ICN in a Peer-Peer Setting

Use Case: Precision cancer treatment using nanobots.

1. Overview

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.

1.1. Entities

There are 6 entities involved in this scenario as given below.

1. Rendezvous server.
2. Primary bot called botA detecting marker 'tumour'.
3. Non-primary bot called botB detecting marker 'acidity'.
4. Non-primary bot called botC detecting marker 'growth'.
5. Non-primary bot called botD detecting marker 'survivin'.
6. Non-primary bot called botE detecting marker 'ecmr'. Please find details in the group report.

1.2. Scenarios

Six different scenarios are emulated as described below.

1. SCENARIO: Diagnosis "healthy" with no unreliability.
    i. Here, all entities function as normal. Rendezvous server is listening for packets and so is every bot that's traveling through the blood stream.
    ii. Bot A detects 1, tethers and activates beacon. Cancer is suspected.

iii. Bots B to E that were searching for a beacon signal pick this up and also tether to the same spot when they get there.
iv. As soon as bots tether, they also sense their respective marker value in that environment. Let bots A, B, C, D and E detect marker values 1, 1, 0, 1, and 0 here respectively.
v. Bots perform neighbor discovery where they detect each other and ascertain presence of all team members.
vi. Bots begin group diagnosis and communicate marker values to each other as currently, they are aware of only the value of their own marker but require that of every other team member as well, to come to a decision.
vii. Once all bots have arrived at a decision, they take action based on the decision which in this case, is 'healthy'.
viii. Bots reset state (untether and move through the blood stream again, searching for another possibly cancerous cell).

2. SCENARIO: Diagnosis "healthy" with non-primary bots being unreliable.
    i. Rendezvous server is listening.
    ii. Bot A detects 1, tethers and activates beacon. Cancer is suspected.
    iii. Bots B, C and E pick this up and arrive at the spot to also tether and detect their marker values respectively.
    iv. Bot D is late. It starts listening, picks up the beacon and tethers to the same spot.
    v. Bot B becomes unavailable/fails (simulated by killing the process). All other bots lose connection with it.
    vi. Rest of the bots wait for a set amount of time x and try again for a set no. of trials y (as set in config files). If a team member bot does not respond in this time frame, then they reset state and move on to another spot. Due to in-network caching, marker values may be available, but other bots still wait for B and move on if it isn't back in time because if the decision is cancer, they must attack as a team to be effective.
    vii. Bot B becomes available again after some time and team once again tethers to a spot where they suspect cancer.
    viii. This time with all bots available, neighbor discovery is complete and they proceed to exchange marker information and complete diagnosis.
    ix. Let detected marker values of A, B, C, D, and E have been 1, 0, 0, 0, and 0 this time. Thus, decision was "healthy".
    x. Bots reset state and continue searching.
3. SCENARIO: Diagnosis "healthy" with rendezvous server being unreliable.
    i. Rendezvous server is listening.
    ii. Bot A detects 1, tethers to the spot and activates beacon. Cancer is suspected.
    iii. Bots B and C that were listening pick this up and also tether to the spot.
    iv. Server goes down.
    v. Bots D and E cannot pick up the beacon (if this were practically implemented, there would be no need for a server to facilitate beacon sensing since actual

sensors onboard the bot would simply pick this up when in range; for the purposes of this simulation however, the rendezvous server is required to simulate simulate this beacon pick up).

vi. Due to connection loss, the bots reset state.
vii. Server comes back up.
viii. All bots once again find each other and tether to a spot.
ix. Neighbor discovery is complete.
x. Diagnosis is complete. Let marker values of A, B, C, D, and E have been 1, 0, 1, 1, and 1 respectively. Thus decision is 'healthy'.
xi. State is reset and search continues.

