison of features of IP and NDN

The differences in the networking paradigm bring about benefits in terms of network efficiency and latency. To demonstrate this, we can see how content retrieval happens in both of these architectures in Figure 2.3 where we look at a case when all consumers are requesting the same data. The data flow in IP is shown in red colour while ICN is shown using the blue colour. We see that the IP network has to obtain the address of the host and individually fetch the same data by making a connection to the host. This has a large overhead as compared to ICN which fetches the data by name until it is found in one of the nodes in the network.

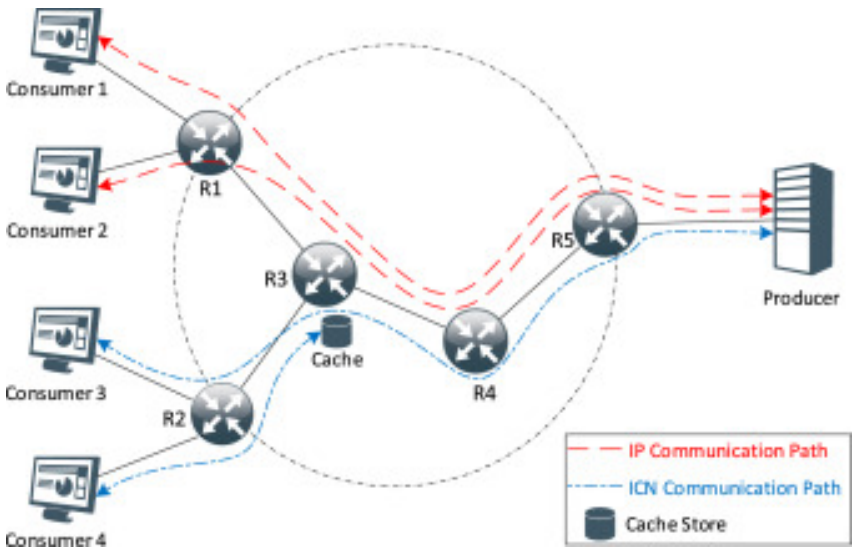

Figure 2.3: IP vs ICN content retrieval

2.3.5 How does NDN work?

Named Data Networking consists of two main components that are responsible for the routing and forwarding in the network.

NDN Forwarding Daemon (NFD) – NFD is the network forwarder component of the NDN platform. It is designed in such a way that it is modular and extensible so that experimenting with new protocol features, algorithms, etc is possible. Let us look at some NFD modules:

- Core - This contains common services like hash computation routines, DNS resolver, config file, face monitoring, etc.
- Faces - Abstracts network interfaces as Faces. Communication channel like TCP, UDP or any other means for socketing
- Tables - Content Store (CS), Pending Interest Table (PIT), Forwarding Information Base (FIB) & other data structures to support forwarding of the packets
- Forwarding - Includes basic forwarding and a framework to support different forwarding strategies
- Management - It is useful for configuration and state management

Named Data Link State Routing Protocol (NLSR) – NLSR [11] is the most popular routing algorithm for NDN. It populates the Forwarding Information Base (FIB) so that the router has the most up-to-date information on the changes in the network. When a request is received for a name prefix, it computes the route to the data. A unique feature of this algorithm is that it calculates multiple next hops for the same or different data producers mentioned in the interest request. The network maintains the states of adjacent nodes and broadcasts changes using Link State Advertising (LSA). The latest LSAs are stored in each node in the Link State Database (LSDB). NSLR uses the ChronoSync protocol which is a hash-based approach for synchronising changes in the database at each node. A unique feature of this algorithm is that it calculates multiple next hops for the same or different data producers mentioned in the interest request and updates the same in Forwarding Information Base. Thus, it reduces latency as well as increases the probability of a successful response from the network.

We have two roles in any network and we will define them as follows:

Data Producers – They are the ones who publish the content into the network and advertise them for other users to discover. They have to sign the content with their private key before publishing and advertise it with a name prefix.

Data Consumers – They request the network for the content using a name and expect the content from the network. If the request is successful, they will receive the data which can be verified by them using the public key of the producers.

While the Internet Protocol uses IP packets for routing, Named Data Networking consists of two types of packets based on which the data flow inside the network.

Interest Packet – The consumers of data send an interest packet with the name of the content requested which is routed by the network to the location where that data exists.

Data Packet – The data packet is sent back in response to an Interest packet received to the node through the network. It will contain the signed data requested by the consumer.

The complete procedure to be followed by the NDN network is detailed below:

- Interest packet is sent by the consumer to the network
- Router checks in the content store (CS) that contains the previously cached data and responds to the request right away if the data is found
- Router checks if Pending Interest Table (PIT) if a similar interest is already present and adds the new interest in the list to receive the same data as other pending requests
- Interest if forwarded to multiple next nodes using the information in the forwarding information base (FIB) until it reaches a node containing the requested data
- The data is sent back through the same path it reached the node and is cached along the way in the content store of the nodes based on operating policy

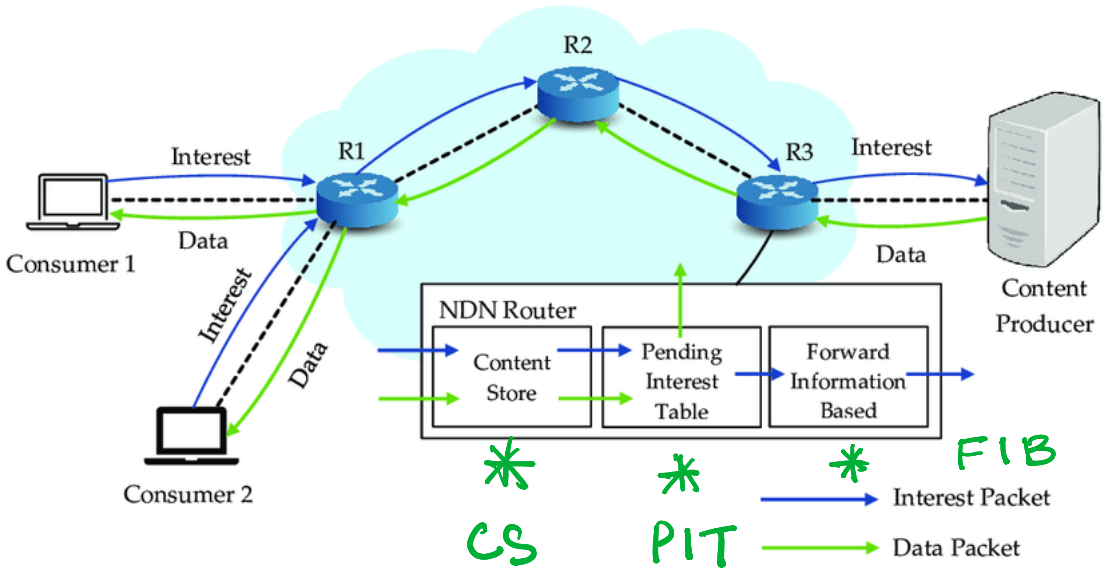

Figure 2.4: NDN network Data Flow

Computer Science is a PIT of FIBs.

