Trinity College Dublin

The University of Dublin

CS7NS1: Scalable Computing

Project 3 Report

Group 22

Version 1.0

Table of Contents

About Group 22

Gayathri Girish Nair

23340334
MSc Data Science
girishng@tcd.ie

Contribution:

Idea seed. Scenario Set Up, ICN, NDN, Research, Python Programming, Writing Report, Tech Architecture, Workflow, PPT.

Tarun Singh

23330140
Msc Data Science
singht1@tcd.ie

Contribution:

Tech Architecture, PPT, Workflow, Security Components, Research, Python Programming, Writing Report, Tech Architecture, Workflow, PPT, Debugging

Tejas Bhatnagar

23334930
MSc Data Science
bhatnagt@tcd.ie

Contribution:

Research, Security, Code review and debugging, Python Programming , PPT, Testing, Writing Report, Workflow,

Demo Video: Scalable-Demo-project-3

Presentation link: Scalable-presentation-project-3

Miro Board *(Architecture & brainstorming):* Scalable-Architecture-project-3

Code Base (Github): Scalable-Github-project-3

Chapter 1: Introduction

1.1 Problem Statement

This project responds to evolving networking paradigms by developing a secure and scalable Information Centric Networking (ICN) protocol, utilizing Named Data Networking principles. It involves creating ICN-based networks, each with five device instances featuring eight interactable local sensors/actuators.

Emphasizing ICN's control and data messaging separation, the practical implementation on Raspberry Pi ensures robustness to node/network failures. Group participation is crucial for success, with constraints on infrastructure reliance and explicit off-Pi implementation articulation, emphasizing scalability, security, and design robustness.

1.2 Abstract

The use case emulated is that of communication between a team of 5 nanobots deployed in the body to detect and attack cancer cells only. A smart wristband worn by the patient (referred to in the code as a rendezvous server) helps the bots find each other. Each of the 5 bots sense a separate biological marker that is commonly associated with a cancer cell. If all 5 bots detect 1 (positive test result for respective marker), then the cell is collectively diagnosed as being cancerous, in which case, the bots attack it.

If even one bot reported a negative test result (0) then, the team diagnoses the cell to be healthy and reset their state (i.e. continue searching). If bots remain in the body too long without having detected cancer at all, then they diffuse themselves allowing the body to safely dispose of them naturally. One of the 5 markers (as defined in the config.json file) is a primary marker which means that it is detection of this marker that causes the team to suspect cancer and triggers group diagnosis.

The bot that detects this marker is the primary bot and is the first one to attach (referred to in code as tether) itself to a suspected cancer site. The primary bot then activates a beacon that bots which detect other markers (non-primary bots) pick up on when they arrive at the same spot via the bloodstream. When a non-primary bot picks up a beacon signal, they too tether themselves to the same spot as the primary bot and diagnosis begins. We've tried our best to cater to the above problem statement through our solution.

Chapter 2: Technical Architecture

2.1 Tech Stack

Programming Language: Python
Python Packages: os, time, socket, json, random, logging, threading, multiprocessing
IoT Device: Raspberry Pi

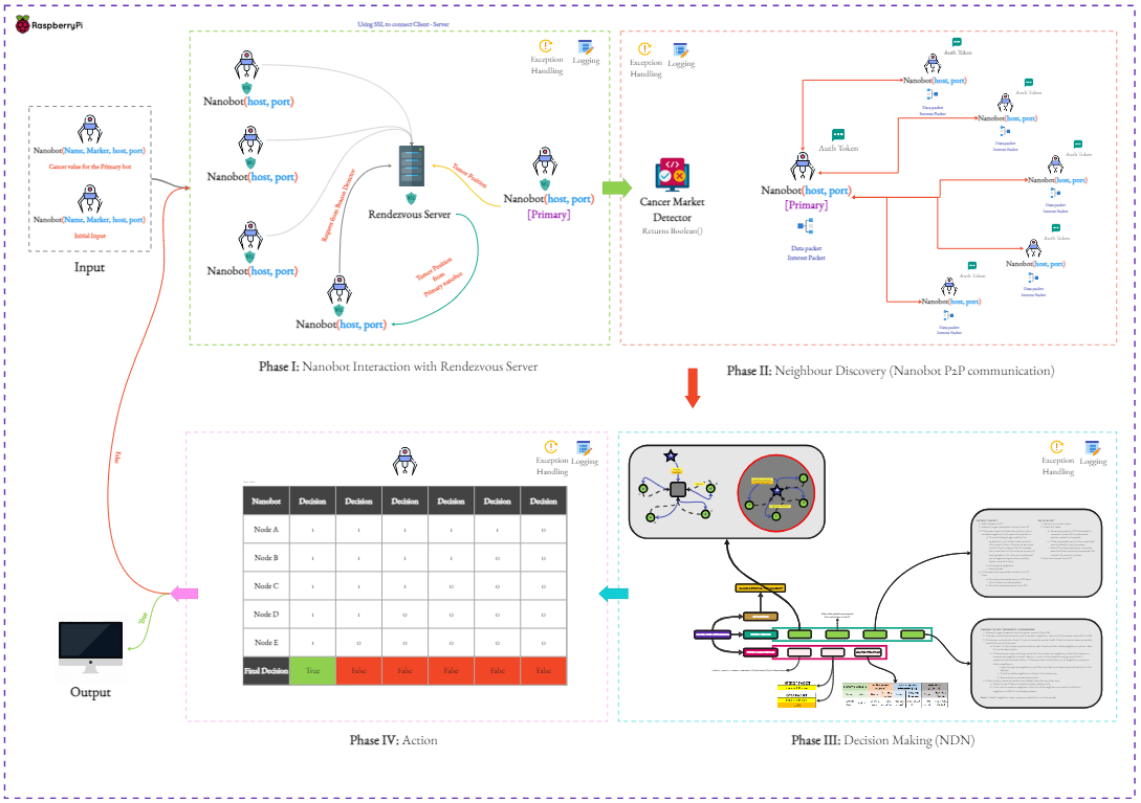
2.2 Roadmap

The overall project was divided into 4 stages; Ideation, Prototyping, Testing and Deployment.

