

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA ĐIỆN - ĐIỆN TỬ  
BỘ MÔN VIỄN THÔNG



## BÁO CÁO ĐỒ ÁN 2

TÌM HIỂU GIAO THỨC MAC TSCH, 6TOP VÀ 6TiSCH CHO  
MẠNG CẢM BIẾN KHÔNG DÂY  
(MAC PROTOCOLS TSCH, 6TOP AND 6TiSCH FOR WSNs)

Sinh viên thực hiện: Nguyễn Vũ Huy - MSSV: 2113527  
Giảng viên hướng dẫn: TS. Võ Quế Sơn

Thành phố Hồ Chí Minh, tháng 12 năm 2024

## LỜI CẢM ƠN

Trong khoảng thời gian học tập và thực hiện Đề án 2 tại Trường Đại học Bách Khoa TP. HCM, nhận được sự hướng dẫn và giảng dạy tận tình của thầy Võ Quê Sơn, người đã truyền đạt những kiến thức về mặt lý thuyết lẫn thực tiễn áp dụng trong học tập trong suốt Đề án 2 cũng như môn học IoT. Đồng thời, cùng sự cố gắng và chăm chỉ nghiên cứu, tìm tòi của em cũng là một trong những yếu tố góp phần giúp em hoàn thiện sản phẩm này.

Từ những thành quả đạt được, em xin chân thành cảm ơn những lời đóng góp ý kiến của thầy. Bên cạnh đó, do kiến thức của em còn hạn hẹp nên không tránh khỏi việc mắc phải những sai sót trong quá trình thực hiện bài báo cáo. Em rất mong nhận được sự thông cảm và đóng góp ý kiến của thầy để em có thể tiếp thu và cải thiện cho những sản phẩm tiếp theo. Một lần nữa, em xin chân thành cảm ơn!

Cuối cùng em kính chúc quý thầy sức khỏe, phát triển hơn trong sự nghiệp cao quý và đạt được nhiều kết quả tốt đẹp trong công việc.

TP.HCM, ngày 20, tháng 12, năm 2024

Nguyễn Vũ Huy

# MỤC LỤC

PHẦN I: GIỚI THIỆU ĐỀ TÀI . . . . .	1
1.1. Tổng quan nội dung báo cáo đồ án 2 . . . . .	1
1.2. Hệ điều hành Contiki-NG . . . . .	2
1.2.1. Tổng quan hệ điều hành Contiki và Contiki-NG	2
1.2.2. Các thành phần của một dự án trong Contiki-OS	3
PHẦN II: TÌM HIỂU LÝ THUYẾT . . . . .	4
2.1. Các giao thức trong Netstack của Contiki-NG . . . . .	4
2.2. Time Slotted Channel Hopping (TSCH) . . . . .	4
2.3. Routing Protocol for Lossy and Low-power network – RPL	9
2.4. Giao thức 6top . . . . .	11
2.5. Giao thức UDP . . . . .	14
2.6. 6TiSCH . . . . .	16
PHẦN III: PHÂN TÍCH CÁC VÍ DỤ 6TiSCH . . . . .	18
3.1. Ví dụ Simple Node . . . . .	18
3.2. Ví dụ ETSI-Plugtest-2017 . . . . .	20
3.3. Ví dụ 6top(6P) . . . . .	22
3.4. Ví dụ Custom schedule . . . . .	23
3.5. Tạo ví dụ Hello-world chạy TSCH và RPL trên Contiki-NG	25
PHẦN 4: Thực hiện thiết kế mô phỏng mạng 6TiSCH . . . . .	27
4.1. Thiết kế lựa chọn Netstack cho project . . . . .	27
4.2. Thực hiện project . . . . .	28
4.2.1. Mô tả project . . . . .	28

4.2.2 Cấu hình file project-conf.h và Makefile . . . . .	28
4.2.3. Lưu đồ giải thuật cho Coordinator . . . . .	36
4.2.4. Lưu đồ giải thuật Node sensor . . . . .	40
4.3. Mô phỏng project trên cooja . . . . .	45
<b>PHẦN 5: Thực hiện triển khai mạng 6TiSCH trên phần cứng CC2538dk</b>	<b>48</b>
5.1. Thực hiện nạp lên phần cứng CC2538dk . . . . .	48
5.2. Chạy thực nghiệm kết quả mô phỏng . . . . .	48
5.3. Nhận xét kết quả thực tế . . . . .	50
<b>PHẦN 6: Kết luận và hướng phát triển</b> . . . . .	<b>52</b>
6.1. Kết luận . . . . .	52
6.2. Hướng phát triển . . . . .	52
<b>PHẦN 7: Tài liệu tham khảo</b> . . . . .	<b>54</b>

# DANH SÁCH HÌNH ẢNH

1	Cấu trúc của một Contiki-NG repository . . . . .	3
2	Contiki-NG network stack . . . . .	4
3	Sơ đồ Timeslot TSCH chuẩn và ví dụ . . . . .	5
4	Hai loại liên kết giữa các node trong TSCH . . . . .	7
5	Giản đồ timeslot của một hệ thống 4 node sử dụng giao thức TSCH . . . . .	8
6	Minh họa của ví dụ 1 . . . . .	8
7	Mô hình mạng của giao thức RPL . . . . .	9
8	Giản đồ quảng bá gói tin để thành lập mạng DODAG của giao thức RPL	10
9	6P Transaction Patterns . . . . .	12
10	Vị trí của 6P trong mạng . . . . .	13
11	Mô hình khung dữ liệu thông tin truyền của 6P . . . . .	13
12	Datagram của giao thức UDP . . . . .	15
13	Netstack của một Node sử dụng 6TiSCH . . . . .	16
14	Topology của một mạng sử dụng 6TiSCH . . . . .	17
15	Kết quả mô phỏng ví dụ Simple-node . . . . .	18
16	Lưu đồ giải thuật example Simple-node . . . . .	19
17	Kết quả mô phỏng ETSI-Plugtest-2017 . . . . .	21
18	Kết quả chạy shell trong mô phỏng ETSI-Plugtest-2017 . . . . .	21
19	Kết quả mô phỏng ví dụ 6top . . . . .	22
20	Lưu đồ giải thuật giải thuật 6top . . . . .	23
21	Kết quả mô phỏng ví dụ Custom schedule . . . . .	24
22	Netstack được sử dụng cho mỗi node cảm biến mạng không dây . . . . .	27
23	Kết quả chạy mô phỏng log RPL . . . . .	33
24	Kết quả chạy mô phỏng log TSCH . . . . .	34
25	Lưu đồ giải thuật Coordinator node . . . . .	36

26	Lưu đồ hàm xử lý UDP packet received . . . . .	39
27	Lưu đồ giải thuật Node sensor . . . . .	41
28	Lưu đồ giải thuật hàm gửi data và hàm ping . . . . .	42
29	Cấu trúc gói tin gửi dữ liệu của Node sensor . . . . .	43
30	Cấu trúc gói tin Ping của Node và Coordinator . . . . .	43
31	Mô hình mạng mô phỏng . . . . .	45
32	Kết quả mô phỏng với hơn 1000 gói . . . . .	46
33	Kết quả mô phỏng ở những gói tin thấp . . . . .	46
34	Kết quả nạp code lên CC2538dk thành công . . . . .	48
35	Hình ảnh bố trí thực tế . . . . .	49
36	Sơ đồ minh họa bố trí phần cứng thực tế . . . . .	49
37	Kết quả sau khi chạy thực tế . . . . .	50

# DANH SÁCH BẢNG

1	So sánh giữa MRHOF và OF0 . . . . .	30
2	Bảng cấu hình thông số TSCH . . . . .	31
3	Các mức log và ý nghĩa . . . . .	32
4	Log TSCH và ý nghĩa mô tả . . . . .	35

## PHẦN I: GIỚI THIỆU ĐỀ TÀI

### 1.1. Tổng quan nội dung báo cáo đồ án 2

Đồ án trình bày về việc tìm hiểu về các giao thức TSCH, 6top và 6TiSCH. Sau đó triển khai chạy các mô phỏng và thực hiện chạy trên phần cứng một mạng 6TiSCH. Mục tiêu là để phân tích các ưu điểm và tính ứng dụng của giao thức TSCH và mạng 6TiSCH trong các thiết bị trong Mạng Cảm Biến Không Dây (Wireless Sensor Networks) và Internet Vạn Vật Công Nghiệp (Industrial Internet of Things - IIoT).

Mô hình bao gồm các node cảm biến có bộ xử lý trung tâm là SoC CC2538 của Texas Instruments, được lập trình bằng hệ điều hành mã nguồn mở Contiki-NG, giao tiếp với nhau thông qua các netstack được hệ điều hành hỗ trợ.

Nội dung bài báo cáo được trình bày bao gồm 5 phần chính:

- **PHẦN 1: Giới thiệu đề tài**

Nội dung của Phần 1 bao gồm giới thiệu tổng quan mục tiêu đề tài và tìm hiểu về hệ điều hành Contiki-NG.

- **PHẦN 2: Tìm hiểu lý thuyết**

Nội dung của Phần 2 bao gồm tìm hiểu các lý thuyết về giao thức TSCH, 6top, 6TiSCH và các giao thức trong lớp Netstack.

- **PHẦN 3: Phân tích các ví dụ 6TiSCH**

Nội dung của Phần 3 bao gồm thực hiện chạy các ví dụ có sẵn trong thư mục 6TiSCH của hệ điều hành Contiki-NG, phân tích và mô phỏng kết quả của các ví dụ.

- **PHẦN 4: Thực hiện thiết kế mô phỏng mạng 6TiSCH**

Nội dung Phần 4 bao gồm triển khai từ các ví dụ đã chạy ở Phần 3 thành một mạng 6TiSCH riêng chạy các giao thức trong lớp Netstack, truyền dữ liệu trong mạng và kiểm chứng và đánh giá các thông số thông qua mô phỏng

- **PHẦN 5: Thực hiện triển khai mạng 6TiSCH trên phần cứng CC2538dk và Kết luận**

Phần 5 thực hiện triển khai mạng 6TiSCH đã mô phỏng ở Phần 4 lên phần cứng thực tế để kiểm chứng và đánh giá thực tế so với mô phỏng, đưa ra các kết luận đối với mạng 6TiSCH đã triển khai.

- **Phần 6: Kết luận và hướng phát triển**

## 1.2. Hệ điều hành Contiki-NG

### 1.2.1. Tổng quan hệ điều hành Contiki và Contiki-NG

Contiki OS, phát triển từ năm 2003 và trở thành mã nguồn mở vào năm 2006, được thiết kế đặc biệt cho các hệ thống mạng cảm biến không dây với yêu cầu nghiêm ngặt về tài nguyên, điển hình là các thiết bị có kích thước mã từ 10kB đến 100kB. Hiện tại, Contiki là nền tảng duy nhất cung cấp khả năng mô phỏng mạng cảm biến không dây toàn diện thông qua Cooja. Tuy nhiên, Contiki OS vẫn tồn tại một số hạn chế:

- Các nền tảng cũ như vi điều khiển 8-bit và 16-bit có tài nguyên hạn chế, trong khi vi điều khiển ARM Cortex-M 32-bit, vốn phổ biến hiện nay, lại không được hỗ trợ tốt.
- Hỗ trợ các giao thức không tiêu chuẩn, trong khi mạng cảm biến không dây (WNS) đang phát triển để tương thích với các tiêu chuẩn quan trọng.