2.3.6 NDN Deployment Alternatives

A real-world deployment of Named Data Networking is necessary for exploiting the benefits that it provides for Internet of Things applications. We have several options [9] for this:

NDN as a native network protocol – This approach will require major changes in internet infrastructure to implement completely. Also, considering that NDN is in the nascent stages of development, this option seems impossible. It could also be used in local networks which do not need any communication with the outside world. While some applications might dictate this need, it brings about limited benefits since scalability is a major part where the NDN architecture outshines IP. Thus, we can say it is unrealistic to implement such a network in the real world currently.

NDN as an overlay on top of IP – This approach is actually possible to implement and creates a uniform NDN layer in the network. In fact, this is the current approach taken in the development of the NDN testbed. But it has several problems, especially for IoT applications. Since the IoT devices are resource-constrained, deploying both IP and NDN is not a good idea even if some devices are capable of the same. Moreover, all the current IP applications must switch to NDN in order to use the network and considering that the internet is dominated by IP applications, this is an arduous task for migration.

NDN-Edge Architecture – For the migration of IP networks into NDN which might happen over years, we can look at an incremental deployment approach in which both the IP and NDN stacks coexist within parts of the network. One such architecture suggests the deployment of NDN at the edge of the network while still maintaining the IP networking at the core. This is a more realistic approach which does not need major changes to either the existing infrastructure or applications while still being able to test out the NDN concept in the real world. Thus, we will be adopting this approach for developing this project.

2.3.7 NDN Implementations

The development of Named Data Networking is in nascent stages and hence a lot of implementations have not yet been tested in real-world deployments. We will look at some of the choices available for developing IoT applications using NDN to decide on a feasible implementation for this project.

2.3.7.1 Emulators and Simulators

A simulator is designed to create an environment that contains all of the software variables and configurations that will exist in an app's actual production environment. In contrast, an emulator attempts to mimic all of the hardware features of a production

environment and software features. We have two options for NDN in this category.

Mini-NDN – Mini-NDN is a lightweight emulator tool for testing and experimenting with the NDN platform. It is based on the popular tool Mininet and shares a lot of similarities in terms of functionality and usage.

ndnSIM – ndnSIM is a simulator for NDN that features extensibility for experimenting with new developments quickly. It is an NS-3 module which implements the NDN communication model.

While they are good for performing quick experiments, they do not take into consideration difficulties faced while developing applications in a real-world environment.

2.3.7.2 Core Implementations

ndn-cxx – ndn-cxx is the core library implementing NDN and is used by other applications currently in research. They include NDN Forwarding Daemon, Named-data Link-State Routing protocol,ndn-tools, ChronoSync, etc. They form a good ecosystem for developing applications using NDN. This library is supported by specific versions of Ubuntu, macOS and centOS.

NDNts – NDNts is a library developed for programming NDN applications for the modern web using the latest technologies like Typescript. This library depends on NFD implementation and also requires Nodejs as a dependency. It has good documentation with examples and is easier to code for people already familiar with creating web applications. The platforms supported by the library include Windows, Mac, Android and iOS 15.

2.3.7.3 IoT Implementations

NDN-Lite – NDN-Lite library implements functionalities like forwarder, ndn packet encoder, service discovery, access control, etc and its lightweight design suits constrained devices. It also supports adaptations for different devices that support the C programming language. However, we need to depend on the community for work builds for different platforms and many of them are marked as broken currently. It has also reduced functionalities like forwarding hints, RIB management, etc. to make the implementation lightweight. Even though it looks promising for IoT application development, it is difficult to get started with this library because of the issues mentioned above.

NDN-RIOT – NDN-RIOT [17] is a library for developing NDN applications on low-powered IoT devices with RIOT OS. It supports the core NDN functionalities with minimum computing resources. It uses a micro-kernel architecture as shown in Figure 2.5. The network modules are implemented as kernel threads and packet passing happens via Inter-process communication. It should be noted that when sending NDN packets over Ethernet, the current implementation sets the destination MAC address to the broadcast address (FF:FF:FF:FF:FF:FF).Also, RIOT-NDN only supports the latest IPv6 networking and assigns a

inet6 address to all IoT devices for communication. The supported programming for application development is C. Unlike, other implementations NDN-RIOT is working on a wide variety of devices and is easy to get started with example applications.