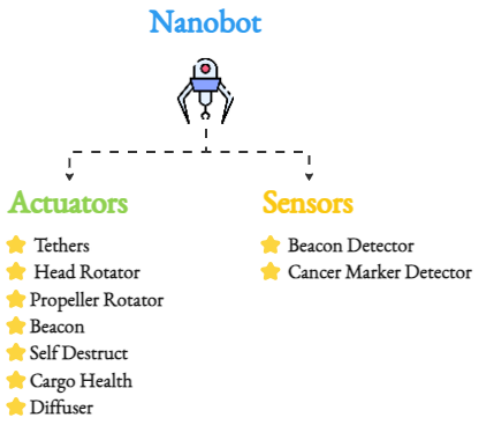
	Ideation	Prototyping	Testing	Deployment
Research	Implementation of a Research Papers	Technical Architecture on Miro	Edge cases	TCP/Socket Programming
Code	Sudo code development	Python code	Unit Testing	Python code on Raspberry Pi
debug	-	Constant Debugging	Bug Identification	Deployment Bugs

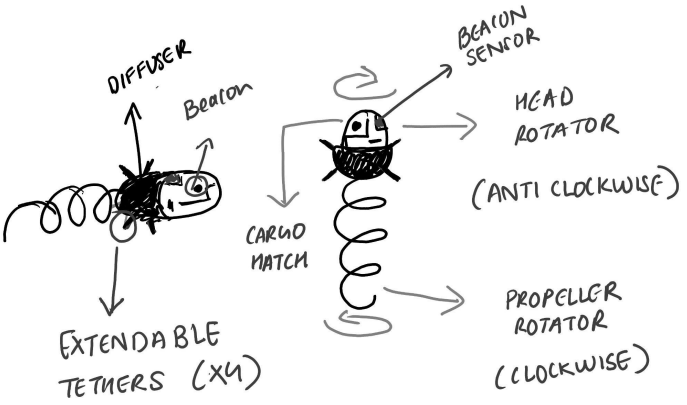
2.3 Architecture

The overall Architecture is divided into 4 phases;



2.3.1 Nanobot Architecture





Nanobots consist of two fundamental components: actuators and sensors. Each individual nanobot is equipped with a pair of sensors and seven actuators.

2.3.1.1 Actuators:

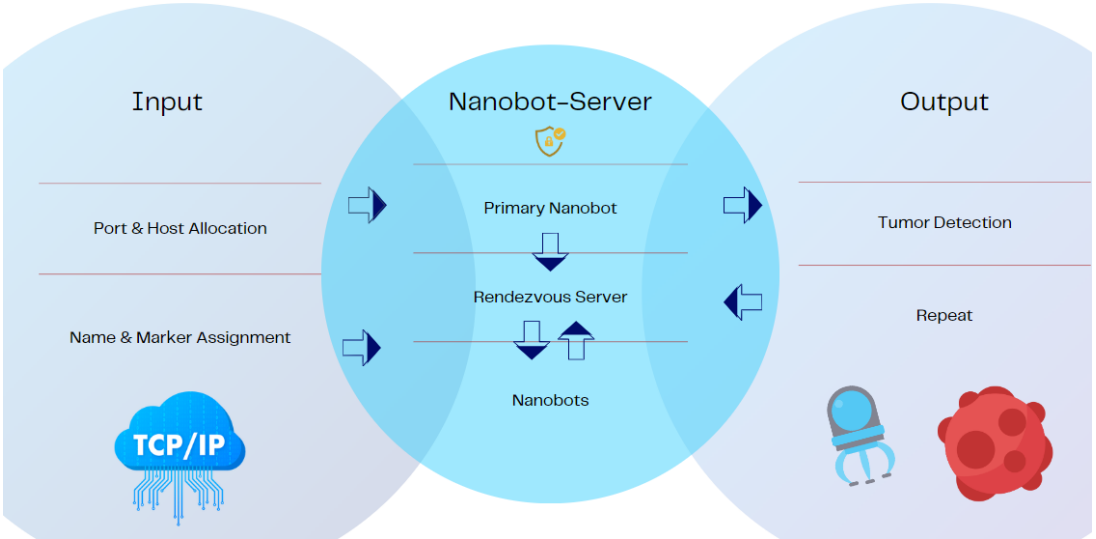
1. **Tethers:** Nanobot comes to a halt (Tumour detected)
2. **Head Rotator & Propeller Rotator:** Rotate in opposite directions to alter nanobot's speed and direction (Back-Forward).
3. **Beacon:** Signals position of primary bot that detects tumour marker.
4. **Self Destruct:** Bot destroys itself while dealing damage to cancerous tissue.
5. **Cargo Hatch:** Container which stores a blood clot inducing enzyme like Thrombin to block cancer feeding blood vessels and starve the tumour.
6. **Diffuser:** Boolean value when 1 indicates that the bot has been turned-off/broken down etc, i.e. diffused safely meaning the body can now discard it naturally.

2.3.1.1 Sensors:

1. **Beacon Sensor:** Detects beacon signals from the primary bot.
2. **Cancer Marker Sensor:** Equipped by each nanobot in a team to detect one of 5 unique chemical markers indicative of cancer.

2.4 Workflow

The workflow we employ is depicted by the following image. In the upcoming section, we delve deeper into said workflow.



Chapter 3: Solution

This section shall summarise our 'zero to almost hero' journey through this project. We began with spending a considerable amount of time on understanding fundamental networking concepts and gaining an overview about how existing TCP/IP networking works. After which we shifted our focus on to gathering information about ICN and existing implementations such as NDN.

We then started to learn about the basics of socket programming in Python and how a basic server and client communication may be set up. We then integrated both client and server logic into a single node to facilitate peer-to-peer communication. Next, our attention shifted towards figuring out how we might implement specific non-networking related aspects of our scenario such as world virtualisation, actuator and sensor representation, etc.

Finally, we moved onto implementing communication between the nodes.

3.1. ICN

This subsection presents a brief overview of our understanding of ICN. (Afanasyev et al., 2018)

Information Centric Networking (ICN) is a data-driven approach to transferring information through a network. This is different from the traditional location-driven approach that TCP/IP adopts where IP addresses marking locations of sender and receiver are prerequisite to data transfer.