Với mục tiêu khắc phục những hạn chế này, Contiki NG ra đời vào tháng 11 năm 2017, Contiki-NG bắt đầu là một nhánh của Contiki OS và vẫn giữ nguyên nhiều tính năng ban đầu, tập trung vào việc phát triển và nâng khả năng ứng dụng bằng cách áp dụng các tiêu chuẩn mới:

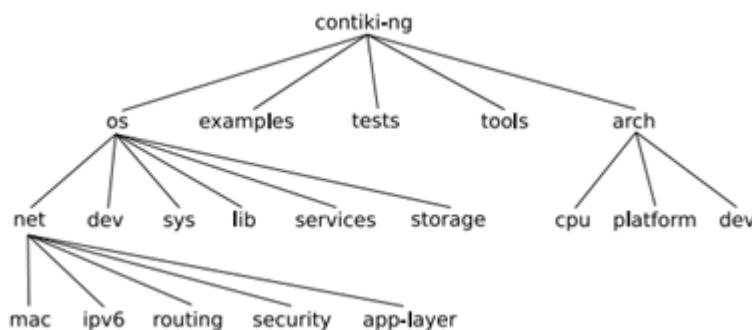
- Hỗ trợ nhiều giao thức tiêu chuẩn như IEEE 802.15.4 TSCH, 6LoWPAN, 6TiSCH, RPL, CoAP, MQTT, LWM2M.
- Tương thích với các nền tảng phần cứng hiện đại, bao gồm Arm Cortex-M của Nordic Semiconductor, NXP, OpenMote, Texas Instruments và Zolertia.
- Đảm bảo độ tin cậy và bảo mật nhờ các cập nhật hiện đại.

Contiki NG giữ lại những tính năng cốt lõi của Contiki OS, như kernel dựa trên sự kiện và cấu trúc thư viện, đồng thời thực hiện các cải tiến nhỏ đối với phần mềm mạng thành phần như 6LoWPAN và RPL (trong Contiki NG gọi là RPL-Classical).

### **1.2.2. Các thành phần của một dự án trong Contiki-OS**

Contiki NG được tổ chức thành hai phần chính: phần không phụ thuộc vào phần cứng (hardware-independent) và phần phụ thuộc vào phần cứng (hardware-specific):

- Hardware-independent: Bao gồm các thành phần có thể triển khai trên nhiều nền tảng khác nhau mà không bị ràng buộc bởi phần cứng cụ thể, như kernel, bộ định thời phần mềm, thư viện cấu trúc dữ liệu, và các giao thức mạng.
- Hardware-specific: Gồm các thành phần cần thiết để hệ điều hành có thể chạy trên một nền tảng cụ thể. Bao gồm các giao diện với phần cứng như bộ định thời, giao diện radio (cả trên chip và ngoài chip), cũng như các thiết bị ngoại vi khác như UART, SPI, I2C, đèn LED, nút nhấn, và cảm biến.



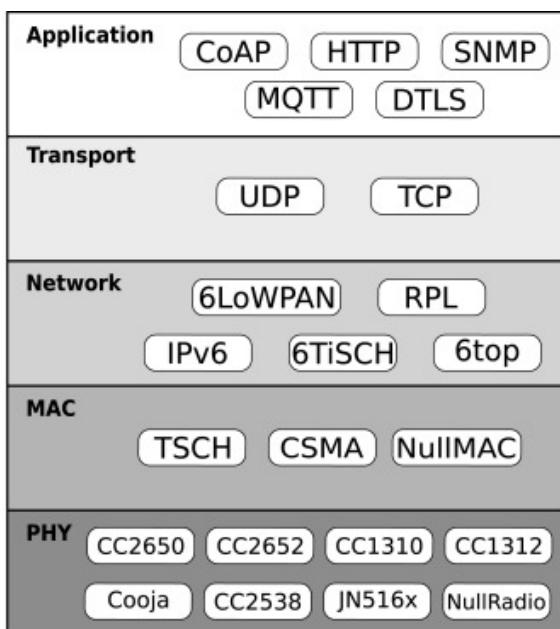
Hình 1: Cấu trúc của một Contiki-NG repository

Trong đó : os chứa platform-independent code và arch chứa hardware-specific drivers.

## PHẦN II: TÌM HIỂU LÝ THUYẾT

### 2.1. Các giao thức trong Netstack của Contiki-NG

Như đã đề cập trong Phần 1.2.1, một trong những mục tiêu chính của Contiki-NG là cung cấp mạng lưới công suất thấp đáng tin cậy, tuân thủ tiêu chuẩn cho các thiết bị nhúng không dây bị hạn chế về mặt tài nguyên. Hình 2 minh họa tổng quan về ngăn xếp mạng Contiki-NG.



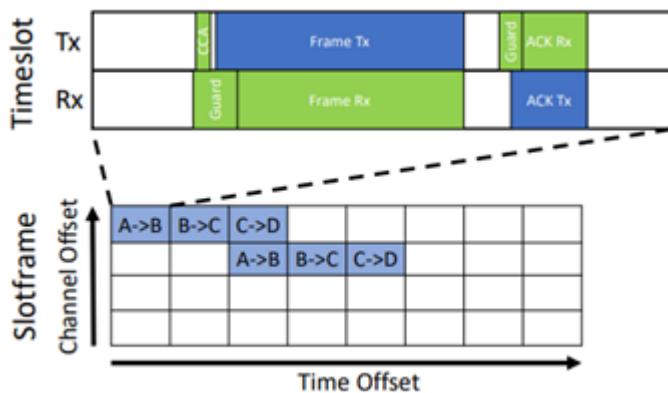
Hình 2: Contiki-NG network stack

### 2.2. Time Slotted Channel Hopping (TSCH)

Time Slotted Channel Hopping (TSCH) là một lớp MAC (Medium Access Control) của tiêu chuẩn IEEE 802.15.4-2015. Thường được sử dụng trong các mạng cảm biến không dây. Là một giao thức truyền thông mạnh mẽ với các ưu điểm là giảm nhiễu, tiết kiệm năng lượng, và cải thiện độ tin cậy trong truyền dữ liệu. Đây là chế độ kết hợp giữa kỹ thuật TDMA (Time-Division Multiple Access) và FDMA (Frequency-Division Multiple Access):

- Việc chia các timeslot và thực hiện theo slotframe làm giảm năng lượng tiêu thụ radio (Radio Duty Cycle – RDC). Thời gian trong TSCH được chia thành các khung thời gian (slotframe), và mỗi khung thời gian bao gồm nhiều khe thời gian (timeslot). Mỗi timeslot được phân bổ cho các hoạt động như truyền, nhận hoặc giữ im

lặng.



Hình 3: Sơ đồ Timeslot TSCH chuẩn và ví dụ

- Một timeslot thường có thời gian khoảng 10ms, khoảng thời gian trống trước, giữa và sau khung dành cho xử lý hàng đợi và mã hóa. Timeslot lặp lại theo thời gian và có chiều dài hữu hạn bất kỳ
- Việc nhảy tần (Hopping Channel) sẽ giảm thiểu tác động của can nhiễu do các thiết bị khác sử dụng cùng tần số gây ra. Bên cạnh đó dựa vào sự phân chia tần số, nhiều node có thể cùng sử dụng một timeslot để giao tiếp với node khác.

Các node trong mạng TSCH được đồng bộ hóa hoàn toàn, mỗi node có một khe thời gian cụ thể để trao đổi thông tin với các node lân cận. Ban đầu các node được đồng bộ bằng cách lắng nghe các EB (Enhance Beacon) được node gốc phát thông điệp tới tất cả các node trong mạng (root node broadcast) nhưng vẫn giữ được lịch trình thông qua liên lạc trong các timeslot của các node lân cận.

Enhanced beacons giúp thiết lập và duy trì đồng bộ thời gian giữa các nút trong mạng. Các beacon này mang theo thông tin về lịch trình truyền thông (schedule) và các thông số quan trọng khác mà các nút cần để đồng bộ hóa hoạt động của mình.

Trong TSCH, root node có vai trò quan trọng trong việc đồng bộ hóa mạng. Root node sẽ thực hiện broadcast các **Beacon frames** hoặc **Enhanced Beacon** nhằm:

- Đồng bộ thời gian: Các gói tin này chứa thông tin về thời gian và lịch trình (schedule) của TSCH. Các node khác sẽ dựa vào thông tin này để đồng bộ thời gian với

root node.

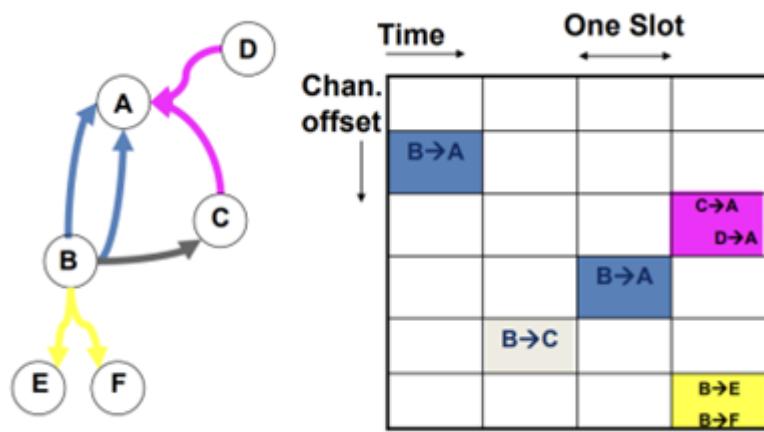
- Thông tin về cấu hình mạng: Ngoài việc đồng bộ thời gian, các gói tin broadcast có thể chứa thông tin về cấu trúc kênh (channel hopping sequence), độ dài timeslot, và các thông số quan trọng khác.
- Trong một mạng TSCH, root node có thể phát sóng các gói tin Enhanced Beacon theo chu kỳ để các node khác trong mạng nhận và đồng bộ thời gian chính xác. Nếu một node mới tham gia vào mạng, nó sẽ lắng nghe các gói broadcast này để tự đồng bộ hóa và bắt đầu tham gia truyền dữ liệu.

Các node trong cùng một network sẽ được cấp một Time-slot trong Slot-frame để giao tiếp với các node khác. Time-slot và Slot-frame có những đặc điểm như sau:

- Sync Phase: Đồng bộ hóa thời gian giữa các node trong mạng. Trong phase này, các node sử dụng gói tin Beacon hoặc Enhanced Beacon để đảm bảo các thiết bị duy trì cùng một mốc thời gian.
- CCA (Clear Channel Assessment): Đánh giá kênh sạch trước khi bắt đầu truyền dữ liệu để tránh va chạm kênh. Node sẽ kiểm tra xem kênh hiện tại có rõ hay không trước khi gửi dữ liệu.
- Tx (Transmission): Truyền gói tin từ node gửi (transmitter) tới node nhận (receiver).
- Rx (Reception): Nhận dữ liệu từ node khác (thường là sau khi một node gửi gói tin).
- ACK (Acknowledgment): Nếu cần thiết, sau khi nhận thành công gói tin, node nhận sẽ gửi gói ACK để xác nhận việc truyền thành công.

Có hai loại liên kết giữa các node trong TSCH:

- Dedicated Link: Một node truyền đến một hoặc nhiều node nhận.
- Shared Link: Nhiều node truyền đến cùng một node.



Hình 4: Hai loại liên kết giữa các node trong TSCH

Ở Hình 4 có đề cập đến từ khóa ‘Channel Offset’ là một khái niệm để chỉ việc các node nhảy qua các kênh tần số khác nhau trong cùng một Time-slot. Channel offset giúp xác định kênh tần số mà mỗi node sẽ sử dụng để truyền và nhận dữ liệu trong một timeslot cụ thể.