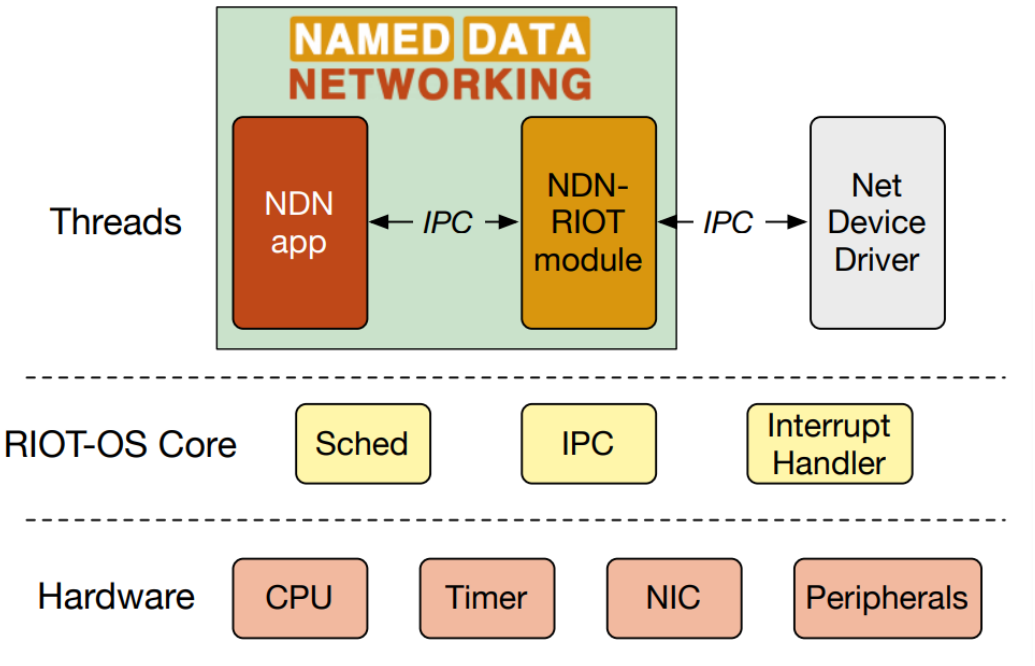

Figure 2.5: NDN RIOT Architecture

esp8266ndn – esp8266ndn is an Arduino library for NDN application development that supports multiple ESP series devices and nRF52. The library is still in development and provides fragmented functionalities on different platforms. For example, the support for IPv6 networking on ESP32 is still not available but works on ESP8266. It also has a limited number of examples like pingserver and pingclient. Moreover, it will depend on the Arduino development kit for running IoT applications on Expressif devices which is not an ideal case.

Platforms	**Riot OS**	**Arduino**	**POSIX, Riot OS**
IoT devices	ESP8266, ESP32, MIPS, RISC-V, ARM7, etc	ESP8266, ESP32, ESP32-S2, ESP32-C3, nRF52	ESP8266, nRF52, Raspberry Pi, Mac OS
Comms	Ethernet, Wifi, Bluetooth, LR-WPAN	Ethernet, UDP, BLE	Ethernet, UDP, BLE, LR-WPAN

Languages	C	C	C, C++
Drawbacks	Broadcasts messages in the network	Limited examples, No Ipv6 on Esp32	Not tested for all platforms, Difficult to get started

Table 2.5: Comparison NDN implementations for IoT devices

Chapter 3

Design

This chapter will focus on identifying the problem based on the state of the art review that the dissertation will focus to solve. It will also present a solution to tackle the problems identified. Lastly, it details the architecture of the proposed solution covering the high level components that form the part of the solution.

3.1 Problem Description

In the previous chapter, we described the background and work currently being carried out in the field of the Internet of Things. We also saw how Internet Protocol was facing challenges to adapt to the new Internet of Things paradigm. We looked at a new network design called Information-Centric Networking (ICN) that promises to solve these challenges and make the Internet of Things more efficient. We identified that the research community has focussed its efforts on Named Data Networking (NDN) architecture out of all other ICN choices. We also looked at the working, deployment alternatives and the current implementations of the Named Data Networking. The two major problems related to the same are summarized below:

1. Named Data Networking is currently still evolving and the various implementations are either broken or not very well tested in the real-world environment. This means that the implementations do not exist for a lot of popular IoT devices or they are missing features. Also, the working solutions have not been evaluated for scalability, mobility, interoperability and other IoT requirements.

2. The current internet is dominated by the Internet Protocol and migration to the Named Data Networking paradigm will require major changes to the infrastructure as well as existing applications based on it. This brings us to the idea of incremental adoption of Named Data Networking during which both IP and NDN coexist in the network. The new IoT applications can exploit the benefits of NDN while still being able to connect to the internet. The real-world designs will enrich NDN experiments and also make it an important focus for incoming IoT solutions.

The dissertation aims to solve the two problems identified through literature review and proposes a solution for them in the next section.

3.2 Proposal

The first problem will require the development of the project to be focused on real-world implementations of Named Data Networking, especially for low-powered IoT devices. We have three types of implementations of NDN currently available - simulators, core implementations and lightweight implementations for IoT devices. Based on the detailed evaluation of them in section 2.3.7, we will choose the following:

ndn-cxx – ndn-cxx library and the ecosystem of NFD, NSLR, ndn-tools, etc. for the development of an NDN network running on Linux desktop environment. This will require the creation of multiple nodes running NDN which will be achieved using Docker containers.

NDN-RIOT – NDN-RIOT library will be used to run the NDN implementation on ESP32 devices and set up a network of low-powered devices. It will be achieved by connecting the IoT devices to the same network using Wifi and the IPv6 module provided by NDN-RIOT.

The second problem requires us to connect the local NDN networks to the internet and thus coexist with the IP networking. We will attempt to connect our NDN network running on the Linux machine and the ESP32 devices for communication between them over the internet using UDP/IP. The NDN containers will be able to request data using an Interest packet from the sensors present in the ESP32 devices.

The detailed architecture for the same is described in the next section.

3.3 Architecture

The project architecture is shown below in Figure 3.1. The architecture is logically composed of three things:

NDN IoT networks – A number of IoT devices running the RIOT-NDN implementation and connected in a network. They will publish a name prefix for the data they are producing for example a temperature sensor publishing the current values. The other nodes in the network are able to send an interest packet for this published data and receive a data packet. We can have multiple such networks with their own namespace producing different data.

NDN network – A set of nodes that run Named Data Networking core components described in section 3.2. Each node is connected to another in the network using a Face which is a network interface abstraction in NDN. It holds a connection to a forwarder and supports interest / data exchange. This network is not connected to the external IP network except for the gateway node.

Gateway Node – The gateway node is the access point for the NDN network to the outside world. When any node in the NDN network needs to communicate with the sensors from the IoT networks, it will be routed through the gateway node. For this the gateway node maintains the list of all the external name prefixes available for communication. It also listens to messages sent by sensors from the external network and forwards them back through the same path in the NDN network.