Following are 3 key ideas associated with ICN that facilitates achievement of main objectives of ICN being improving content accessibility, delivery efficiency, content availability and data authenticity.

3.1.1. Content Naming: Content naming is an integral part of ICN. It lies at the heart of content-driven data transfer.

Instead of senders and receivers in TCP/IP, ICN comprises publishers and subscribers. Also, unlike TCP/IP where it's the sender that drives transfer; here, it's the receiver/subscriber that does this, as briefly explained below.

- When a **subscriber** requires a certain kind of data, it sends out an interest packet into the network with the "name" of the data it is looking for. The subscriber does not need to know where to find (at what address/device) to find this data.
- Data **publishers** create and introduce new data into the via **named** data packets.
- Devices in the network forward/propagate interest and data packets such that senders of interest packets with the same/similar "name" as existing data packets, receive them.

Thus, in ICN, content is retrieved based on names of desired data packets. This means content names must be globally unique and persistent. Following are popular namespace designs (strategies for naming data).

3.1.2 Flat Naming: This approach involves passing content to a hashing algorithm and then binding the resulting unique hash to the content along with a user generated name.

3.1.3 Hierarchical Naming: This convention is inspired from HTTP URLs where the name comprises portions each capturing some context regarding content separated by '/'s with added portions at the end that ensure uniqueness. Example: /edu/tcd/computer_science/cs101/lecture.pdf/1 could be the name assigned to the 1st data packet related to a computer science lecture released by TCD. This type of naming is more human readable and is also more scalable than the previous one in that routers can match prefixes of the names instead of the entire name which in turn reduces size of forwarding tables.

3.1.4 Attribute Based Naming: This type of naming looks like HTTP GET requests. Names are created by combining various attribute values associated with a piece of data such as date, location, creator, etc.

This kind of data naming, in-effect decouples data from location, application, transportation, and storage (Datatracker, n.d.). This makes it possible to implement the following features.

3.1.5 In Network Storage and Caching

Nodes in an ICN network may cache a copy of data packets that they receive and forward/use. This allows for replication. Thus, the same data being available at different points in the network makes access more efficient (fewer hops required) and robust to failure (even one copy of data is unavailable, chances are, another node will have another copy of the same data). This replication and improved data availability and accessibility comes at the price of need for more storage.

3.1.6 Content Level Security

In TCP/IP, security often involves securing the communication path though establishment of secure tunnels. But with ICN, it's the data content itself that is secured (Fu, Kutscher, Misra, & Li, 2018). A simple approach to this would be to modify the classic Digital Signature Algorithm (DSA) (Scenes, 2023) each publisher adds a cryptographic signature to data packets. Subscribers verify signatures using the corresponding universal public key (all subscribers know about this key) to ensure content authenticity. Since the signature is a part of the data itself, every copy of it shall also contain the same signature. Thus, the source of each copy of the data in the network can be verified (Bilal & Pack, 2020).

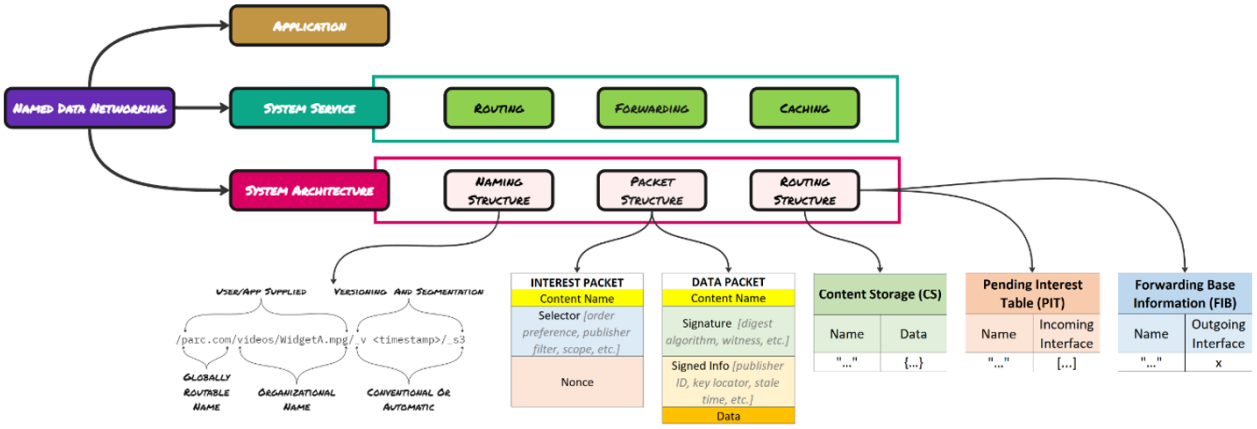
3.2. NDN

This section presents a brief overview of our understanding of NDN.

(Afanasyev, et al., 2018) (NPTEL-NOC IITM, 2023) (NPTEL-NOC IITM, 2023) (SG NDN Telkom University, 2021)

Named Data Networking (NDN) is a network architecture design that implements ICN principles.

Different applications may be built on top of the NDN architecture with characteristic naming, packet, and routing structures by leveraging system services like routing, forwarding, caching, etc, that it facilitates (SG NDN Telkom University, 2021).

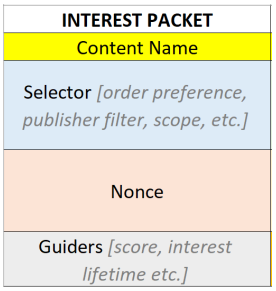


3.2.1 Naming Structure

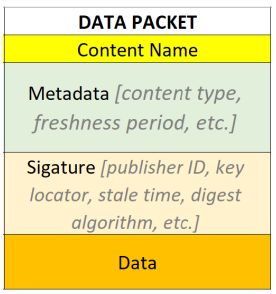
NDN does not enforce strict namespace management rules as part of the architecture. This gives one the freedom to experiment. As long as the end points in a network follow the same name assigning and querying convention, NDN can be adopted. That said, typically, NDN implementations follow a hierarchical naming convention.

3.2.2 Packet Structure

Packets may be of 2 types. An interest packet, and a data packet.