Công thức tính toán tần số thực tế:

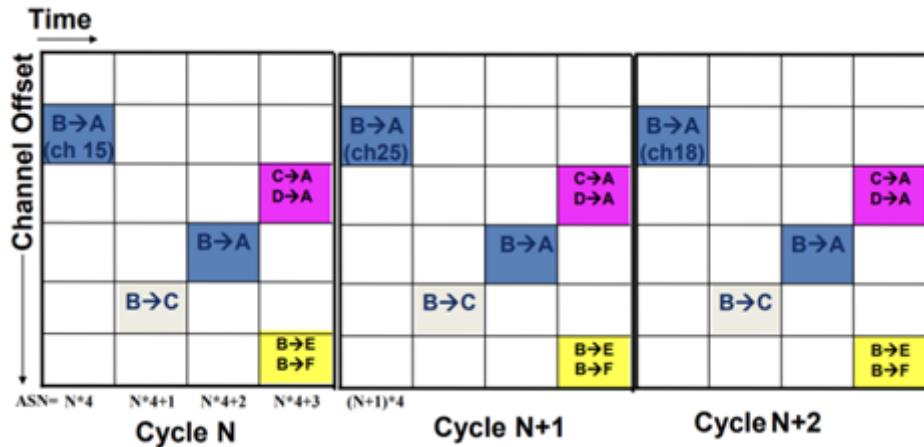
Channel offset:

$$f = F((ASN + chOff) \mod n_{ch})$$

với

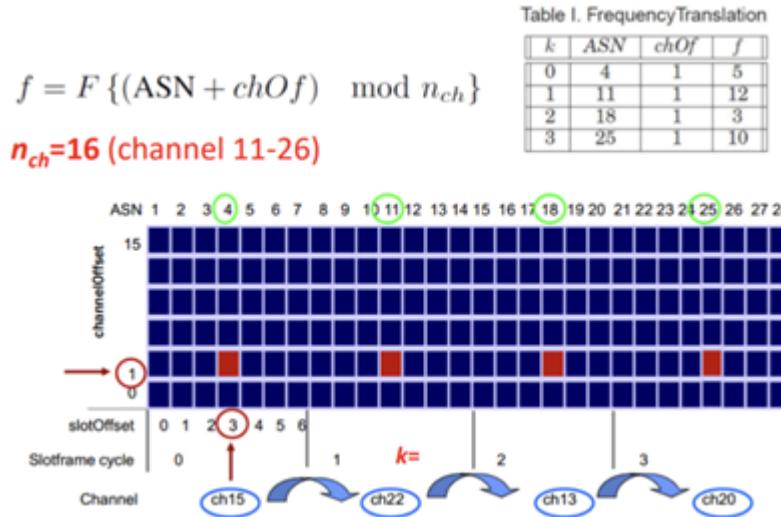
$$ASN = (k \times S + t)$$

- $S$  là slotframe size.
- $k$  là slotframe cycle.
- $t$  là thứ tự của slotframe trong một time
- $n_{ch}$  là tổng số channel sử dụng. 16 channel từ channel 11 đến channel 28.
- $chOf$  là channel offset (do người dùng define).
- $F$  là look-up table tra cứu chứa các bộ kênh có sẵn.



Hình 5: Giản đồ timeslot của một hệ thống 4 node sử dụng giao thức TSCH

Ví dụ 1: mạng TSCH có slotframe gồm 7 timeslot  $S = 7$ , một node được xếp vào timeslot thứ 3 của slotframe ( $t = 3$ ), tổng số channel của mạng là 16 kênh ( $n_{ch} = 16$ ), channel offset được set là 1 ( $chOff = 1$ ).



Hình 6: Minh họa của ví dụ 1

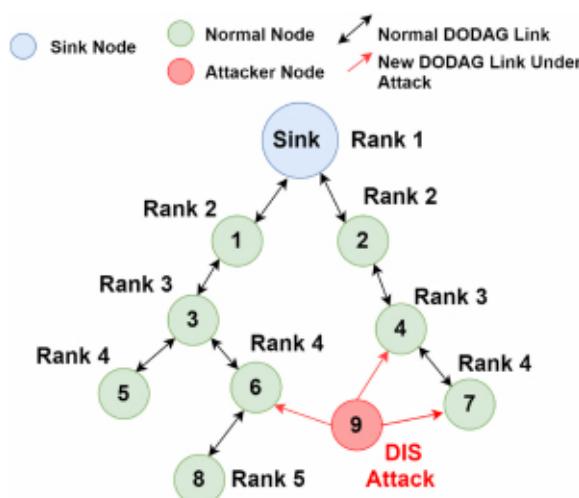
Như vậy, mỗi timeslot sẽ sử dụng một kênh khác nhau, đảm bảo tính phân tán và tránh xung đột.

### 2.3. Routing Protocol for Lossy and Low-power network – RPL

RPL – Routing Protocol for Lossy and Low-power Network là giao thức định tuyến sử dụng IPv6 dành cho Lossy and Low-power networks, được triển khai trong network Ipv6 stack của Contiki-NG. RPL duy trì và xây dựng cấu trúc Destination-Oriented Directed Acyclic Graph (DODAG) topology bắt nguồn từ root node.

Mục tiêu của RPL là cung cấp một định tuyến tối ưu nhất trong một mạng DODAG

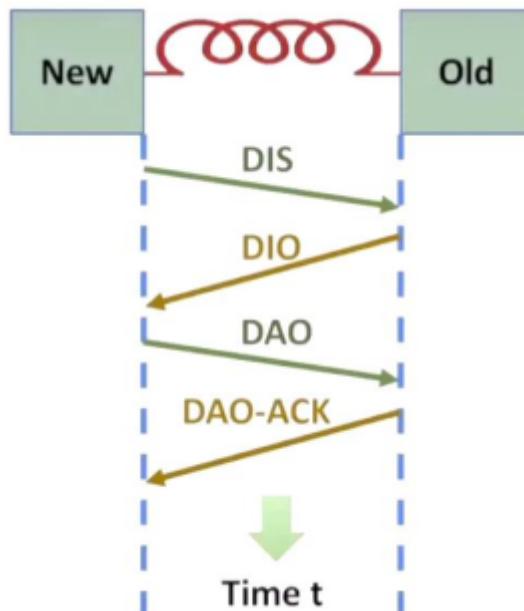
- Directed Acyclic Graph (DAG): là sơ đồ không tồn tại bất cứ vòng lặp nào, giống với mô hình Spanning Tree.
- Root: là điểm truyền tin cuối cùng của các node khác trong mạng, root node không có liên kết ra ngoài.
- Destination-Oriented Directed Acyclic Graph (DODAG): là một loại mô hình đặc biệt của DAG – tất cả các node đều gửi gói tin đến cùng một điểm đích.
- Rank: số lần nhảy (chuyển) gói tin để đến root node.
- DODAG ID: mỗi DODAG có một IPv6 dài 128-bit. ID này của root node và không bao giờ thay đổi.
- Parent node: node nhận dữ liệu từ child node và chuyển tiếp đến node tiếp theo.
- Child node: node gửi dữ liệu đến root node cần phải thông qua parent node.



Hình 7: Mô hình mạng của giao thức RPL

### RPL Control Message:

- DIS – DODAG Information Solicitation: gói tin DIS được gửi bởi một node mới muốn tham gia vào mạng DODAG để kiểm tra xem có bất kỳ mạng DODAG nào đang ở trong tầm hoạt động hay không.
- DIO – DODAG Information Object: gói tin DIO là multi-casted downwards, được gửi bởi các node đã tham gia mạng DODAG đến các node đã gửi DIS để thông báo có mạng DODAG đang ở trong tầm hoạt động của node đó.
- DAO – DODAG Advertisement Object: đây là một request được gửi từ child node đến root node với mục đích thông báo yêu cầu được tham gia vào mạng DODAG.
- DAO – ACK: đây là response được gửi từ root node hoặc một parent node đến child node, có thể là YES hoặc NO.



Hình 8: Giải đồ quảng bá gói tin để thành lập mạng DODAG của giao thức RPL

Các chế độ hoạt động của RPL trong Contiki NG khác nhau ở việc định tuyến downward.

### **Storing mode:**

- Bảng định tuyến sẽ được lưu ở mỗi Node, giảm dung lượng của IPv6 header khi truyền đi nhưng đòi hỏi bộ nhớ lớn ở mỗi node.

Định nghĩa định tuyến Storing mode trong chương trình:

```
MAKE_ROUTING = MAKE_ROUTING_RPL_CLASSIC
```

### **Non-storing mode:**

- Routing table không được sử dụng để định tuyến, thay vào đó header IPv6 sẽ tăng lên nếu gửi upward. Gửi từ root node xuống sẽ không bị ảnh hưởng.

```
#define RPL_CONF_MOP RPL_MOP_NON_STORING
```

### **Mode of operation 0 (MOP0):**

- MOP0 sẽ vô hiệu hóa downward routing.

```
#define RPL_CONF_WITH_STORING 0
#define RPL_CONF_WITH_NON_STORING 0
#define RPL_CONF_MOP RPL_MOP_NO_DOWNWARD_ROUTES
```

## **2.4. Giao thức 6top**

6P (6top Protocol) là giao thức giúp các node mạng trong mạng 6TiSCH giao tiếp với nhau để quản lý việc phân bổ hay hủy phân bổ các ô truyền dữ liệu (cells) giữa hai node lân cận. Giao thức này cho phép các nút phân bổ tài nguyên theo cách phân tán, mà không cần phụ thuộc vào một node tập trung.

6P định nghĩa một số lệnh để quản lý các ô giữa hai nút trong cùng phạm vi phủ sóng vô tuyến:

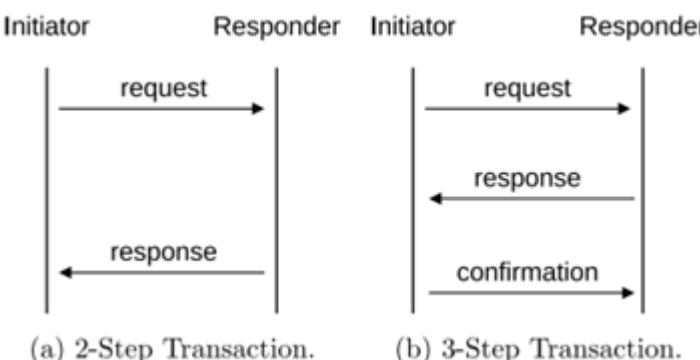
### Các lệnh được định nghĩa trong 6P:

- ADD: Thêm các ô vào lịch trình truyền dữ liệu với một nút.
- DELETE: Hủy các ô đã lên lịch với một nút.
- RELOCATE: Di chuyển các ô từ vị trí này sang vị trí khác trong khung thời gian (slotframe), tương tự như kết hợp giữa ADD và DELETE.
- CLEAR: Hủy toàn bộ các ô đã được lên lịch với một nút.
- COUNT và LIST: Xác nhận xem hai nút có cùng lịch trình truyền dữ liệu hay không.
- SIGNAL: Định nghĩa các lệnh tùy chỉnh nếu cần.