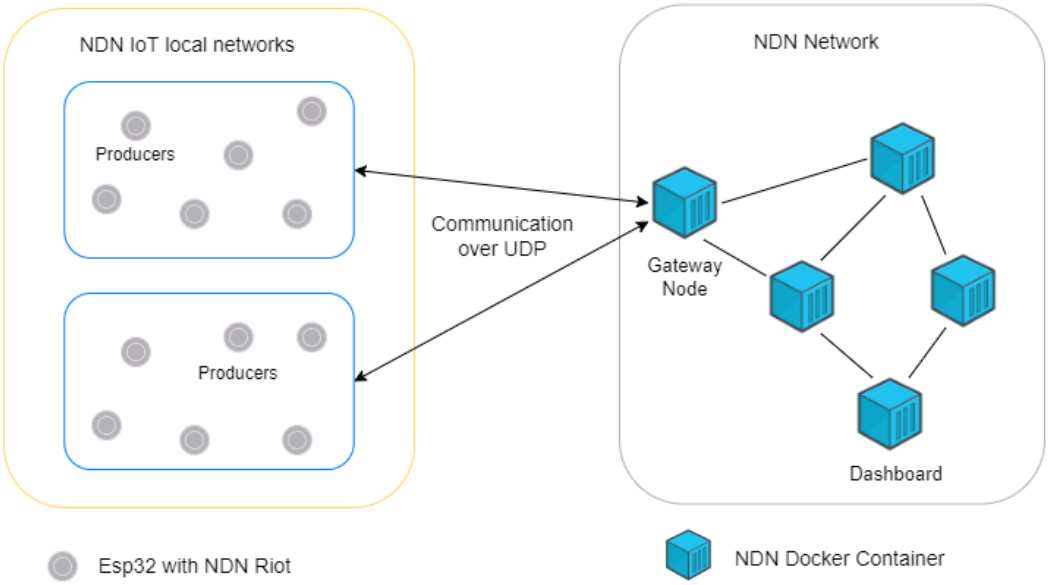

Figure 3.1: Project Architecture

This shows the high-level design of the system and sets the background for what it will be achieving. The details of all the components and their setup is described in the next chapter along with the complete explanation of the working of the system.

Chapter 4

Technical Implementation

This chapter will first describe all the low-level components of the system including how they were setup and how they work individually. The next part will focus on the complete workflow of the system to achieve the objective described in section 3.2. Lastly, it will evaluate the project and discuss the results for the same.

4.1 System Components

This section will show in brief the different components that make up the system and also the working of those components.

4.1.1 Internet of Things Devices

The main component in the system will be the IoT devices running the NDN implementation. We have chosen the ESP32 device for this because of it's low cost and resource-constrained nature suitable for the actual IoT environment. Each device can act as a data producer as well as data consumer. As discussed earlier, we have configured the devices to run the RIOT OS and the NDN-RIOT library for the NDN functionality. The programming for these devices was done using C language and the ESP-IDF build system helps us to run the code on the devices.

The RIOT OS provides us with the esp_wifi module for connecting the device to the internet and in the process we obtain an IP address as shown in Figure 4.1 below. We had earlier noted that the RIOT OS only supports communication through the latest IPv6 of the Internet protocol and hence we are able to see that highlighted IP address follows the IPv6 format.

```
2022-08-16 16:56:37,005 # Connect to serial port /dev/ttyUSB0
Welcome to pyterm!
Type '/exit' to exit.
ifconfig
2022-08-16 16:57:03,072 # ifconfig
2022-08-16 16:57:03,075 # Iface  9  HWaddr: 3C:61:05:4C:3C:38  Channel: 11  Link: up
2022-08-16 16:57:03,078 #          L2-PDU:1500  MTU:1488  HL:64  RTR
2022-08-16 16:57:03,081 #          RTR_ADV
2022-08-16 16:57:03,082 #          Source address length: 6
2022-08-16 16:57:03,087 #          Link type: wireless
2022-08-16 16:57:03,092 #          inet6 addr: fe80::3e61:5ff:fe4c:3c38  scope: link  VAL
2022-08-16 16:57:03,099 #          inet6 addr: 2001:bb6:1d26:1200:3e61:5ff:fe4c:3c38  scope: global  VAL
2022-08-16 16:57:03,101 #          inet6 group: ff02::2
2022-08-16 16:57:03,104 #          inet6 group: ff02::1
2022-08-16 16:57:03,106 #          inet6 group: ff02::1:ff4c:3c38
2022-08-16 16:57:03,110 #          inet6 group: ff02::1a
2022-08-16 16:57:03,111 #
2022-08-16 16:57:03,115 #          Statistics for Layer 2
2022-08-16 16:57:03,117 #            RX packets 691  bytes 54963
2022-08-16 16:57:03,121 #            TX packets 7 (Multicast: 5)  bytes 476
2022-08-16 16:57:03,126 #            TX succeeded 6 errors 0
2022-08-16 16:57:03,129 #          Statistics for IPv6
2022-08-16 16:57:03,147 #            RX packets 188  bytes 15524
2022-08-16 16:57:03,151 #            TX packets 7 (Multicast: 5)  bytes 456
2022-08-16 16:57:03,153 #            TX succeeded 7 errors 0
```

Figure 4.1: Esp Wifi module for IPv6 Address

We have multiple devices connected in a network that are able to publish data as well as send data and interest packets for communication. We can see the device acting as publisher of humidity values in Figure 4.2 and waiting for any consumer requesting the data. A node in the network is able to send interest packet for this data as shown in Figure 4.3. We can see in Figure 4.4 that the first node receives the interest packet and responds with a data packet.

```
Twisted not available, please install it if you want to use pyterm's JSON
2022-08-17 09:21:04,135 # Connect to serial port /dev/ttyUSB1
Welcome to pyterm!
Type '/exit' to exit.
ndn server /humidity 11
2022-08-17 09:21:17,570 # ndn server /humidity 11
2022-08-17 09:21:17,579 # server (pid=4): start
2022-08-17 09:21:17,592 # ndn: ADD_FACE message received from pid 4
2022-08-17 09:21:17,615 # server (pid=4): register prefix "/humidity"
2022-08-17 09:21:17,630 # ndn: ADD_FIB message received from pid 4
2022-08-17 09:21:17,636 # server (pid=4): enter app run loop
```