4. SCENARIO: Diagnosis "cancer" with primary bot being unreliable.
    i. Server is listening.
    ii. Bot A picks up 1, tethers and activates beacon. Cancer is suspected.
    iii. Bots B, C, D, and E pick this up but bot A goes down.
    iv. Bots B, C, D, and E have to move on as connection with the team cannot be established. They reset state.
    v. Bot A comes back up and searches, tethers and activates beacon again.
    vi. Other bots pick this us and neighbor discovery is completed.
    vii. Diagnosis is also completed. Let marker values of A, B, C, D and E have been 1, 1, 1, 1, and 1 this time. Thus the diagnosis was cancer.
    viii. Bots collectively decide to attack and open their hatch to release disruptive enzyme thrombin and self destruct at the same spot thereby dealing damage to the cancerous cell.
5. SCENARIO: Diagnosis "cancer" with no unreliability.
    i. Server is listening.
    ii. Bot A picks up 1, tethers and activates beacon. Cancer is suspected.
    iii. Bots B, C, D, and E pick up the beacon and tether at the same spot.
    iv. Neighbor discovery is complete.
    v. Diagnosis is complete. Let marker values of A, B, C, D and E have been 1, 1, 1, 1, and 1 this time. Thus the diagnosis was cancer.
    vi. Bots collectively decide to attack and open their hatch to release disruptive enzyme thrombin and self destruct at the same spot thereby dealing damage to the cancerous cell.
6. SCENARIO: No cancer suspicion so no need for diagnosis.
    i. Server is listening.
    ii. Bot A never picks up 1 (this is simulated by it picking up 0 for a set no. of trials, here 2). No cancer is suspected.
    iii. Bot A diffuses itself as it is no longer required in the body.
    iv. Bots B, C, D and E that were searching for a beacon, keep doing so for a set amount of time and trials (here 2) after which they too diffuses themselves on account of np longer being required in the body.

2. Run Instructions

Original demonstration involved manual input of marker values 1/0 on separate terminal windows (one for each entity) for better readability. Please find a link to this video here: https://drive.google.com/file/d/1kFizBfB--VzE7GYL8Q8Azu2tmKfvhihN/view?usp=sharing.

2.1. Pre-Requisites

Kindly ensure you navigate to the folder containing the following files.

- config.json
- nanobot.py
- protocol.py
- rendezvous_server.py
- runme.py
- run.sh

You may run the script on either one of our 2 Pi's. Paths to the desired folder on each is stated below.

- On Pi 22, please execute: cd project3.
- On Pi 32, please execute: cd scp3.

In order to make it easier to run above scenarios that were originally demonstrated manually, they have been automated and can now be run using a single command.

- If you are unable to due to restricted read-write-execute permission, please run chmod +x run.sh and try again.
- Please execute: ./run.sh.
- Should you wish to run any of the python files manually, kindly use the command python3 instead of python on the pi as version differences seem to be causing syntax errors otherwise.

2.2. Expected Output

IMPORTANT!

- Program waits at various points during execution. This is intentional. The pauses either denote time after which a DOWN entity comes back UP or allows time for the scenarios to play out. Kindly do not kill the terminal. Please wait for program to proceed. All scenarios should play out in around 5 mins (max 10 mins).
- Output reading "Connection Error" is also to be expected on the terminal. This is intentional and marks the moments when connection is hindered as expected when a an entity becomes unreachable.
- Some looping/repetition of outputs may be observed. This is because when running automatically, the bots always detect the specific scenario's values again and again after diagnosis if the decision was "healthy" since in that case state resets. When run manually, this is stopped when there are no more positive markers (Bot A detects 0 as in scenario 6 here). Also, due to multi-threading and multi-processing, order of statements printed may not be in the right order every time. This is also expected, is

not extreme and does not violate the logic in any manner (they still run as expected, the printing is sometimes out of sync).

Upon running the run.sh file, the following results are expected.

- The scenarios start playing out. Beginning of each scenario is indicated by text that says "SCENARIO [number]: [description]" and its end is marked by text that says "SCENARIO [number] COMPLETE :)".
- A logs folder wherein all activity (including packet transfers) are logged.
- Output indicative of status of scenario is printed on the terminal. Drawing parallels with section 1.2. above, following are meaning of the various printed statements.
    - UP! => Process starting.
    - DOWN! => Process killed.
    - Listening on [host] port [port]. => Server listening.
    - Searching for beacon... => Non primary bot searching for beacon signal.
    - Tethered to [position] => Bot has tethered.
    - Sensing cancer marker value: [marker value 1/0] => Value sensed by the value.
    - Neighbor discovery complete => Neighbor discovery is completed.
    - Servicing all neighbor interest packets => Neighbor discovery is completed.
    - Ready to diagnose. => Bot is ready to begin diagnosis.
    - Stale connection => Connection with team member has timed out due to no response.
    - State rest => Bot is resetting own state to go back to searching phase.
    - Beacon turned off. => Primary bot has turned off the beacon signal.
    - Diagnosis = cancer. => Diagnosis decision made by bot is 'cancer'.
    - Diagnosis = healthy. => Diagnosis decision made by bot is 'healthy'.
    - Hatch open. Thrombin deployed. => Bot has activated hatch actuator and deployed the cargo carried inside.
    - Preparing to self-destruct. => Bot is self destructing and damaging cancer cell in the process.
    - Detonated at position [position]. => Bot self destructed at the specified position (index in bloodstream array).
    - Diffused. Goodbye. => Bot was deemed unnecessary in the body and has thus safely diffused (turned off/broken down) itself so that body may naturally discard it.