### Mô hình hoạt động của 6P trong mạng

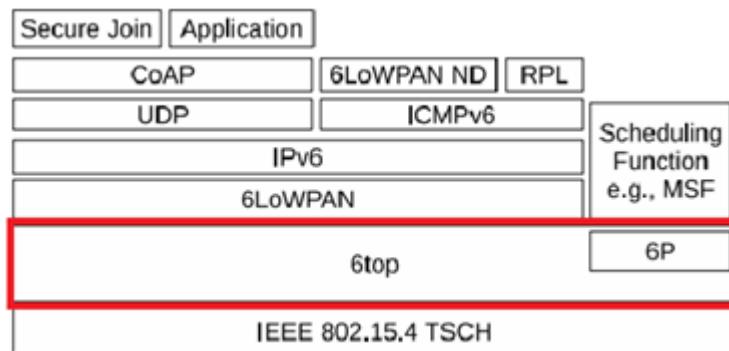
6P hoạt động dựa trên mô hình tin nhắn yêu cầu-phản hồi, thường diễn ra theo hai bước hoặc ba bước:

- Mô hình chuyển đổi 2 bước(2-Step Transaction): Một nút gửi yêu cầu, nút kia phản hồi để hoàn tất. Ví dụ, với lệnh ADD, nút gửi yêu cầu có thể chỉ định các ô để thêm, và nút nhận quyết định chấp nhận hoặc từ chối.
- Mô hình chuyển đổi 3 bước(3-Step Transaction): Khi yêu cầu ban đầu không có các ô cụ thể, nút phản hồi sẽ cung cấp một danh sách các ô để nút khởi tạo chọn lọc và xác nhận lại.



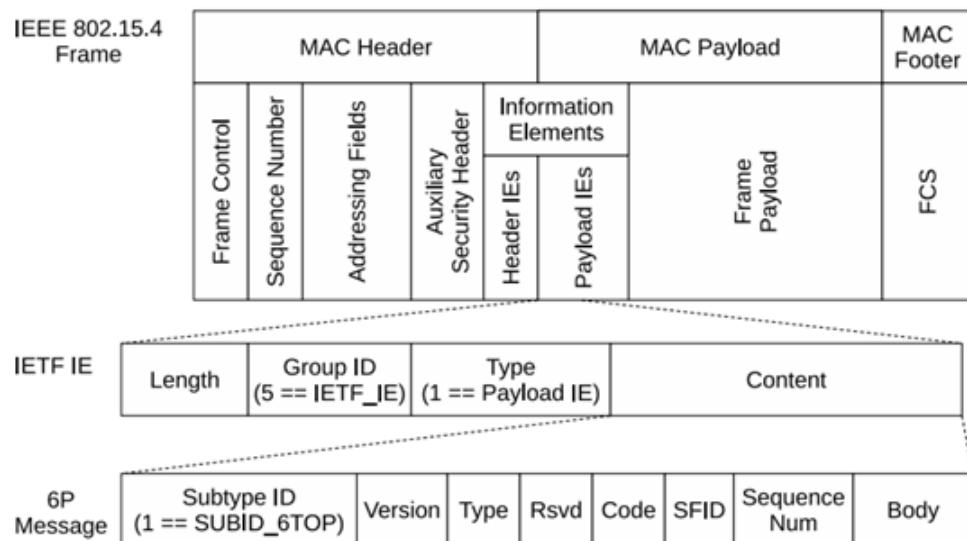
Hình 9: 6P Transaction Patterns

Trong một mô hình mạng, 6top được đặt trên lớp vật lý như hình 9



Hình 10: Vị trí của 6P trong mạng

Thông điệp của 6P được truyền dưới dạng một "Payload Information Element" trong một khung IEEE 802.15.4, và có định dạng nhất định để truyền tải các thông tin



Hình 11: Mô hình khung dữ liệu thông tin truyền của 6P

## 2.5. Giao thức UDP

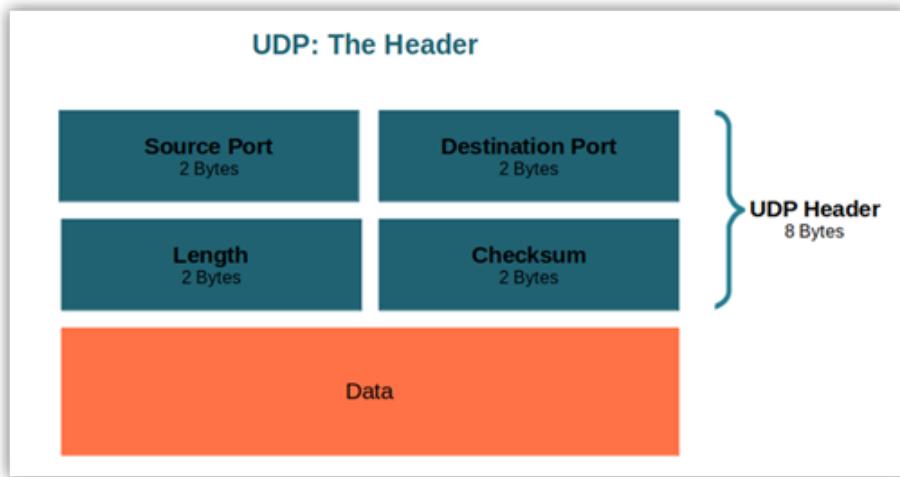
UDP (User Datagram Protocol) là một giao thức thuộc tầng Transport (vận chuyển) trong mô hình TCP/IP. Khác với TCP (Transmission Control Protocol), UDP là giao thức không kết nối (connectionless) và không đảm bảo độ tin cậy. Các đặc điểm chính của UDP bao gồm: **Các đặc điểm nổi bật của giao thức UDP:**

- Không kết nối (Connectionless): Không cần thiết lập kết nối trước khi gửi dữ liệu.
- Không đảm bảo độ tin cậy: Không kiểm tra xem dữ liệu có đến đích hay không, không có cơ chế hồi đáp hoặc xác nhận (ACK).
- Không sắp xếp thứ tự: Các gói tin (datagrams) có thể đến nơi theo thứ tự khác nhau hoặc bị mất mà không có cơ chế phục hồi.
- Tốc độ cao: Vì không có các bước kiểm tra, thiết lập hoặc xác nhận, UDP rất nhanh và phù hợp cho các ứng dụng yêu cầu tốc độ hơn độ tin cậy.

**Cấu trúc header của một gói tin UDP đơn giản hơn TCP với 4 trường cơ bản sau:**

- Source Port (16 bits): Cổng nguồn.
- Destination Port (16 bits): Cổng đích.
- Length (16 bits): Độ dài toàn bộ gói UDP (header + dữ liệu).
- Checksum (16 bits): Giá trị kiểm tra lỗi của gói tin (tùy chọn).

**Phần payload (dữ liệu chính) nằm ngay sau phần header này được phân bố như hình 12.**



Hình 12: Datagram của giao thức UDP

Trong Contiki NG hỗ trợ API của UDP là simple-udp.

```
[frame=single] % Tùy chọn khung bao quanh
static struct simple_udp_connection udp_conn;
simple_udp_register(&udp_conn, UDP_PORT, NULL,
                    UDP_PORT, udp_rx_callback);

/*---Datagram-receive call-back function---*/
static void udp_rx_callback(struct simple_udp_connection *c,
                            const uip_ipaddr_t *sender_addr,
                            uint16_t sender_port,
                            const uip_ipaddr_t *receiver_addr,
                            uint16_t receiver_port,
                            const uint8_t *data,
                            uint16_t datalen){...}

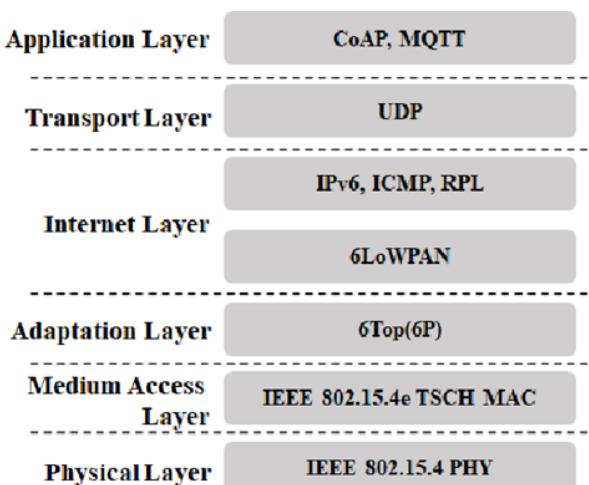
/*---Send UDP datagram---*/
uint8_t payload[64] = { 0 };
simple_udp_sendto(&udp_conn, payload, 2, &destination_ipaddr);
```

## 2.6. 6TiSCH

Đối với TSCH chỉ nêu cách đồng bộ hóa giữa các node thông qua EB, cơ chế cấp timeslot và quản lí slotframe, vì vậy việc lập lịch schedule và duy trì lịch trình liên lạc trong mạng TSCH nằm ngoài các tiêu chuẩn đã thiết lập cho TSCH.

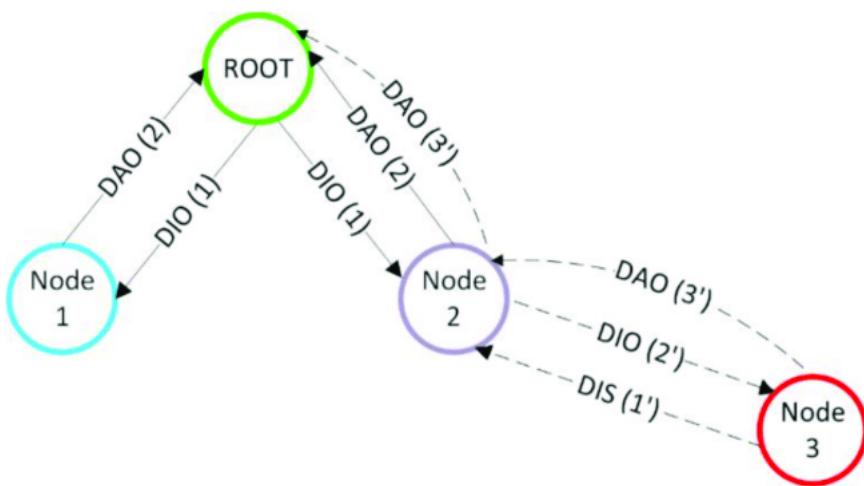
Vào năm 2013, IETF Working Group được thành lập để kích hoạt IPv6 trên TSCH. Trong đó bao gồm các chuẩn kiến trúc cơ bản, giao diện, bộ lập lịch, và bảo mật. 6TiSCH cho phép quản lí node và lịch biểu thông qua CoAP, xây dựng trên COMI (CoAP Management Interface). Nó cũng định nghĩa 6top, một lớp con cho phép cài đặt và loại bỏ các slot của các node lân cận với nhau, vận hành các chức năng lập lịch, xác định các quy tắc về thời điểm cũng như thêm và xóa vị trí timeslot tại mỗi node.

6TiSCH xác định cách TSCH tương tác với các lớp trên và giao thức định tuyến RPL, đảm bảo tính ánh xạ nhất quán của RPL routing topology và TSCH time-source graph.



Hình 13: Netstack của một Node sử dụng 6TiSCH

Kết hợp hai lớp trên và dưới với nhau bằng giao thức 6top do IETF 6TiSCH đề xuất:



**Hình 14:** Topology của một mạng sử dụng 6TiSCH

Trước khi 6TiSCH topology được hình thành, cần thực hiện hai quá trình:

- TSCH synchronization:
  - Mục đích: xây dựng một mạng lưới đồng bộ
  - Cách thực hiện: trao đổi các EB frame chứa IE
- - RPL DODAG construction
  - Mục đích: xây dựng hệ thống các node theo mô hình DODAG
  - Cách thực hiện: trao đổi các DODAG Infomation Object Packets

Để kết hợp được hai quá trình này, ta sử dụng IETF 6TiSCH minimal configuration:

- Lập lịch cho Root node share-timeslot và sink gửi các gói tin EBs và DIOs.
- Node muốn tham gia mạng sẽ giữ Radio On và lắng nghe gói tin EB.
- Sau khi node bắt được EB: node sẽ learn minimal schedule và được đồng bộ hóa.
- Sau khi node bắt được DIO: node sẽ chọn parent node và broadcast EBs, DIOs như root node.