An interest packet comprises content name and can also contain conditions/rules for network devices to consider during forwarding/routing such as order in which data packets are expected, time for which this packet is valid, etc along with a nonce that facilitates linking of data packets to the interest packet that requested it.



A data packet contains content name and the data payload itself. Additionally, it may comprise meta information like the kind of content the packet contains (size, audio/video etc.), as well as security imbibing data like the signature, key locator, digest algorithm, etc.

3.2.3 Forwarding/Routing Structure

Forwarding/routing in an NDN setting is facilitated by 3 key tables/mappings as given below.

Content Storage (CS)		Pending Interest Table (PIT)		Forwarding Base Information (FIB)	
Name	Data	Name	Incoming Interface	Name	Outgoing Interface
content name	{data}	interest content name	[face1, face2, ...]	interest content name	[face1, face2, ...]

The Content Store (CS) maps content name to data received. This often acts as the cache. When a node does not contain requested data, it maps content identifier of received interest to one or more incoming interfaces in the Pending Interest Table (PIT) to keep track of all interested parties for a specific content. The Forwarding Information Base (FIB) table maintains a dictionary of interest packet content names mapped to the outgoing interface that it was forwarded to.

3.2.4 Caching

NDN implements in-network caching which means that network devices may choose to store a copy of content received within data packets locally. This means that on subsequent hits, a subscriber can find data previously requested by itself/another subscriber on a device that is physically closer to itself (reachable in few, ideally a single hop). This greatly reduces response time. However, there must be some mechanism to clear the cache from time to time to minimise storage overhead.

3.2.5 Forwarding/Routing

The core idea behind flow of packets through an NDN based network can be summarised as follows.

Receive Interest Packet	Receive Data Packet
1. Check CS. 2. If corresponding data is found in CS, send this data in a data packet to interested party/parties. 3. Else, add an interested party to PIT. 4. Check FIB to see if a known outgoing interface for this interest exists. 5. If so, forward packet out through this interface. 6. Else, pick a suitable interface (not the one the interest came from) to forward the packet to if possible. 7. If a valid outgoing interface is identified, populate FIB and forward interest packet our chosen interface.	1. Check PIT for any parties interested in received content. 2. If no pending interest is found, drop the packet. 3. For every pending interest found, forward data packet. 4. Possibly cache content in CS.

All data transfer, staying true to ICN, is receiver driven. Whenever a node (subscriber) requires information, it sends an interest packet into the network for corresponding data packet. Network devices then forward this interest packet until it gets to a node which contains requested data (publisher/in-network cache). This data then arrives back to the subscriber via reverse path forwarding (right hand side column in above table).

3.3. Our Implementation

Each team has one primary bot and four secondary bots. Each of these bots is designed to detect one of the following five markers of cancer in the cell.

- *"tumour" marker*: This is the most important marker and will be detected by the primary bot and refers to chemical markers most closely associated with particular kinds of cancer (e.g. Prostate-specific antigen (PSA) associated with Prostate Cancer (National Cancer Institute, 2021).
- *"acidity" marker*: Cancerous cells are more acidic in nature than healthy cells, hence this marker (Trafton, 2019).
- *"growth" marker*: This marker represents growth factors like Insulin-like Growth Factors (IGFs) and the Epidermal Growth Factor (EGF) that mark uncontrolled growing nature of cancerous tissue (Witsch et al., 2010).
- *"survivin"* marker: This protein counteracts growth inhibitors in the body that allows cancer to keep growing (Jaiswal et al., 2015).
- *"ecmr"* marker: Extracellular Matrix Remodelling Enzymes like metalloproteinases (MMPs) and lysyl oxidase (LOX) help reshape surroundings of the cancer cell and aid in immune suppression (Winkler et al., 2020).

The primary bot's detectors can supersede other diagnoses and trigger a collective diagnosis.

Henceforth, let **P** stand for primary and **NP** for secondary.

We had demonstrated the functioning of 1 such team with 5 bots in our interview with Prof. Ciaran.

There is also a 6th node in the network called the 'rendezvous server' which can be imagined as, perhaps, a smartwatch worn by the patient. TThis server was introduced to simulate NP bots detecting beacon signals emitted by a P bot. If this were in real life, NP bots would simply sense the beacon when it comes within range. Thus, the rendezvous server here, merely serves to pick up beacon data packets from P bots (active beacon actuator) and relay them to the NP bots that are actively searching for the beacon.

The *blood stream* is captured by a circular array whose length, and speed of blood is predefined in the config.json file. Speed of a bot is computed as `CONFIG['blood_speed'] + self.__actuators['head_rotator'] + self.__actuators['propeller_rotator']`. Movement related logic may be found in the `move()` function within the nanobot.py file.

3.3.1 Multithreading

The multi-threading execution pattern facilitates efficient switching between different processes all running on the same core. This means that time wasted while some processes wait for input or another event, can be utilised to execute another process. This, almost simultaneous execution with reduced latency improves performance and responsiveness. The python `threading` library was used for this (Iroegbu, 2023).

Number of threads were kept as low as possible to account for limited capabilities of the Raspberry Pi.

Each nanobot has the following 3 threads associated

- Main thread
- Connection thread
- Event thread.

The *connection thread* listens for incoming connections on the socket of this bot. While the *event thread* listens for the following

- Availability of a diagnosis
- Whether the bot is tethered to a cell indicating having detected a possibly cancerous tumour (P bot only).
- Presence of a beacon signal from a primary bot (NP bot only)
- Connection having become stale (if this bot was connected to a team and it has received no communication from this team in the last set x amount of time, then connection is considered stale, and state of the bot shall be reset).

Multiprocessing was also leveraged to run all nodes simultaneously in the runme.py file.

3.4 Program Flow

The control flow within the program can be divided into the following 4 phases.

Phase 1: *Environment Sensing*

Initially ***all bots are moving through the blood***.

The P bot listens/searches for a primary tumour marker. This search is simulated through requesting user input such that an input of 1 indicates tumour detected while anything else indicates nothing detected. If nothing is detected for too long (here, if the user inputs something other than 1 for a set x no. of trials) then this bot diffuses as its presence in the body is deemed unnecessary.