Figure 4.2: Sensor as Data Producer in NDN network

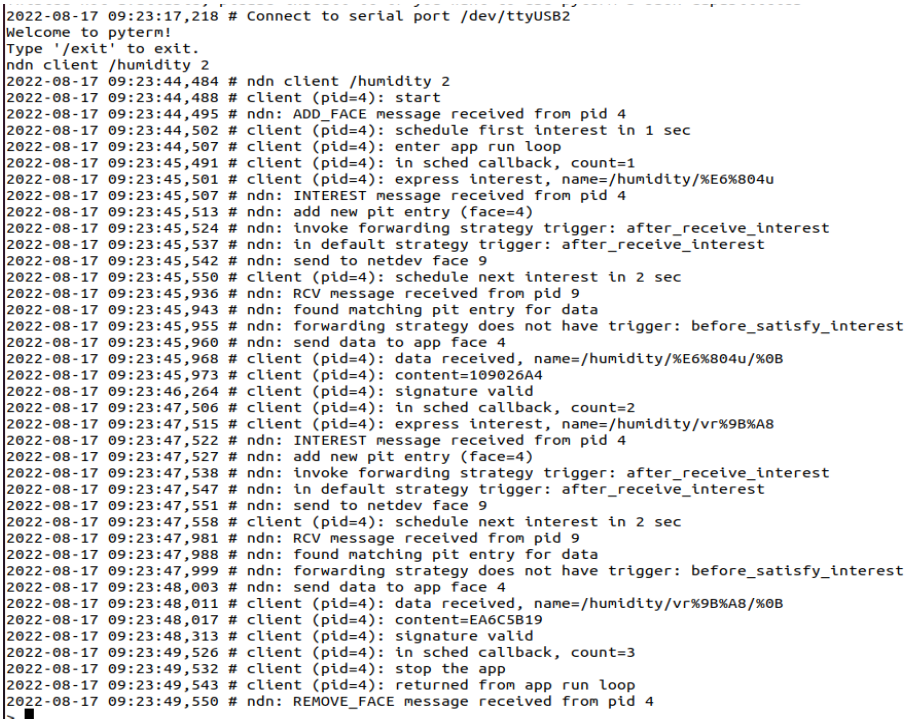

Figure 4.3: IoT device sending Interest packets

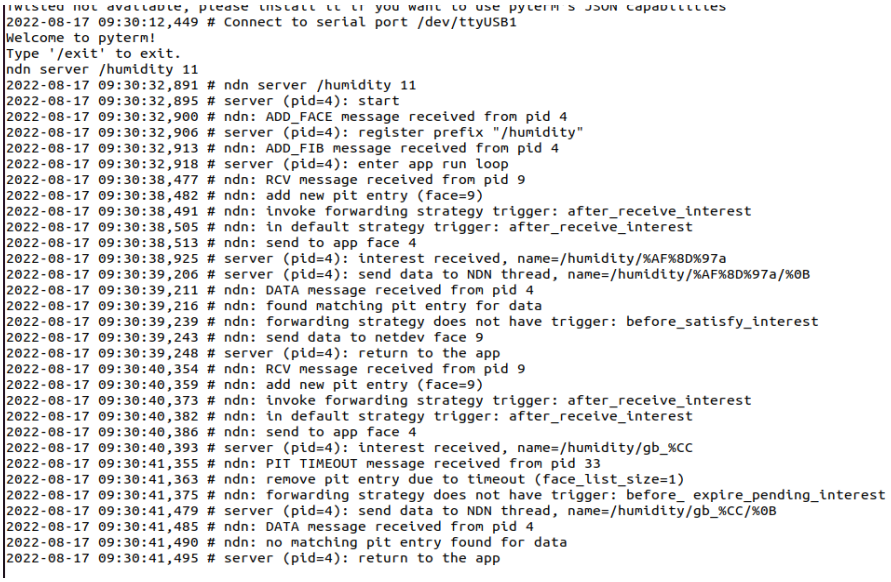

Figure 4.4: NDN server sending back Data packets

4.1.2 NDN Network

A network of nodes running the ndn-cxx implementation and all the other ecosystem of the Named Data Networking using docker containers. The docker containers must be configured for Ipv6 networking so that RIOT-NDN will be able to communicate with the network. Figure 4.5 shows the details of custom ndn network created using docker. The containers key shows

the three nodes which are part of this this network and their details. We observe that the config key shows both the Ipv4 and IPv6 subnet as well as the gateway address. This means that the containers are configured for IPv6 traffic.

Figure 4.5: IPv6 enabled NDN network with three containers

We see in Figure 4.6 that node3 will be publishing data from a file using the ndnputchunks command from the ndn-tools modules. node1 which exists at some hops distance away in the network will be able to send an interest packet for this data and it receives a response with a data packet for the requested name prefix.

```
pathakpratik@ubuntu:~$ sudo docker exec -d node3 bash -c "ndnputchunks -Nt ndn:/ndn-docker/speed/v=3 < /home/speed.txt"
[sudo] password for pathakpratik:
pathakpratik@ubuntu:~$ sudo docker exec node3 ndncatchunks ndn:/ndn-docker/speed/v=3
ERROR: Fetching terminated but no final segment number has been found
pathakpratik@ubuntu:~$ sudo docker exec node3 ndncatchunks ndn:/ndn-docker/speed
90 km/h


All segments have been received.
Time elapsed: 0.00158041 seconds
Segments received: 1
Transferred size: 0.009 kB
Goodput: 45.557827 kbit/s
Congestion marks: 0 (caused 0 window decreases)
Timeouts: 0 (caused 0 window decreases)
Retransmitted segments: 0 (0%), skipped: 0
RTT min/avg/max = 1.172/1.172/1.172 ms
pathakpratik@ubuntu:~$ sudo docker exec node1 ndncatchunks ndn:/ndn-docker/speed
90 km/h


All segments have been received.
Time elapsed: 0.00111664 seconds
Segments received: 1
Transferred size: 0.009 kB
Goodput: 64.479094 kbit/s
Congestion marks: 0 (caused 0 window decreases)
Timeouts: 0 (caused 0 window decreases)
Retransmitted segments: 0 (0%), skipped: 0
RTT min/avg/max = 0.467/0.467/0.467 ms
pathakpratik@ubuntu:~$
```

Figure 4.6: Publish & Fetch Data in the NDN network

4.1.3 Gateway Node

The Gateway node is one of the node from the NDN network assigned to communicate with the external node and runs a python script which can send & receive messages to external network i.e IoT networks in our case. The gateway is listening on a custom PORT 8080 as seen the Figure 4.7 below.

```
root@80549bb06f44:/home# python3 gw.py
Listening on UDP port:  8080
```

Figure 4.7: The Gateway Node listening for incoming messages

4.2 Project Steps

The communication between the NDN network and the IoT sensors which are running in separate NDN deployments takes place in the following six steps:

1. Register Prefix for Producers from the IoT network at the external Gateway Node
2. Send Interest Packet from Dashboard can be any node in the network in our case for the advertised Prefix

3. The Gateway node translates the NDN message into a UDP packet and sends it to the external IoT network

4. The IoT node listens to the UDP message and propagates it as an NDN message locally in the network

5. The producer listens for the NDN message and sends the data to the gateway by converting it to a UDP packet

6. The gateway listens to the UDP packet and translates it back to the NDN message and forwards it along the same path as the interest packet

4.3 Project Execution

The IoT nodes will be running a UDP server for any communication with the external network through the gateway node. An example of this is to notify the gateway about a new name prefix registered by any of the IoT sensors in the network which is demonstrated in Figure 4.8