Cuối cùng, tất cả các node đều được đồng bộ TSCH và hình thành được DODAG topology.

## PHẦN III: PHÂN TÍCH CÁC VÍ DỤ 6TiSCH

### 3.1. Ví dụ Simple Node

Đây là ví dụ về các nút sử dụng TSCH kết hợp RPL trong mạng. Minh họa cách các node được thiết lập để tham gia mạng, cách cấu hình node làm root, và in bảng định tuyến theo chu kỳ.

**Các chế độ hoạt động:** Theo mặc định, ứng dụng sẽ hoạt động trong vai trò của một nút RPL+TSCH thông thường. Thay đổi giá trị của biến `is_coordinator` khi khởi động để đặt nó ở chế độ điều phối TSCH.

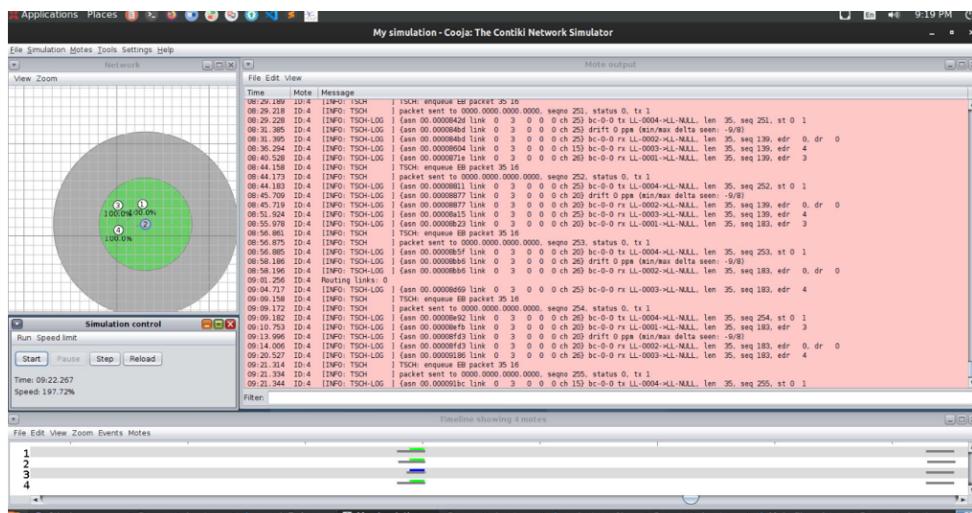
Nền tảng Cooja mote và Zolertia Z1 thường được sử dụng để mô phỏng. Chế độ điều phối được tự động bật nếu ID nút bằng 1.

- Mặc định ban đầu sẽ hoạt động như một node bình thường
- Khi khởi tạo biến `is_coordinator` thì thiết lập node được đặt làm root của DAG.

Thực hiện chạy mô phỏng với Root là Node ID 2. Để in định tuyến, cần cấu hình file Makefile như sau:

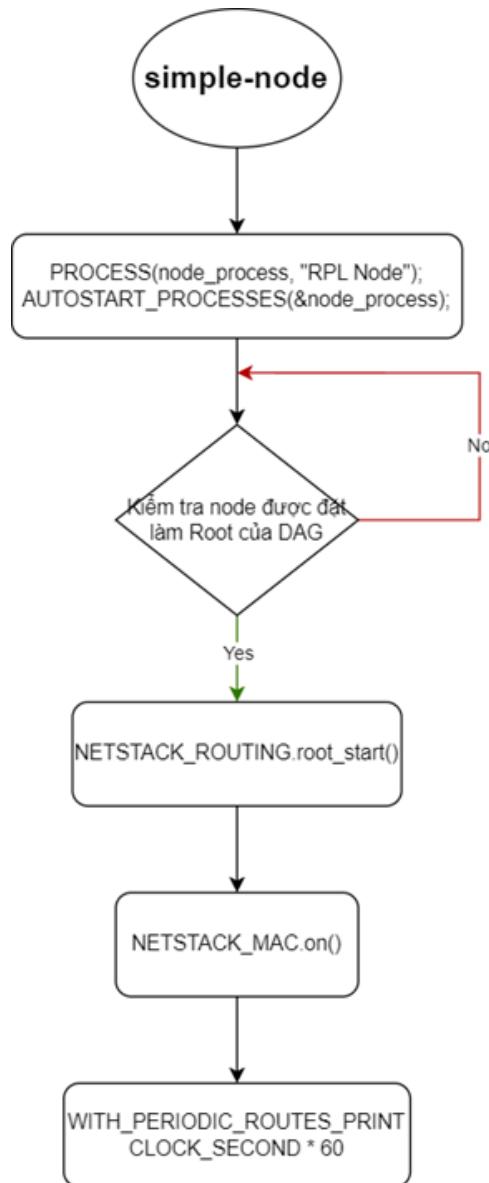
```
MAKE_WITH_PERIODIC_ROUTES_PRINT = 1
```

Dưới đây là kết quả mô phỏng trên Cooja sử dụng Contiki-NG



Hình 15: Kết quả mô phỏng ví dụ Simple-node

Nhận xét: Kết quả chạy ta xem các Log của TSCH hiển thị phân chia các kênh, độ dài và tần suất . Ở phần timeline, ta thấy được các lần truyền dữ liệu của các node đều nhau. Khi ngắt kết nối của một node trong mạng, timeline của node hiển thị liên tục, nghĩa là node luôn phát các gói DIS để tìm kiếm mạng xung quanh.



Hình 16: Lưu đồ giải thuật example Simple-node

### 3.2. Ví dụ ETSI-Plugtest-2017

Node hoạt động như một node bình thường trong mạng 6Tisch, có thể thay đổi node thành DAG Root bằng cách set lệnh rpl-set-root thông qua serial giao tiếp với phần cứng bằng shell.

Tính năng shell được thêm vào có sẵn khá nhiều command để giao tiếp với node như là kiểm tra trạng thái RPL, lịch TSCH, xem thông tin địa chỉ IPv6 của các node lân cận...

**RPL+TSCH:** Sử dụng RPL để định tuyến trong mạng với cấu trúc DAG, còn TSCH quản lý truyền thông tin.

**Serial shell:** Nhập lệnh và xem thông tin từ node thông qua serial Truy cập vào node thông qua Serial. Sau đó nhập >help để hiển thị các shell có sẵn.

```
> help

Available commands:

'> help': Shows this help

'> ip-addr': Shows all IPv6 addresses

'> ip-nbr': Shows all IPv6 neighbors

'> log module level': Sets log level (0--4) for a given module
(or "all"). For module "mac", level 4 also enables per-slot logg

'> ping addr': Pings the IPv6 address 'addr'

'> rpl-set-root 0/1 [prefix]': Sets node as root (on) or not (off).
A /64 prefix can be optionally specified.

'> rpl-status': Shows a summary of the current RPL state

'> rpl-global-repair': Triggers a RPL global repair

'> rpl-local-repair': Triggers a RPL local repair

'> routes': Shows the route entries

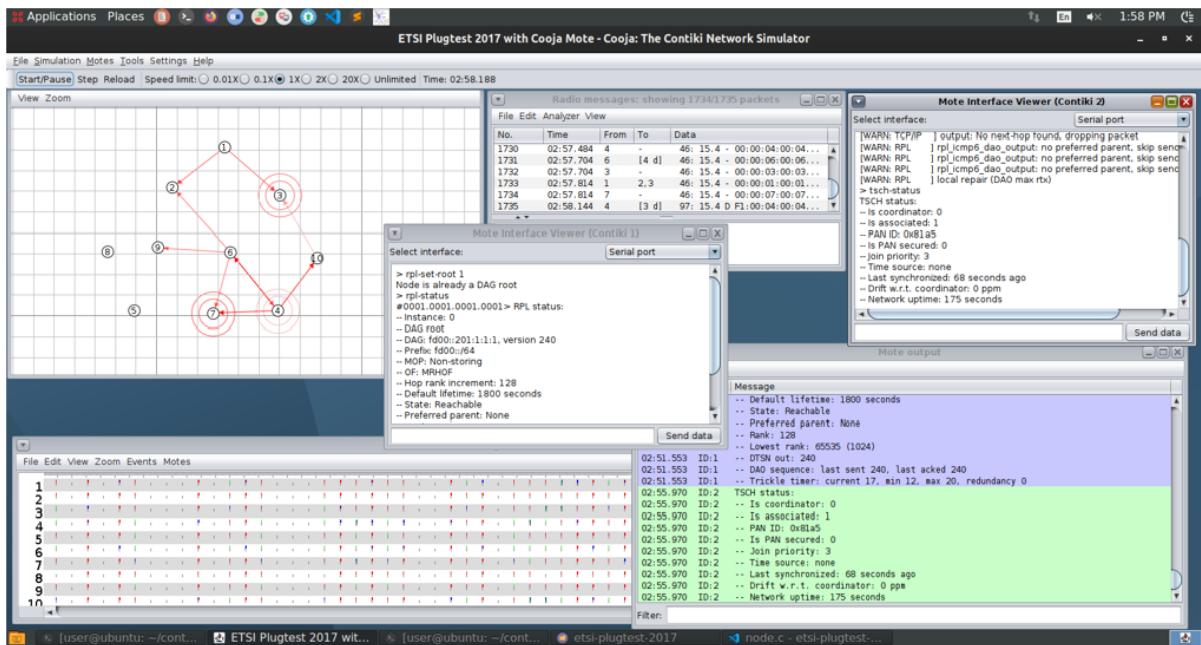
'> tsch-schedule': Shows the current TSCH schedule

'> tsch-status': Shows a summary of the current TSCH state

'> reboot': Reboot the board by watchdog_reboot()

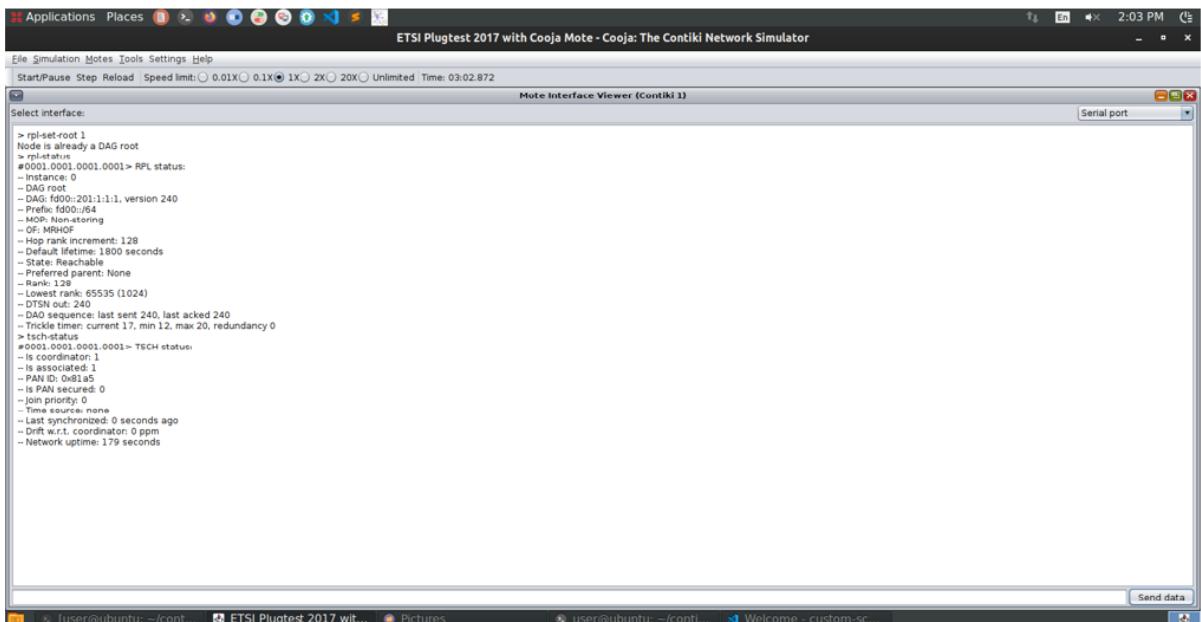
'> 6top help': Shows 6top command usage
```

Kết quả mô phỏng và thực hiện chạy các lệnh shell



Hình 17: Kết quả mô phỏng ETSI-Plugtest-2017

Chạy một số lệnh shell để kiểm tra định tuyến của mạng như rpl-status để kiểm tra định tuyến, tsch-status để kiểm tra trạng thái của tsch, rpl-set-root 1 để set node làm root.