Meanwhile, ***NP bots listen for a beacon signal*** produced by a P bot indicating presence of a possibly cancerous structure that requires their attention. This is simulated by the NP bots

sending an interest packet of the form <node-id>/beacon/on to the rendezvous server. This interest packet is sent repeatedly for a certain, predefined number of trials at equal intervals. Once a threshold is reached, the bot is diffused as its presence in the body is deemed unnecessary.

When a P bot detects a cancer marker, it updates its content store with this latest measurement while also mapping it to the content name as *"marker/<self.marker>"*. It then tethers to the spot by slowing itself down and attaching to the tumour. Now that the bot is stationary, it activates its beacon actuator which sends a data packet with position data with the content name as <node-id>/beacon/on to the rendezvous server.

Rendezvous server forwards data packet from P node to all NP nodes of the team. Upon receiving this data packet with the substring "beacon" in the name, NP bots are now aware of the primary bot's position and other attributes. The NP bots then update their CS to include marker/<primary-marker> = 1. When each of the NP bots reach the position of P bot, they also tether themselves there and begin sensing for their specific cancer markers. This sensing action is also simulated via requesting the user for input. The NP bot updates the CS with detected marker value as marker/<self.marker> = <input-value>.

Phase 2: *Neighbour Discovery*

Once an NP bot completes environment sensing, implementation of NDN having been done over TCP/IP demands that when packets are exchanged between nodes, they know the recipient's address (host and port). In true NDN, packets are just sent through connected interfaces but since here, the bots are simulated, there are no interfaces. Thus, a mapping of each neighbour and their host, port and marker is maintained in a neighbours dictionary instead. The process by which each peer populates this table of theirs is what is referred to here as *neighbour discovery*.

NP nodes initiate neighbour discovery by sending an interest packet to the primary node (this is the only node in its neighbour table at this point) with the name <node-id>/neighbour/<desired-marker-type> for every possible marker type whose value this node is to be knowledgeable about, in order to make a diagnosis, but knows nothing about currently. All code files have access to the config.json file, where possible markers are listed. This is how the nodes know about what types of markers are collectively detected by the team.

When a P node receives such a neighbour related interest packet, it adds the sender to its own map of neighbours, updates FIB with the sender's reachability details like host and port, and adds this request to PIT. Pending interests are served by the P node only when 4 (no. of team members - 1) other such interests from bots with different NP markers have been received. This is to ensure that all required members of a team are present and ready for neighbour discovery before exchanging neighbour data. Once all (4 different, one for each other marker) peers have expressed interest for neighbour discovery, P bot sets `neighbor_discovery_complete = True` and sends contact information (name, host, port) regarding each of them (that's not the

sender themselves) to each interested party in the form of a data packet with the same content name as interest packet received.

When NP nodes receive above sent data packets, they update their neighbours dictionary and map marker content names for each possible marker to respective neighbour names in the FIB. As soon as the NP node becomes aware of 4 (no. of team members - 1) other neighbours (one for each marker), it too sets `neighbor_discovery_complete = True`.

Phase 3: *Diagnosis*

As soon as neighbour discovery is complete, NP nodes express an interest to begin diagnosis by sending an interest packet of the form <node-id>/diagnose to the P Bot.

When a P bot receives such a diagnosed interest packet, it increments its `ready_to_decide` variable by 1. When this counter reaches 4, it means that all peers (the entire) team is ready to initiate diagnosis. So, the primary marker sends an interest packet to all the interested peers stating its own interest to start diagnosis.

When each NP bot receives this single diagnose interest from P bot, it can infer that all peers are available and ready to diagnose.

All peers (P and NP bots) begin collective diagnosis involving the following steps.

1. For each possible marker, this bot first tries to fetch its value from CS.
2. If marker value is found in CS, then knowledge is updated immediately.
3. Else, for any marker whose value is yet to be in its own CS, an interest packet is created.
4. Own interest packet is added to PIT.
5. A suitable neighbour to which this packet may be forwarded is determined from FIB and neighbours dictionary.
6. Interest packet of the form <node-id>/marker/<desired-marker-type> is forwarded to the chosen best possible neighbour.

Upon receiving such an interest from another bot, each bot either forwards the packet to another peer if it, itself does not contain the marker's value in CS, or if it does, will send back a data packet with this value and same content name back to the sender.

Upon receiving a data packet in response to the interest packet that they previously sent, each bot does the following.

1. Updates own knowledge with latest marker value received.
2. Check if diagnosis can be made. Diagnosis may be made once all 5 marker values are known (not = -1) in own knowledge dictionary.
3. If diagnosis can be made, do so and set own `diagnosis` variable to 'cancer' or 'healthy'. Decision is "cancer" only if all marker values were found to be 1. Else if at least 1 value is 0, diagnosis is "healthy". This was chosen to be implemented like this, since healthy and cancerous cells both often contain multiple similar proteins/chemical markers. Thus, it was deemed better to ensure all cancer markers are detected before confirming cancer, to

reduce false positives as the main intention of such targeted treatment is to eliminate unnecessary destruction of healthy cells while attacking cancer.

Here, the packet forwarding/routing mechanism and strategy for picking the best neighbour to forward them to, remains the same at every instance for each peer.

Phase 4: *Response*

The setting of the `diagnosis` state variable is picked up by the event listener thread of each bot.