Figure 4.8: IoT device notifies Gateway about new NDN name prefix

The IoT network is up and running for handling any requests from external network. The next step is to send an Interest packet from the External NDN network for the name prefix registered at the gateway. This is shown below in Figure 4.9. On receiving the Interest packet in UDP format, the IoT node converts it into an NDN message and propagates it locally to the producer that has the registered prefix.

```
2022-08-17 16:35:24,037 # ---------- Communicating message locally ------------client (pid=12): start
2022-08-17 16:35:24,044 # ndn: ADD_FACE message received from pid 12
2022-08-17 16:35:24,050 # client (pid=12): schedule first interest in 1 sec
2022-08-17 16:35:24,056 # client (pid=12): enter app run loop
2022-08-17 16:35:24,064 # Success: send 30 byte to 2001:bb6:1d26:1200:3e61:5ff:fe4c:3c38
> 2022-08-17 16:35:25,031 # client (pid=12): in sched callback, count=1
2022-08-17 16:35:25,042 # client (pid=12): express interest, name=/network/17/sensor/temperature/%B3%B0%24e
2022-08-17 16:35:25,047 # ndn: INTEREST message received from pid 12
2022-08-17 16:35:25,052 # ndn: add new pit entry (face=12)
2022-08-17 16:35:25,064 # ndn: invoke forwarding strategy trigger: after_receive_interest
2022-08-17 16:35:25,073 # ndn: in default strategy trigger: after_receive_interest
2022-08-17 16:35:25,077 # ndn: send to netdev face 9
2022-08-17 16:35:25,085 # client (pid=12): schedule next interest in 2 sec
2022-08-17 16:35:25,558 # ndn: RCV message received from pid 9
2022-08-17 16:35:25,563 # ndn: found matching pit entry for data
2022-08-17 16:35:25,575 # ndn: forwarding strategy does not have trigger: before_satisfy_interest
2022-08-17 16:35:25,580 # ndn: send data to app face 12
2022-08-17 16:35:25,591 # client (pid=12): data received, name=/network/17/sensor/temperature/%B3%B0%24e/%0E
2022-08-17 16:35:25,595 # client (pid=12): content=D6C5B3E6
2022-08-17 16:35:25,887 # client (pid=12): signature valid
2022-08-17 16:35:27,052 # client (pid=12): in sched callback, count=2
2022-08-17 16:35:27,058 # client (pid=12): stop the app
2022-08-17 16:35:27,064 # client (pid=12): returned from app run loop
2022-08-17 16:35:27,070 # ndn: REMOVE_FACE message received from pid 12
2022-08-17 16:35:27,075 # Recvd: /network/17/sensor/temperature
```

Figure 4.9: Propagating Gateway message in local NDN network

This is done by just unpacking the message containing the name prefix and sending an interest packet with that prefix to the IoT network running NDN. The NDN network will route this packet to the producer and it will then send the data packet packaging it as a UDP packet to the gateway node as shown in Figure 4.10

```
ndn server /network/17/sensor/temperature 14
2022-08-17 16:34:58,234 # ndn server /network/17/sensor/temperature 14
2022-08-17 16:34:58,240 # server (pid=4): start
2022-08-17 16:34:58,247 # ndn: ADD_FACE message received from pid 4
2022-08-17 16:34:58,256 # server (pid=4): register prefix "/network/17/sensor/temperature"
2022-08-17 16:34:58,264 # ndn: ADD_FIB message received from pid 4
2022-08-17 16:34:58,281 # ------------ Notify Gateway for new NDN name prefix!!!! ----------server (pid=4): enter app run loop
2022-08-17 16:35:25,157 # ndn: RCV message received from pid 9
2022-08-17 16:35:25,165 # ndn: add new pit entry (face=9)
2022-08-17 16:35:25,174 # ndn: invoke forwarding strategy trigger: after_receive_interest
2022-08-17 16:35:25,185 # ndn: in default strategy trigger: after_receive_interest
2022-08-17 16:35:25,188 # ndn: send to app face 4
2022-08-17 16:35:25,211 # server (pid=4): interest received, name=/network/17/sensor/temperature/%B3%B0%24e
2022-08-17 16:35:25,470 # ------------ Communicating with gateway by UDP!!!! ----------Success: send 3 byte to 2001:bb6:1d26:1200:8445:8c47:70cd:bce1
2022-08-17 16:35:25,480 # server (pid=4): send data to NDN thread, name=/network/17/sensor/temperature/%B3%B0%24e/%0E
2022-08-17 16:35:25,485 # ndn: DATA message received from pid 4
2022-08-17 16:35:25,492 # ndn: found matching pit entry for data
2022-08-17 16:35:25,504 # ndn: forwarding strategy does not have trigger: before_satisfy_interest
2022-08-17 16:35:25,509 # ndn: send data to netdev face 9
2022-08-17 16:35:25,513 # server (pid=4): return to the app
```

```
oot@80549bb06f44:/home# python3 gw.py
istening on UDP port:  8080

'Register Prefix: /network/17/sensor/temperature'
'22C'
```

Figure 4.10: Package the NDN packet as UDP and forward to Gateway Node

Once the gateway receives the packet in UDP format and sends it back to the node from which the request originated. To achieve this the gateway has to store a list of all pending interest packets similar to how NDN stores it in the Pending Interest Table (PIT). It has to remove the entry from the table once it packages the UDP message received for the interest into an NDN data packet and sends it back to the node where the interest originated. Failure scenarios like request timeout, name prefix expired, external server down, etc. also need to

be handled when updating Pending Interest Table (PIT). This step of implementation would require more technical development currently not achieved as part of this project.

4.4 Results & Evaluation

The dissertation focussed on real-world implementations of Named Data Networking for IoT applications and proposed a solution for incremental deployment of NDN using a Gateway Node. A direct comparison of performance benefits between NDN and IP could be useful but the fairness of the results is questioning given the different configurations as highlighted in [9]. Thus, we rely on the feature suitability of NDN for IoT networks compared to the limitations that IP imposes. In this section, we will first look at various applications using NDN at the network layer in IoT applications. Next, we will look at some solutions proposed for the inter-networking of NDN and IP for comparison with the current project.

4.4.1 NDN IoT applications

The development of low-end IoT is still in initial stages even with IP and hence a full-fledged working application exploiting NDN is difficult to find for evaluation. If we look at the official projects list of Named Data Net at https://redmine.named-data.net/projects, we see that apart from core implementations like NFD, NLSR, ChronoSync they have NDN for browser and WebRTC. For embedded devices, it only mentions Raspberry Pi as a platform which is not a resource constrained device as discussed in section 2.1.3. We list some of the solutions found on Github or described as part of research in reputed journals like IEEE, ACM, etc.