Hình 18: Kết quả chạy shell trong mô phỏng ETSI-Plugtest-2017

### 3.3. Ví dụ 6top(6P)

Theo mặc định, chương trình sẽ hoạt động như một nút RPL+TSCH thông thường. Thay đổi giá trị của biến `is_coordinator` khi khởi động để đặt nó ở chế độ điều phối TSCH.

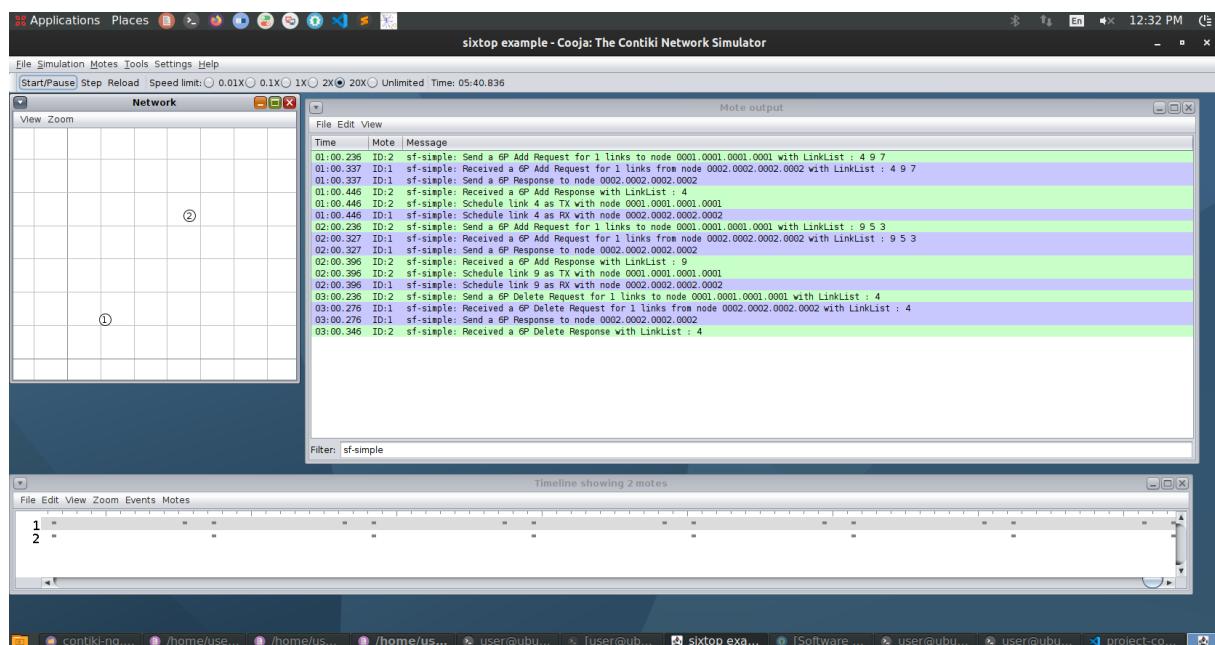
#### Hoạt động của 6P trong chương trình mô phỏng

Nếu không ở chế độ điều phối TSCH:

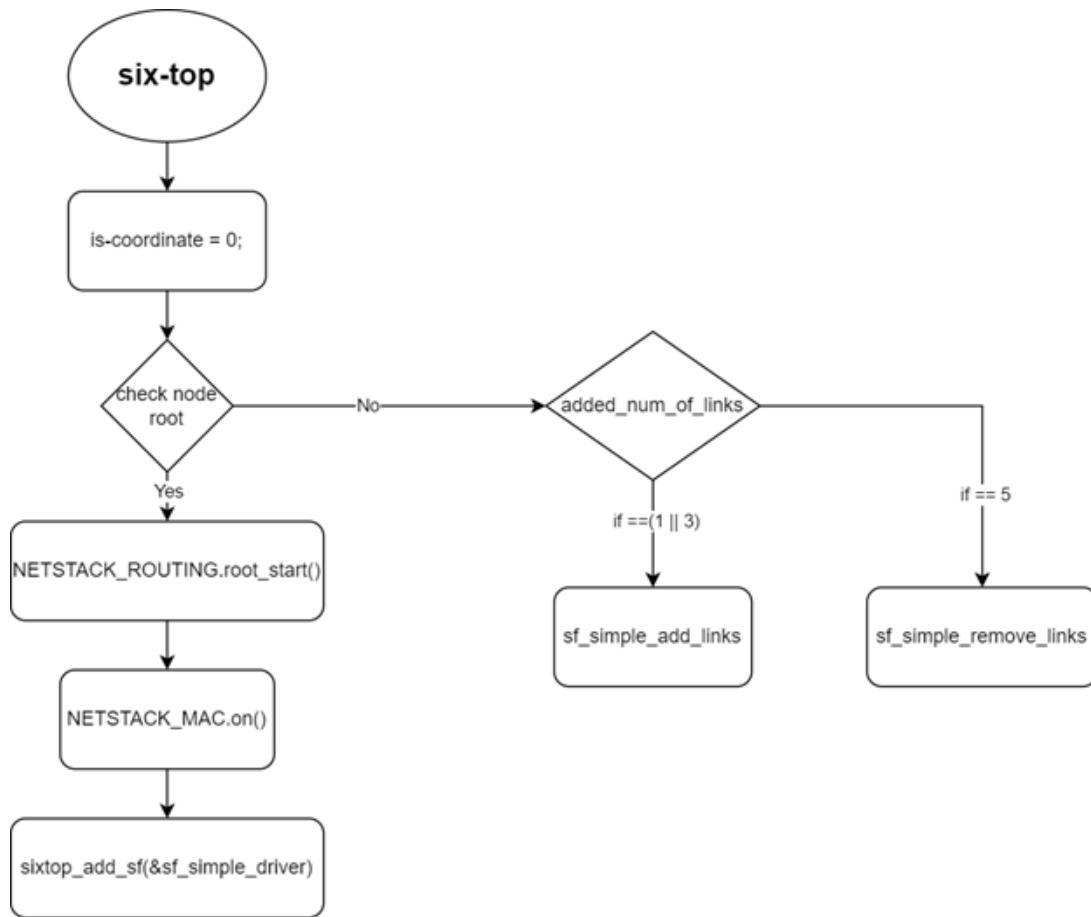
- Ứng dụng kích hoạt Yêu cầu thêm 6P vào 6dr
- Sau đó, ứng dụng kích hoạt một yêu cầu 6P khác Thêm vào 6dr
- Sau một khoảng thời gian, ứng dụng kích hoạt Yêu cầu xóa 6P tới 6dr

Khi chạy mô phỏng, cửa sổ "Mote output" của Cooja sẽ hiển thị các thông báo về quá trình trao đổi 6top, bao gồm:

- Thông báo nhận Sixtop IE.
- Thông báo gửi phản hồi liên kết đến nút khác.
- Thông báo sắp xếp liên kết cho các chế độ RX và TX với các nút khác.



Hình 19: Kết quả mô phỏng ví dụ 6top

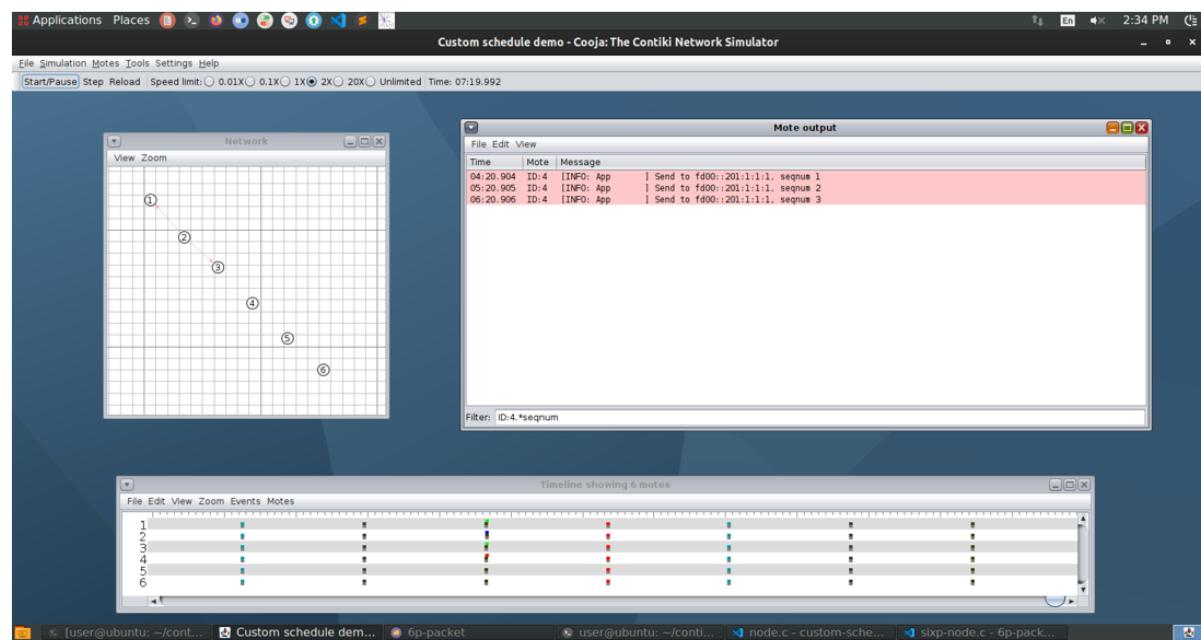
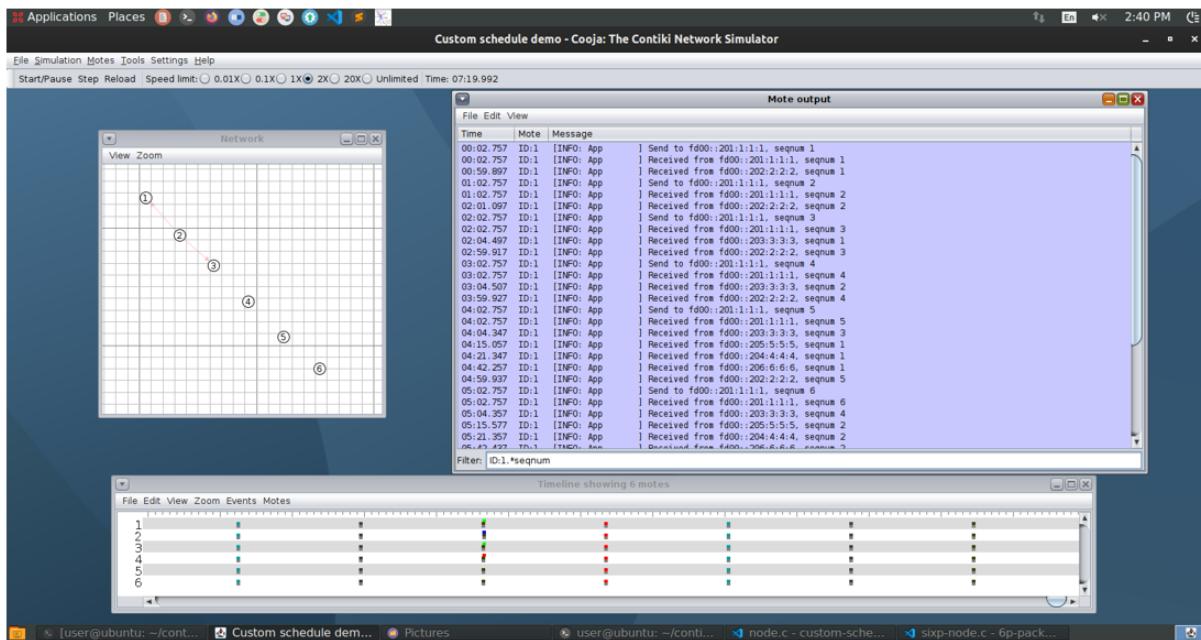


Hình 20: Lưu đồ giải thuật giải thuật 6top

### 3.4. Ví dụ Custom schedule

Ví dụ thực hiện xây dựng lịch trình riêng cho TSCH.