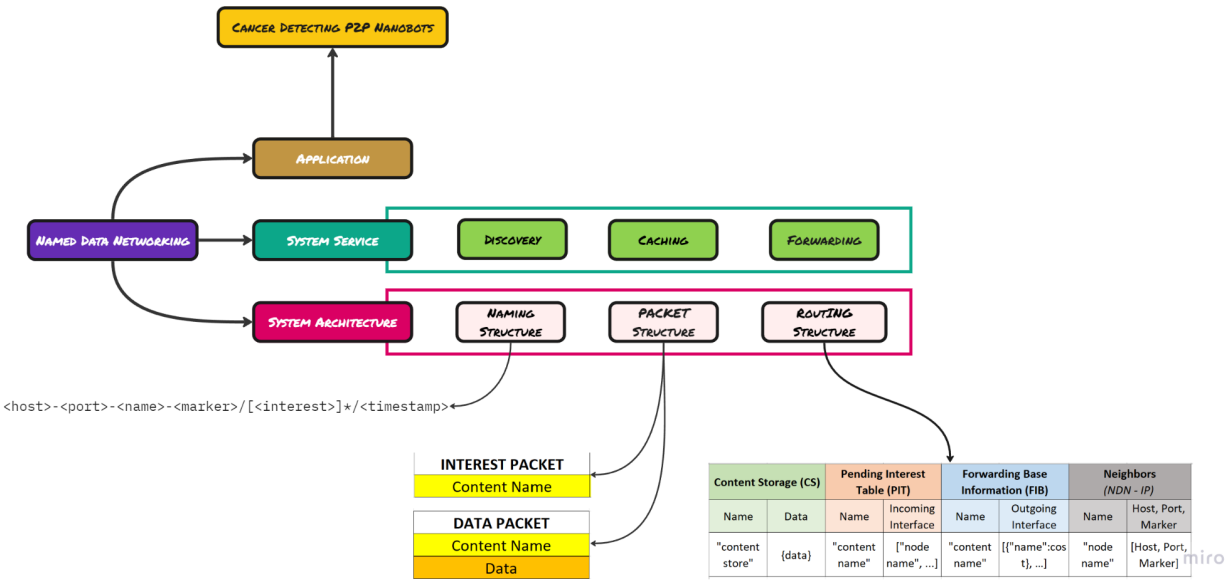
If the decision is 'cancer', then the attack sequence is initiated involving opening of the hatch (hatch actuator set to 1) that releases blood clotting enzyme thrombin that inhibits blood flow surrounding the tumour thereby starving it. This is followed by the bots destroying themselves (self destruct actuator set to 1) and as a consequence dealing damage to the tumour.

If the decision is "healthy", then the state of the bot is reset. All bots untether from the attached spot, increase speed of head and propeller rotator actuators and move through the blood steam again. The system returns to phase 1.

3.5 Attempt At ICN

Our implementation of networking between the bots in a P2P setting were inspired from NDN and attempts were made to incorporate all ICN principles.

Following image draws parallels between our implementation of communications between peers and true NDN as previously discussed in section 3.2.



3.6 System Architecture

NDN is a clean slate take on ICN that is typically not intended to be built on top of IP. So, our first attempt at NDN involved trying to work with the data layer directly using Python libraries like Scapy to facilitate communication without the need for IP addresses. This however, led to a dead end, when raw sockets could be leveraged without administrator access on the Pis. Thus, a workaround had to be designed to simulate ICN over TCP/IP.

This proved to be a challenge primarily because IP necessitates knowledge of host address and port number of the receiver before sending requests/responses via protocols like TCP. But doing this would mean that we're violating the fundamental principle of ICN which is that data transfer is content name driven and not location driven.

Naming Structure

The idea was to meet NDN and IP somewhere in the middle and implement content name driven communication while including location data like host and port as part of the unique identifier that identifies interest packets (<node-id> = <host-port-name-marker>). These IDs were used only during the neighbour discovery phase wherein, each node populates their FIB such that instead of interfaces as is normally the case with NDN, outgoing faces shall refer to names of neighbours whose host and port addresses are mapped to their names in an additional table called the neighbours table (implemented as a python dictionary). This also helps alleviate the challenge of not having physical interfaces to work with given nodes are virtual.

Thus, the naming convention adopted here (<host-port-name-marker>/[content-name]*/<timestamp>) is a hybrid design that combines both hierarchical as well as attribute based naming conventions with attributes being host, port and name.

Packet Structure

The protocol.py file contains functions like `make_interest_packet()` and `make_data_packet()` in addition to `send_tcp()`. Both interest and data packets are implemented as dictionaries containing keys "content_name" and "type" with "type" being either "interest"/"data". The "data" packet also contains an added key called "data" that maps to the data payload that is to be transmitted.

NDN is similar to IP in that they both have a thin waist functional shape. Both allow usage of any transport media for datagram packet switching (Afanasyev et al., 2018). Here, TCP connections/tunnels were the transport medium of choice.

Routing Structure

Our implementation of routing/forwards involves a CS, PIT and FIB much like classic NDN. There is however an added Neighbours table to overcome the challenge of implementing ICN over IP as mentioned under the "System Architecture" heading previously in the same section as this one.

Normally, content names are mapped to interfaces in the PIT and FIB. Here, they are mapped to names (keys) in the neighbours dictionary. The neighbours dictionary stores mapping of unique names of bots assigned during creation to their host, port and marker.

Content Storage (CS)		Pending Interest Table (PIT)		Forwarding Base Information (FIB)		Neighbors	
Name	Data	Name	Incoming Interface	Name	Outgoing Interface	Name	Host, Port, Marker
"content name"	{data}	"content name"	["node name", ...]	"content name"	[{"name":cost}, ...]	"node name"	[Host, Port, Marker]

System Services

Traditional NDN system services include Routing, Forwarding and Caching. Our system implements discovery instead of dynamic routing to populate the FIB table.

Discovery

Discovery here, refers to the "neighbour discovery" mechanism mentioned above in section 3.3.1 as part of Phase 2. At the end of this mechanism, the FIB table is populated with prefixes associated with marker value contents mapped to corresponding peers that detect these markers. This is similar to the Prefix announcement strategy of populating FIB tables (Shi, 2023).

Caching

Caching is implemented to allow for more reuse of data and to make it available closer to the subscriber to reduce latency. Out implementation adopts the simple leave-copy everywhere caching mechanism. This is not very scalable when there are many nodes in a network or when volume/variety of traffic through the network is high due to the corresponding amount of storage overhead.

In our case however, since teams of bots are always limited at a small number like 5 and information exchanges are also limited to discovery or marker value related small messages only, this simple caching strategy is justified here.