3. Code Files

Following is a quick overview of files in this repository and what they do.

- run.sh: This file runs executes the python runme.py script in the linux environment,
- runme.py: This file runs all simulated scenarios.
- rendezvous_server.py: This file contains the definition of the Server class. The rendezvous server that facilitates beacon signal detection, is an instance of this class. It contains the following methods.

    - __init__(...)``: This method initializes a Server` object and creates a TCP socket that is bound to given host and port. Dictionaries that shall store primary and non-primary bot data is also initialized here.
    - listen(...): This function triggers listing on port for connections.
    - handle_incoming(...): This function catches incoming data packets and depending on whether they're interest/data packets, sends it to following 2 functions for specialized handling.
    - handle_data_packet(...): This function handles data packets.
    - handle_interest_packet(...): This function executes slightly different logic for various kinds of interest packets received.
    - serve_beacon_interested_parties(...): This function looks up whether there are existing non-primary nodes searching for an available beacon. It forwards data regarding the beacon to them, if found.
- protocol.py: This file contains functions that facilitate creation and transfer of data/interest packets.
    - send_tcp(...): Given a message which may be an interest or data packet, encodes it as utf-8 and sends it to given host and port.
    - make_interest_packet(...): Creates an interest packet from given content name.
    - make_data_packet(...): Creates an interest packet from given data and content name.
- nanobot.py: This file contains logic regarding all functionalities of the nanobot entity each of whom act as both client and server as part of the peer to peer network.
    - __init__(...): Initializes nanobot's state variables and starts threads that listen for connections and events.
    - __print_tables(...): Used for debugging only to print FIB, CS, PIT, Neighbors.
    - __print(...): Custom print where the name of the bot is printed before content to print.
    - sense_cancer_marker(...): Detects current value of own cancer marker.
    - add_to_pit(...): Maps incoming faces/neighbors to content name in PIT.
    - get_from_pit(...): Gets faces/neighbors mapped to content name from PIT.
    - add_to_fib(...): Maps outgoing faces/neighbors to content name in FIB while updating cost accordingly.
    - get_from_fib(...): Gets FIB mappings for content name.
    - get_random_viable_neighbor(...): Gets a random neighbor from FIB that is not sender itself.
    - get_fwd_neighbor(...): Get's best possible neighbor from FIB to which given content can be forwarded to.
    - add_to_cs(...): Adds data to the CS.
    - get_from_cs(...): Adds data from the CS.
    - initiate_attack_sequence(...): Executes attack sequence of a bot involving opening hatch and self destructing.
    - initiate_state_reset(...): Executes state reset by setting state variables back to what they were like during initialization.

    - satisfy_interest(...): Satisfies own interest using a received marker value related data packet which is to diagnose.
    - handle_incoming(...): This function catches incoming data packets and depending on whether they're interest/data packets, sends it to following 2 functions for specialized handling.
    - handle_data_packet(...): This function handles data packets.
    - handle_interest_packet(...): This function executes slightly different logic for various kinds of interest packets received.
    - listen_conn(...): Runs in the background (different thread) to listing for connections.
    - listen_event(...): Listens for various kinds of events (different thread) like availability of diagnosis, beacon signals and stale connections.
    - move(...): Implements moving through blood stream logic.
    - start_diagnosis(...): Starts diagnosis process wherein interest packets are sent to peers in order to get data regarding their marker values.
    - start_neighbour_discovery(...): Non primary bots call this function to start neighbor discovery by sending interest packets to the primary bot.
    - set_sensors(...): Setter for various sensors of the bot. Implements variations based on type of sensor(cancer_marker/beacon), value (0/1) and bot (primary/non-primary).
    - set_actuator(...): Setter for various actuators of the bot. Implements variations based on type of actuator(tethers/beacon/cargo_hatch/self_destruct/diffuser), value (0/1) and bot (primary/non-primary).

- config.json: This file contains configuration settings for globally visible variables like markers, timeout, blood_speed, markers, primary_marker, etc.