Wearable device with temperature sensor [12] – This research mentions an application that sends the temperature from the wearable device to a consumer device using NDN. The links for deployment mentioned are currently not accessible. The IoT devices used are again Raspberry Pi which is expensive and consumes a lot of resources considering the benefits it provides for the application.

NDNOverUDP [13] – This project is a library which implements a lightweight NDN implementation for the Arduino platform. It has an example code for sending Interest and Data packets for getting home temperature over Ethernet. But it only works over a local network of a few devices and does not support connecting to the Internet apart from a lot of missing features listed on the repository. This project is not active as the last update was a long time ago.

Farming Application [9] – This research paper mentions a farming application that implements the ndn-cpp Lite library for the Arduino platform. The IoT NDN network communicates over IEEE 802.15.4 standard and the communication to external networks is through a gateway device over UDP/IP. This project has a similar idea to this dissertation but differs in the approach taken to implement it and is discussed in detail in the next section.

NDN-IoT-Android [14] – This project implements an application for NDN communication, secure sign-on and trust policy switching between the Android phone and IoT devices. The IoT devices use the ndn-lite package for NDN communication with the Android Phone over Bluetooth. The application can run on low-powered devices that have adaptions for the ndn-lite package. It does not implement any communication to external networks.

The summary of research shows that there are very few and basic NDN applications running on low-end IoT devices. This is because both the technologies are new and still in their initial stage of development. Our solution is one of the few NDN applications and is running NDN-RIOT on ESP32 devices as well as communicates with external NDN networks.

4.4.2 NDN IP Inter-networking Solutions

We will look at some proposed solutions for allowing NDN and IP coexist and deploy NDN networks without major changes in the existing infrastructure. All these solutions found for inter-networking of NDN and IP work on a similar idea of deploying a gateway node in some part of the architecture but there are some fundamental changes which are discussed in detail below:

IP-NDN Gateway [15] – This project presents a solution to help the existing IP applications to communicate with the NDN network without any changes. The gateway will handle the conversion of IP packets to NDN packets and vice-versa. It consists of a gateway daemon which spawns and orchestrates the traffic handlers based on a configuration file. The traffic handlers actually receive packets based on the filter set while running them and convert the packets before forwarding to the NDN network or IP applications. This enables the existing applications to exploit the benefits of NDN without major any changes and thus makes the adoption of NDN networks simple. For communication between two NDN networks as is the case in this dissertation, we will need to place one gateway node for communication with IoT network and another for NDN network. The solution is currently not tested for IoT devices but it looks feasible to implement even though not ideal for the use case.

NDN/HTTP Gateway [16] – This paper presents a solution along similar lines as the first one with a gateway for translating packets between NDN network and rest of the internet. But it consists of two types of gateway - ingress gateway and egress gateway separating responsibilities of incoming requests to the NDN network and outgoing requests from the NDN network. But handling any request requires three Interest/Data packet exchanges and this looks like a big overhead for handling a single request. Once again, this solution is not implemented for IoT devices but it looks feasible. The implementation makes use of Network Functions Virtualization (NFV) which helps in deploying NDN as software functions inside the

existing infrastructure. This will greatly help reduce the cost of adoption before investing in the new technology and hence this idea could be included in future real-world deployment of NDN technology.

IoT NDN Gateway [9] – This paper proposes the idea of a gateway node in each NDN network consisting of IoT devices for communication with the outside world. Each network serves requests based on a common prefix (CP). The IoT devices used for the prototype are Arduino and a superior device Raspberry Pi is used as the gateway node as it needs to support the translation of packets, device identity and access control, authentication server, etc. The applications can access this network through the gateway node by communicating over UDP/IP and local communication happens using IEEE 802.15.4 standard. This solution also includes a stateless packet compression algorithm to reduce overhead in the IoT network again included in the gateway node. The disadvantage of the approach is the need of a high-powered device as gateway for each IoT network which may not always be possible. Also, it suffers from single point of failure if the gateway node crashes.

The above projects present various ideas that could be implemented in the prototype developed in this dissertation. The prototype also holds certain advantages over other approaches and they are listed in section 5.1. Overall, these solutions show the need for Named Data Networking to be deployed in real-world scenarios while coexisting with IP at present and also about implementing it in IoT environments thus validating the direction of research presented in this dissertation. But, all the NDN applications as well as the deployment solutions are in the initial stage of development and thus require more effort for successful integration in the current internet.

Chapter 5

Conclusion and Future Work

This chapter concludes the work presented in this dissertation and is spread across two sections. The first section lists the main contributions of the dissertation and the second section mentions the possible future enhancements for the prototype developed as part of this dissertation.

5.1 Conclusion

The aim of this project as identified in section 3.1 was to develop an IoT application for resource-constrained devices exploiting the benefits provided by Named Data Networking paradigm and to develop a method to practically deploy it coexisting with the current infrastructure dominated by Internet Protocol. The prototype developed as part of this dissertation works on ESP32 device with NDN-RIOT implementation and a gateway node using Python for the NDN network running on Linux machine using containerized deployment of ndn-cxx library. The prototype developed is feasible for deployment in the current Internet Infrastructure and also takes advantages of all the features like name-centric design, in-network caching and security of named content. The main contributions of the project are highlighted below:

- Survey of existing NDN implementations available for both resource-constrained devices as well as powerful machines. Identifying the fragmentation issue in IoT devices where many libraries are broken or are missing features for some IoT devices
- Using the Named Data Networking paradigm, composed a network of ESP32 devices and developed an application where the device can act as a data producer like sensor as well as a consumer for actuation
- The NDN implementations for Linux are difficult to install with many issues and hence developed a docker container which includes the ecosystem for development of NDN applications supporting communication over the latest IPv6 protocol
- Identified the issue of lack of communication means outside the local NDN IoT network and proposed a solution for connecting the same with network of NDN containers
- Developed a gateway program which helps in communication between the NDN containers and IoT network using UDP/IP and thus designing a way for deploying the NDN network in real-world incrementally with existing internet.

5.2 Future Work

Named Data Networking is a Future Internet Architecture which is in its initial stage of development to the point that it has not been deployed into real-world scenarios. The testbed for NDN is a shared resource for research consisting of 34 nodes with 88 links at participating institutions across locations. This means that there is a lot of future scope for any NDN project developed before it can be used in production deployments. The following list covers some of the important considerations for improving this research further.