Forwarding

The protocol given below is followed when deciding whether/how to forward packets

Receive Interest Packet	Receive Data Packet

1. Map content name to sender name in PIT. 2. Attempt to serve the interested party from information in CS. 3. If requested data is unavailable in CS, **an appropriate outgoing face is chosen from FIB** for this packet to be forwarded to. To avoid taking longer paths, the cost of the last used route for this content, if any, is updated (+1). If there is a last used route X then it means that this packet has come back to this node as a result of having taken X. So, this cost update will encourage picking another possibly better route this time. Forward packet to chosen neighbour. 4. If this peer has the requested content in its CS. For every interested party in PIT send a data packet with requested data and same content name as in received interest. Remove entry corresponding to served interested parties from PIT.	1. Cache in CS. 2. Check PIT. If there are no interested parties, drop the packet. 3. For every interested party in PIT, if the interested party is not self, send a data packet with requested data and same content name as received interest packet, to them. 4. If the interested party is self, satisfy own interest. 5. Remove entry corresponding to served interested parties from PIT.

When content requested for by an interest packet is not available in the CS of the peer that received the packet, it must be forwarded as in point 3 in the left of above table. The logic implemented for picking a great interface/neighbour from FIB is as follows.

1. A node always tries to get the cheapest route (less looping back to self/less hops) for a given content name from the FIB.
2. If a known route does not exist, a random neighbour is returned. This decision is added to FIB (random neighbour selected added against content-name in fib with current cost of 0).
3. If a known route exists, a check is performed to determine if it points towards the sender itself.
    3.1. If the known route does not point to the sender of the packet itself, the cost of the route is checked.
    3.2. If cost is 0, then this is the best neighbour.
    3.3. If cost > 0, then there may be a better path meaning that other viable neighbour options may be explored.
    3.4. If there are as many existing routes for this content as there are neighbours then this means no unexplored neighbour exists. Hence, the current route, despite costing more than 0, remains to be the best option.
    3.5. If there are less routes (FIB mappings for content name) than no. of neighbours, explore other neighbours by looping through all neighbours and finding one that's not been explored yet (not in the FIB yet). If a viable neighbour is found, this is the best option. Else last selected neighbour was the best option.
4. If the known route is the same as sender, find another route.
    4.1. Check to see if there is another known viable route.
    4.2. If so, this is the new best neighbour.

4.3. Else, pick a random viable neighbour. This decision is added to FIB.

3.7 Haves and Have Nots

We are aware of the fact that this system, though capturing aspects of ICN, does not implement NDN accurately. There are primarily 2 reasons for this.

1. Implementing NDN over IP means that true NDN which does not rely on locations at all is extremely difficult, if not impossible, to set up.
2. Our understanding of ICN and NDN and IP in general was built up almost from scratch in stages throughout this project. Despite doing as much research and learning as time permitted before beginning implementation, our knowledge during development was still largely incomplete/inconsistent with the true big picture, especially in terms of implementation details like "how are FIB tables populated?" etc.

Since the demo, we have invested more time into research in and around how NDN works in practice as we've worked on this report.

This subsection thus, analyses our system (as at time of demonstration) and lists down elements of it that is similar to ICN/is advantageous as well as flaws/deviations from ICN/NDN.

Haves

Following are a list of ICN/NDN/Project requirements/features that our implementation meets.

- Requester driven data transfer.
- 5 Nodes in a network.
- At least 8 sensors/actuators.
- NDN style forwarding.
- In-network caching.
- Content name centric data transfer.
- Some security elements from section 3.8.

Have Nots

Following are a list of flaws that we are aware of. We arrived at this list upon analysis of our implementation after further learning after the demo. (Afanasyev et al., 2018)

- NDN, like IP, performs datagram packet switching which involves splitting of large data packets into smaller chunks and sending it via one or more routes such that the recipient reassembles them. Our current implementation does not account for dealing with packets greater than 2048 bytes. This design choice does not hinder our use case as it is now, since data transferred between peers is limited to small strings and boolean values. This is however not very scalable as either increase in size of packets (due to signatures, metadata etc.) or names of content can easily cause this limit to overflow.
- NDN interest packets typically contain no data regarding the sender. In our implementation, however, it does (<node-id> contains host and port). This was to facilitate NDN over IP. The part of the prefix stored in FIB, PIT and CS does not contain requested information. This data populated the neighbours table alone.

- NDN forwarding involves largest prefix matching which allows for partial prefix matches if their length is smaller than entries in FIB, PIT or CS. Our implementation matches whole prefixes only.
- Our scenario does not require inter-network communication. This was partly because in a use case such as ours, all teams of bots are deployed in the same person and therefore ideally would be part of the same network.
- NDN security involves a cryptographic signature created from packet data and name as part of the packet. This has not been implemented in the current version.
- While a timestamp is a part of packets sent, this is yet to be used to set a timeout on data within CS and FIB which should ideally be done to make the system more scalable when new data names add to the cache in order to prevent it from getting full/being too large.
- The presence of the rendezvous server means that the system is not 100% peer to peer. However, it does not contribute significantly towards the communications between the bots. In effect, it simply supports beacon detector sensors of NP bots.

3.8 Security

We attempted to implement SSL into our network. By doing so, we further secured TCP and eliminated the Man-In-The-Middle problem. We would first generate a public and a private key, the former being used to encrypt our message and the latter to decrypt the message at the target location. The private key would be sent along with the message being relayed. These tokens were 32bit tokens and thus, we validate our input. While doing so, we were also keeping logs to document what was going on. However, we failed to implement an exact version of SSL. Instead for the time being, we have hard-coded our encryption into each bot.

Chapter 4: Conclusion & Future Work

Overall, this project has been a great learning experience. While there are shortcomings to our current implementation, as is the case with any new project in the real world, we're confident that since we are now aware of most of the flaws in our work, if we were to work on it further, we can achieve a fully functional, bigger/more complex system that's much more closely aligned with ICN principles. We're happy to have gone into this project with almost no knowledge in this field, then tackled an exciting real world use case, and to have emerged with a good understanding of both traditional IP as well as ICN networks.

Future work on this project could entail implementation of the following.