- Tạo một lịch TSCH đơn giản cho việc truyền thông giữa các nút, với các liên kết đã được cấu hình trước.
- Sử dụng UDP để gửi các gói tin chứa số thứ tự tới nút root.
- Hiển thị log thông tin về các gói tin đã nhận và đã gửi.



Hình 21: Kết quả mô phỏng ví dụ Custom schedule

### 3.5. Tạo ví dụ Hello-world chạy TSCH và RPL trên Contiki-NG

Thực hiện chỉnh sửa ví dụ Hello-world trên Contiki-NG Cho chương trình hello-world chạy TSCH bằng cách thêm vào Makefile

```
MAKE_MAC = MAKE_MAC_TSCH
```

Tiếp theo tạo file project-conf.h để bật TSCH và cấu hình làm Coordinator

```
#define TSCH_CONF_COORDINATOR 1
#define IEEE802154_CONF_PANID 0x1234 // PAN ID của mạng
#define TSCH_SCHEDULE_CONF_WITH_6TISCH_MINIMAL 1
```

Sửa đổi file hello-world.c để khởi động TSCH và thiết lập node làm Coordinator Kết quả các thông số của một node sau khi mô phỏng

```
00:01.106 ID:2 [INFO: TSCH      ] scanning on channel 15
00:01.109 ID:2 [INFO: TSCH      ] starting as node
00:01.112 ID:2 [INFO: Main     ] Starting Contiki-NG
00:01.115 ID:2 [INFO: Main     ] - Routing: RPL Lite
00:01.117 ID:2 [INFO: Main     ] - Net: sicslowpan
00:01.119 ID:2 [INFO: Main     ] - MAC: TSCH
00:01.123 ID:2 [INFO: Main     ] - 802.15.4 PANID: 0x1234
00:01.128 ID:2 [INFO: Main     ] - 802.15.4 TSCH default
hopping sequence length: 4
00:01.130 ID:2 [INFO: Main     ] Node ID: 2
00:01.135 ID:2 [INFO: Main     ] Link-layer address:
c10c.0000.0000.0002
00:01.138 ID:2 [INFO: RPL      ] initializing
00:01.145 ID:2 [INFO: Main     ] Tentative link-local
IPv6 address: fe80::c30c:0:0:2
00:01.148 ID:2 [INFO: Z1       ] CC2420 CCA threshold -45
00:01.149 ID:2 Hello, world
00:01.152 ID:2 [INFO: TSCH Sched] add_slotframe 0 7
```

```
00:01.160 ID:2 [INFO: TSCH Sched] add_link
sf=0 opt=Tx|Rx|Sh type=ADV ts=0 ch=0 addr=ffff.ffff.ffff.ffff
00:01.164 ID:2 [INFO: TSCH ] starting as coordinator,
PAN ID 1234, asn=0.0
```

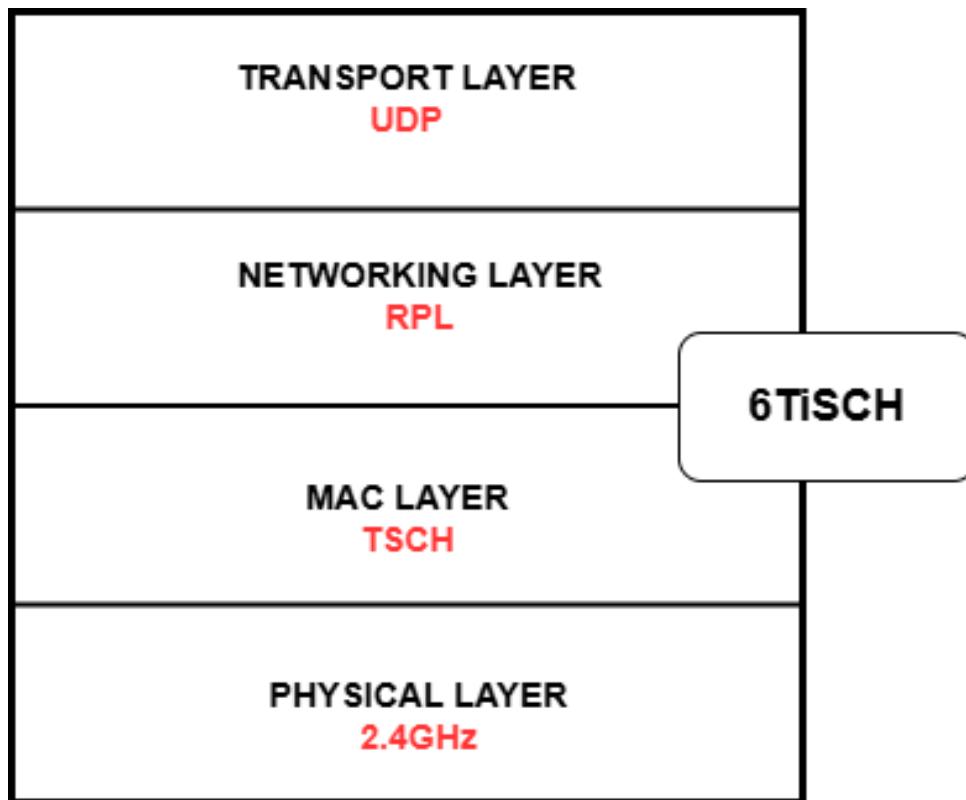
Nhận xét kết quả:

- Node khởi tạo, quét kênh 15 và bắt đầu hoạt động.
- Sử dụng RPL Lite để định tuyến, với nén gói tin IPv6 bằng 6lowPan.
- TSCH quản lý kênh và nhảy tần, với PAN ID là 0x1234 và độ dài chuỗi nhảy mặc định là 4.
- Địa chỉ MAC của node là c10c.0000.0000.0002 và địa chỉ IPv6 liên kết là fe80::c30c:0:0:2.

## PHẦN 4: Thực hiện thiết kế mô phỏng mạng 6TiSCH

### 4.1. Thiết kế lựa chọn Netstack cho project

Sau khi đã thực hiện tìm hiểu lý thuyết và chạy các ví dụ mô phỏng có sẵn trong thư mục 6TiSCH của Contiki-NG. Tiến hành thực hiện một project chạy các netstack đã triển khai. Netstack được sử dụng cho mỗi node cảm biến được mô tả như hình bên dưới



Hình 22: Netstack được sử dụng cho mỗi node cảm biến mạng không dây

- **Physical Layer:** Module CC2538 sử dụng và phát sóng vô tuyến 2.4GHz trên 16 channel từ kênh 11 đến kênh 26 với độ rộng mỗi kênh là 5MHz.
- **MAC Layer:** Sử dụng giao thức TSCH vì khả năng giảm nhiễu và độ tin cậy cao, phù hợp với các ứng dụng IoT có yêu cầu về tiết kiệm năng lượng và tính chính xác.
- **Networking Layer:** Sử dụng giao thức RPL phù hợp cho các ứng dụng mô hình mạng Root node – child node trong các mô hình như Smart home, ứng dụng nông nghiệp, ... Bên cạnh đó sự hỗ trợ Multihop làm cho mô hình mạng có khả năng bao quát được các khu vực rộng lớn hơn.

- **Transport Layer:** Sử dụng UDP để phù hợp truyền gói tin có kích thước nhỏ, không yêu cầu thiết lập kết nối, có tiêu đề nhỏ hơn, tiết kiệm năng lượng và băng thông, phù hợp với các ứng dụng IoT nơi không cần đảm bảo tính toàn vẹn dữ liệu chặc chẽ.
- **Scheduler:** Sử dụng 6TiSCH có hỗ trợ 6top giúp tối ưu hóa việc sử dụng tài nguyên thông qua việc đàm phán slot giữa các node với nhau.

## 4.2. Thực hiện project

### 4.2.1. Mô tả project

Project thực hiện một mạng 6TiSCH, bao gồm các Netstack đã mô tả ở phần 1. Trong đó bao gồm các node con đảm nhận việc đọc dữ liệu cảm biến và các thông số như TX, RX, tỉ lệ nhận gói tin data(PPR), cường độ tín hiệu (RSSI), của dữ liệu truyền từ node đến coordinator. Bên cạnh đó, coordinator và node còn thực hiện Ping dữ liệu thông qua UDP để đánh giá chất lượng đường truyền, cũng như tỉ lệ mất gói và độ trễ của mạng. Các dữ liệu sẽ được log hiển thị thông qua serial của root để có thể theo dõi.

### 4.2.2 Cấu hình file project-conf.h và Makefile

Định nghĩa một số thông số khởi tạo RPL

RPL\_CONF\_OF\_rpl\_mrhof: Định nghĩa cách RPL chọn đường dẫn có rank thấp nhất dựa trên một thông số tối ưu hóa (Metric), thường là ETX (Expected Transmission Count) hoặc năng lượng.

- MRHOF cố gắng tối thiểu hóa Rank (throughput metric).
- Mỗi node tính Rank của mình bằng cách cộng Rank của parent với metric của liên kết (link metric).
- Ví dụ với ETX (Expected Transmission Count)

$$\text{Rank node} = \text{Rank parent} + \text{ETX(link)}$$

- ETX đo lường số lần truyền cần thiết để gửi thành công một gói tin trên liên kết. Liên kết nào có ETX thấp hơn sẽ được ưu tiên.