Resilient Gateway Node Architecture – In the current scenario, the gateway is a single point of failure as the system will stop functioning if it stops working. This can be overcome by using distributed systems concepts of leader election and heartbeat. Various algorithms like 3-phase commit [18] and Paxos [19] have been devised for a system to handle distributed processing. There are also available tools like Apache Zookeeper [20] and Kubernetes that can handle this task and thus they could be integrated for solving this problem.

Gateway Node for IoT network – All the IoT devices mandatorily run the UDP server apart from the RIOT-NDN as they expect to receive an interest packet or to send back the data packet for the sensor values it produces. While the resource-constrained devices are capable of doing this, the overhead could be reduced by assigning a gateway node for all the communication with external networks as done in the ndn-cxx implementation. This is currently very difficult to achieve as leader election and consensus require a lot of computation but more research is needed to determine this.

Authentication – The current focus of the research was to implement a project on low-powered IoT devices and devise a solution for incremental deployment. Hence, the authentication module which is a common requirement for all real-world projects was not developed. The gateway node needs to be authenticated by each IoT network before communicating any data packet to it. The same can be done by the gateway node for registering valid name prefixes from authenticated IoT networks. The authentication module could be integrated into the project keeping the new architecture in mind and could become a direction for research.

Bibliography

[1] IoT Business News (2022, May 19). State of IoT 2022: https://iotbusinessnews.com/2022/05/19/70343-state-of-iot-2022-number-of-connected-iot-devices-growing-18-to-14-4-billion-globally/

[2] IHS Markit (2017, October 24). Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says: https://news.ihsmarkit.com/prviewer/release_only/slug/number-connected-iot-devices-will-surge-125-billion-2030-ihs-markit-says

[3] Boubakr Nour, Kashif Sharif, Fan Li, Sujit Biswas, Hassine Moungla, Mohsen Guizani, Yu Wang, A survey of Internet of Things communication using ICN: A use case perspective, Computer Communications, Volumes 142–143, 2019, Pages 95-123, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2019.05.010.

[4] S. Al-Sarawi, M. Anbar, K. Alieyan and M. Alzubaidi, "Internet of Things (IoT) communication protocols: Review," 2017 8th International Conference on Information Technology (ICIT), 2017, pp. 685-690, doi: 10.1109/ICITECH.2017.8079928.

[5] E. Baccelli, O. Hahm, M. Günes, M. Wählisch and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2013, pp. 79-80, doi: 10.1109/INFCOMW.2013.6970748.

[6] CloudFlare (2022, May 19). What is the Internet Protocol?: https://www.cloudflare.com/learning/network-layer/internet-protocol/

[7] Hail, Mohamed. (2019). IoT-NDN: An IoT Architecture via Named Data Netwoking (NDN). 74-80. 10.1109/ICIAICT.2019.8784859.

[8] Afanasyev, Alexander, Jeff Burke, Tamer Refaei, Lan Wang, Beichuan Zhang and Lixia Zhang. “A Brief Introduction to Named Data Networking.” MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM) (2018): 1-6.

[9] Amar Abane, Mehammed Daoui, Samia Bouzefrane, Soumya Banerjee, Paul Mühlethaler. A Realistic Deployment of Named Data Networking in the Internet of Things. Journal of Cyber Security and Mobility, River Publishers, 2020, 9 (1), ff10.13052/jcsm2245-1439.911ff. ffhal-02920555ff.

[10] Fukuda, Ken-ichi. “Next-generation Network Architecture Led by Information-Centric Networking.” (2016).

[11] L. Wang, V. Lehman, A. K. M. Mahmudul Hoque, B. Zhang, Y. Yu and L. Zhang, "A Secure Link State Routing Protocol for NDN," in IEEE Access, vol. 6, pp. 10470-10482, 2018, doi: 10.1109/ACCESS.2017.2789330.

[12] S. K. Datta and C. Bonnet, "Interworking of NDN with IoT architecture elements: Challenges and solutions," 2016 IEEE 5th Global Conference on Consumer Electronics, 2016, pp. 1-2, doi: 10.1109/GCCE.2016.7800509.

[13] Antonio Cardace (2016) NDNoverUDP [Source Code]. https://github.com/acardace/NDNOverUDP.

[14] gujianxiao (2019) NDN-IoT-Android [Source Code]. https://github.com/gujianxiao/NDN-IoT-Android.

[15] Tamer Refaei, Jamie Ma, Sean Ha, and Sarah Liu. 2017. Integrating IP and NDN through an extensible IP-NDN gateway. In Proceedings of the 4th ACM Conference on Information-Centric Networking (ICN '17). Association for Computing Machinery, New York, NY, USA, 224–225. https://doi.org/10.1145/3125719.3132112.

[16] Xavier MARCHAL, Moustapha El Aoun, Bertrand Mathieu, Wissam Mallouli, Thibault Cholez, Guillaume Doyen, Patrick Truong, Alain Ploix, and Edgardo Montes De Oca. 2016. A virtualized and monitored NDN infrastructure featuring a NDN/HTTP gateway. In Proceedings of the 3rd ACM Conference on Information-Centric Networking (ACM-ICN '16). Association for Computing Machinery, New York, NY, USA, 225–226. https://doi.org/10.1145/2984356.2985238

[17] Shang, Wenli & Afanasyev, Alex & Zhang, Lixia. (2016). The Design and Implementation of the NDN Protocol Stack for RIOT-OS. 1-6. 10.1109/GLOCOMW.2016.7849061.

[18] D. Skeen and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," in IEEE Transactions on Software Engineering, vol. SE-9, no. 3, pp. 219-228, May 1983, doi: 10.1109/TSE.1983.236608.

[19] Lamport, Leslie. (2001). Paxos Made Simple. Sigact News - SIGACT. 32.

[20] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. 2010. ZooKeeper: wait-free coordination for internet-scale systems. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC'10). USENIX Association, USA, 11.

Appendix

Abbreviation	Expansion
IoT	Internet of Things
IP	Internet Protocol
ICN	Information-Centric Networking
NDN	Named Data Networking
CS	Content Store
FIB	Forwarding Information Base
PIT	Pending Interest Table
LAN	Local Access Networks
BLE	Bluetooth Low Energy
IDE	Integrated Development Environment
SOC	System on a chip
CCN	Content Centric Networking
NFD	NDN Forwarding Daemon
NLSR	Named Data Linked State Routing Protocol
HTTP	Hypertext Transfer Protocol
LSA	Link State Advertising
LSDB	Link State Database
NFV	Network Function Virtualization
CP	Common Prefix

Table 1: List of Abbreviations