- **NDN Security:** One way to implement this would be by generating a hash of the data/interest packet (name + content) along with private and public keys such that the signature resulting from having encrypted this hash is a part of the packet sent and the public key is known to all nodes in the network (feasible since each team here are only 5 strong).
- **Datagram packet switching:** This shall involve splitting long messages into multiple packets and then introducing order preference into the data/interest packet at the sender's end and consequently implementing ordered reassembly at the receiver's end.
- **Cache and FIB clearing:** Periodic length/time based clearing of caches and FIB.
- **Flexible Prefix Matching:** Longest prefix matching that also allows for partial prefix lookup.
- **Multi-Team Squads:** The rendezvous server may be improved further to manage multiple teams of bots and perform more tasks like division of bots into active and backup bots, keeping track of their health and status, etc.
- Existing libraries like PyNDN2 may be explored further to aid with implementation of more authentic NDN.
- **Further Failure Recovery:** Below are a few more ideas for improving robustness of the system at scale that leverages the rendezvous server to greater extent.
    - Rendezvous server crashes: In such cases, where the rendezvous server breaks down, we will proceed as follows.
        - We will first ping the server again and wait for a response. We shall repeat this n times.
        - Should we still not receive a response from the rendezvous server, we will shift to a backup rendezvous server.
        - Should the rendezvous server also fail, we shall diffuse all bots inside the body.
    - Primary bot crashes: We will have back up primary bots in the bloodstream. If and when a primary bot fails, we will proceed as follows.
        - We will ping the bot again and wait for a response. We shall repeat this n times.
        - After having established that the bot is, in fact, unresponsive, we will relay this information back to the rendezvous server.
        - Once the rendezvous server receives this bit of information, the remaining bots of the team will reset themselves and continue flowing in the bloodstream and looking for a primary bot.
        - Once a primary bot is identified, the bot will attach itself to its new team and relay it back to the rendezvous server.

- Secondary bot crashes: Just like backup primary bots, we will also have backup secondary bots in the bloodstream. If and when a secondary bot fails, we will proceed as follows.
  - We will ping the bot again and wait for a response. We shall repeat this n times.
  - After having established that the bot is, in fact, unresponsive, we will relay this information back to the rendezvous server.
  - Once the rendezvous server receives this information, it will ping a secondary bot that is close to the team's location.
  - This bot will now attach itself to the team.
  - The team will populate this bot with all the information they had about the last bot.

References

Afanasyev, A., Burke, J., Refaei, T., Wang, L., Zhang, B., & Zhang, L. (2018). A Brief Introduction to Named Data Networking. *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)*. 10.1109/milcom.2018.8599682

Bilal, M., & Pack, S. (2020). Secure Distribution of Protected Content in Information-Centric Networking. *IEEE Systems Journal*, *14*(2), 1921-1932. 10.1109/jsyst.2019.2931813

Blockchain Behind The Scenes. (2023, January 17). *What is Digital Signature Algorithm ? | by Blockchain Behind The Scenes*. Medium. Retrieved November 25, 2023, from https://medium.com/@21_000_000/digital-signature-algorithm-60c8318cf9b6

Datatracker. (n.d.). *Information-Centric Networking (icnrg)*. IETF Datatracker. Retrieved November 25, 2023, from https://datatracker.ietf.org/rg/icnrg/about

Fu, X., Kutscher, D., Misra, S., & Li, R. (2018). Information-Centric Networking Security. *IEEE Communications Magazine*, *56*(11), 60-61. 10.1109/mcom.2018.8539022

IITM, NPTEL-NOC. (2023, June 16). \/. YouTube. Retrieved November 25, 2023, from https://www.youtube.com/watch?v=F_NpB6trbzo&list=PLyqSpQzTE6M9voigbAaFzKiybgN7Xgvka&index=64

Iroegbu, S. (2023, November 7). *Python Multithreading: Benefits, Use Cases, and Comparison*. Pieces for Developers. Retrieved November 25, 2023, from https://code.pieces.app/blog/python-multithreading-benefits-use-cases-and-comparison

Jaiswal, P. K., Goel, A., & Mittal, R. D. (2015, April). Survivin: A molecular biomarker in cancer. *Indian Journal Medical Research*, *141*(4), 389-97. 10.4103/0971-5916.159250

Little, A. (2021, October 2). *NDN vs IP*. YouTube. Retrieved November 25, 2023, from https://www.youtube.com/watch?v=ud9YOlekH6c

Named Data Networking. (n.d.). *NFD Overview — Named Data Networking Forwarding Daemon (NFD) 22.12 documentation*. Named Data Networking (NDN). Retrieved November 25, 2023, from https://docs.named-data.net/NFD/current/overview.html

National Cancer Institute. (2021, May 11). *Tumor Markers - NCI*. National Cancer Institute. Retrieved November 25, 2023, from https://www.cancer.gov/about-cancer/diagnosis-staging/diagnosis/tumor-markers-fact-sheet

NPTEL-NOC IITM. (2022, October 2). . . . - YouTube. Retrieved November 25, 2023, from https://www.youtube.com/watch?v=OM-cBILswZk&list=PLyqSpQzTE6M9voigbAaFzKiybgN7Xgvka&index=67

NPTEL-NOC IITM. (2023, June 16). \/. YouTube. Retrieved November 25, 2023, from https://www.youtube.com/watch?v=aVZw1Qd6RLo&list=PLyqSpQzTE6M9voigbAaFzKiybgN7Xgvka&index=65

NPTEL-NOC IITM. (2023, June 16). \/. YouTube. Retrieved November 25, 2023, from https://www.youtube.com/watch?v=Eh5yHPeqBI0&list=PLyqSpQzTE6M9voigbAaFzKiybgN7Xgvka&index=69

Shi, J. (n.d.). *PrefixAnnouncement - NFD*. NDN project issue tracking system. Retrieved November 25, 2023, from https://redmine.named-data.net/projects/nfd/wiki/PrefixAnnouncement

Trafton, A. (2019, March 20). *How tumors behave on acid*. MIT News. Retrieved November 25, 2023, from https://news.mit.edu/2019/how-tumors-behave-acid-0320

Winkler, J., Abisoye-Ogunniyan, A., Metcalf, K. J., & Werb, Z. (2020, October). Concepts of extracellular matrix remodelling in tumour progression and metastasis. *Nature communications*, *11*(1). 10.1038/s41467-020-18794-x

Witsch, E., Sela, M., & Yarden, Y. (2010, April). Roles for Growth Factors in Cancer Progression. *Physiology (Bethesda)*, *25*(2), 85-101. 10.1152/physiol.00045.2009