- MRHOF sử dụng cơ chế hysteresis để tránh việc liên tục thay đổi parent nếu các giá trị metric (ETX, độ trễ) chỉ thay đổi nhỏ. Điều này giúp tăng độ ổn định của mạng.
- Một node chỉ thay đổi parent nếu Rank của parent mới tốt hơn một ngưỡng nhất định (ví dụ: giảm hơn 10

RPL\_CONF\_INIT\_LINK\_METRIC RPL\_INIT\_LINK\_METRIC\_ETX: Đặt giá trị khởi tạo cho chỉ số ETX, giúp mạng RPL khởi động nhanh hơn. ETX (Expected Transmission Count) là số lần truyền trung bình cần thiết để một gói tin được gửi thành công qua liên kết. Giá trị ETX thấp tương ứng với liên kết chất lượng cao (ít lỗi).

```
#define RPL_CONF_OF rpl_mrhof  
#define RPL_CONF_INIT_LINK_METRIC RPL_INIT_LINK_METRIC_ETX
```

Việc sử dụng MRHOF thay vì OF được so sánh thông qua các tiêu chí như sau:

Tiêu chí	MRHOF	OF0
Tiêu chí chọn parent	ETX (hoặc metric khác, ví dụ: năng lượng)	Số lượng hop
Độ phức tạp	Cao hơn (tính ETX, hysteresis)	Thấp (chỉ tính hop count)
Độ tin cậy liên kết	Cao (chọn liên kết tốt nhất)	Thấp (có thể chọn liên kết kém)
Ứng dụng mạng lớn	Tốt (quản lý chất lượng liên kết tốt hơn)	Hạn chế (số hop không đủ để đánh giá liên kết)
Ôn định (Hysteresis)	Có (giảm chuyển đổi parent liên tục)	Không (chuyển parent ngay khi Rank thay đổi)
Phù hợp thiết bị IoT nhỏ	Trung bình (yêu cầu tài nguyên tính toán)	Tốt (yêu cầu thấp, đơn giản hơn)

Bảng 1 So sánh giữa MRHOF và OF0

Kích hoạt Sixtop Protocol trong giao thức TSCH (Time-Slotted Channel Hopping). Để đảm bảo các node có thể đàm phán số lượng timeslot với các neighbor. Hữu ích để tối ưu hóa hiệu suất và giảm thiểu xung đột timeslot trong mạng.

```
#define TSCH_CONF_WITH_SIXTOP 1
```

Định danh duy nhất cho mạng TSCH bằng việc định nghĩa PAN ID (Personal Area Network ID). Node chỉ kết nối với mạng có cùng PAN ID, giúp ngăn ngừa xung đột giữa các mạng TSCH trong cùng khu vực.

```
#define IEEE802154_CONF_PANID 0xabcd
```

Việc cấu hình bật TSCH thủ công sẽ được kích hoạt khi gọi NETSTACK\_MAC.on() phù hợp để RPL khởi tạo trước sau đó mới khởi tạo TSCH

```
#define TSCH_CONF_AUTOSTART 0
```

Cấu hình thông số của giao thức TSCH trong file Project-conf.h

Cấu hình	Nội dung
TSCH_CONF_EB_PERIOD	Chu kỳ phát Enhanced Beacon (2 giây).
TSCH_CONF_DEFAULT_TIMESLOT_LENGTH	Độ dài timeslot mặc định (10 ms).
TSCH_SCHEDULE_CONF_DEFAULT_LENGTH	Số timeslot trong một slotframe (7 timeslot).
TSCH_CONF_MAX_FRAME_RETRIES	Số lần retry tối đa khi gửi gói tin (5 lần).
QUEUEBUF_CONF_NUM	Số lượng buffer hàng đợi cho gói tin (16 buffer).
UIP_CONF_BUFFER_SIZE	Kích thước bộ đệm cho gói tin IP (256 byte).

Bảng 2 Bảng cấu hình thông số TSCH

Cấu hình 6TiSCH Minimal để cài đặt các thông số khác mặc định

```
#define TSCH_SCHEDULE_CONF_WITH_6TISCH_MINIMAL 1
```

Cấu hình TSCH trong Makefile

```
MAKE_MAC = MAKE_MAC_TSCH
MODULES += $(CONTIKI_NG_MAC_DIR)/tsch/sixtop
```

Tiếp theo, để xem các Log chi tiết của từng Netstack, chúng ta có thể đặt mặc định định nghĩa các Log trong file Project-conf.h

Đầu tiên, định nghĩa Log định tuyến RPL để xem các thông tin về định tuyến giữa các node và coordinator bao gồm các thông báo điều khiển (DIO, DAO), thông tin về định tuyến, thay đổi parent, quá trình xây dựng cây DODAG.

Để xem các thông tin RPL ta có thể thêm các định nghĩa như sau

```
/* Enable logging */

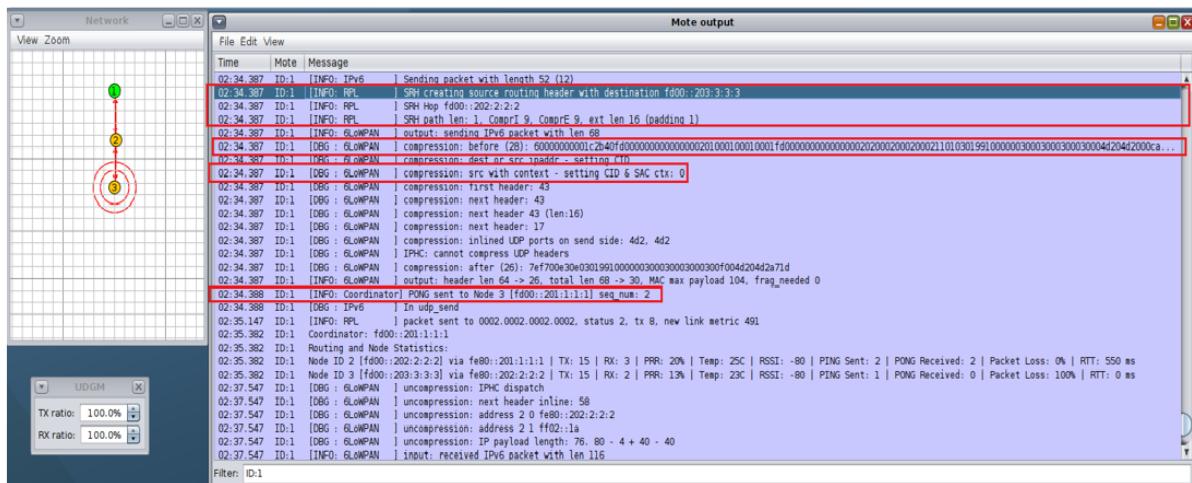
#define LOG_CONF_LEVEL_RPL LOG_LEVEL_DBG
#define LOG_CONF_LEVEL_IPV6 LOG_LEVEL_DBG
#define LOG_CONF_LEVEL_6LOWPAN LOG_LEVEL_DBG
#define LOG_CONF_LEVEL_MAC LOG_LEVEL_INFO
#define LOG_CONF_LEVEL_FRAMER LOG_LEVEL_ERR
```

Trong Contiki còn hỗ trợ việc tùy chỉnh mức Log để chúng ta có thể tập trung vào Log mà mình mong muốn khi hiển thị.

Mức log	Ý nghĩa
LOG_LEVEL_NONE	Không hiển thị log từ module.
LOG_LEVEL_ERR	Chỉ hiển thị các lỗi.
LOG_LEVEL_WARN	Hiển thị cảnh báo và lỗi.
LOG_LEVEL_INFO	Hiển thị thông tin cơ bản, cảnh báo, và lỗi.
LOG_LEVEL_DBG	Hiển thị đầy đủ thông tin gỡ lỗi, bao gồm cả DEBUG.

Bảng 3 Các mức log và ý nghĩa

Kết quả mô phỏng khi chạy thử log RPL



**Hình 23:** Kết quả chạy mô phỏng log RPL

- Giải thích một số thông số trong mô phỏng
    - Node ID:1 đang gửi một gói tin IPv6 với Header Routing Nguồn (SRH) đến địa chỉ đích fd00::203:3:3:3.
    - RPL SRH chứa danh sách các hop mà gói tin cần đi qua, bao gồm fd00::202:2:2:2 là hop đầu tiên.

```
[INFO: IPv6] Sending packet with length 52 (12)
[INFO: RPL] SRH creating source routing header with
destination fd00::203:3:3:3
[INFO: RPL] SRH Hop fd00::202:2:2:2
```

- Gói tin IPv6 được nén thông qua 6LoWPAN để giảm kích thước, tối ưu cho mạng không dây công suất thấp.

[DBG: 6LoWPAN] compression: IPHC: cannot compress UDP headers

- Gửi và nhận gói PONG

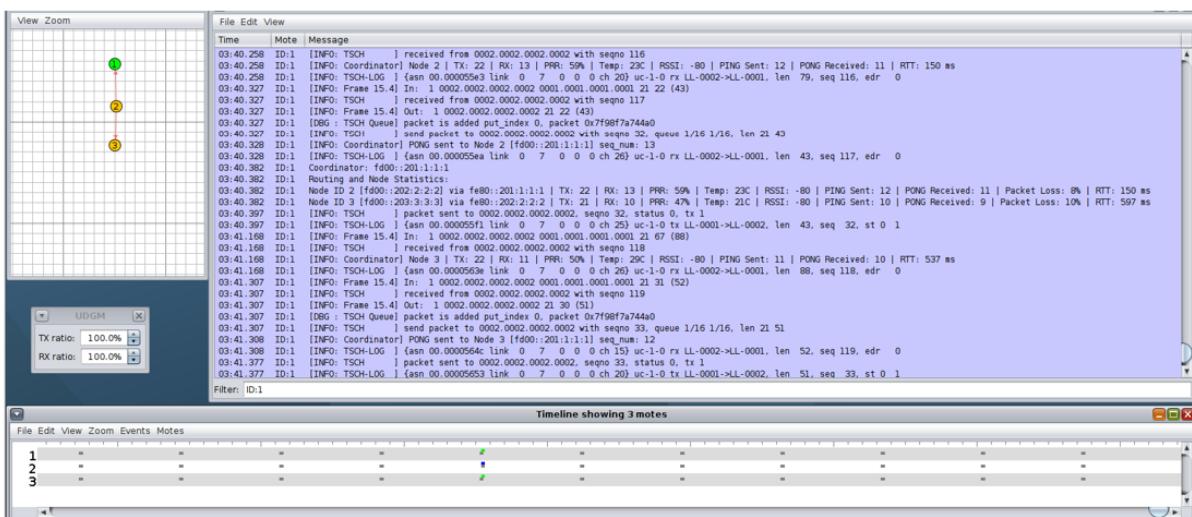
[INFO: IPv6] PONG sent to fd00::202:2:2:2, status 2,  
tx 8, new link metric 491

Tiếp theo, phân tích các log của TSCH để xem cách hoạt động trong mạng 6TiSCH.  
Định nghĩa các thông số bên dưới

```
/* Enable TSCH logging */

#define LOG_CONF_LEVEL_TSCH LOG_LEVEL_DBG
#define LOG_CONF_LEVEL_MAC LOG_LEVEL_DBG
#define LOG_CONF_LEVEL_FRAMER LOG_LEVEL_INFO
```

Kết quả mô phỏng như sau:



Hình 24: Kết quả chạy mô phỏng log TSCH

Ta có thể thấy chi tiết việc gửi dữ liệu , hàng đợi (queue) , các Enhance beacons thông qua

Mục	Nội dung
Log nhận gói tin từ các node	[INFO: TSCH] received from 0002.0002.0002.0002 with seqno 116 [INFO: TSCH] received from 0002.0002.0002.0002 with seqno 117
TSCH Queue	[INFO: TSCH] send packet to 0002.0002.0002.0002 with seqno 32, queue 1/16 [DBG: TSCH Queue] packet is added put_index 0, packet 0x7f98f7a74a40
Enhanced Beacon (EB)	[INFO: TSCH] TSCH: Sending EB in timeslot 1
Đồng bộ thời gian (Time Drift Correction)	[DBG: TSCH] TSCH: Drift correction - Adjusting clock by 5 ticks

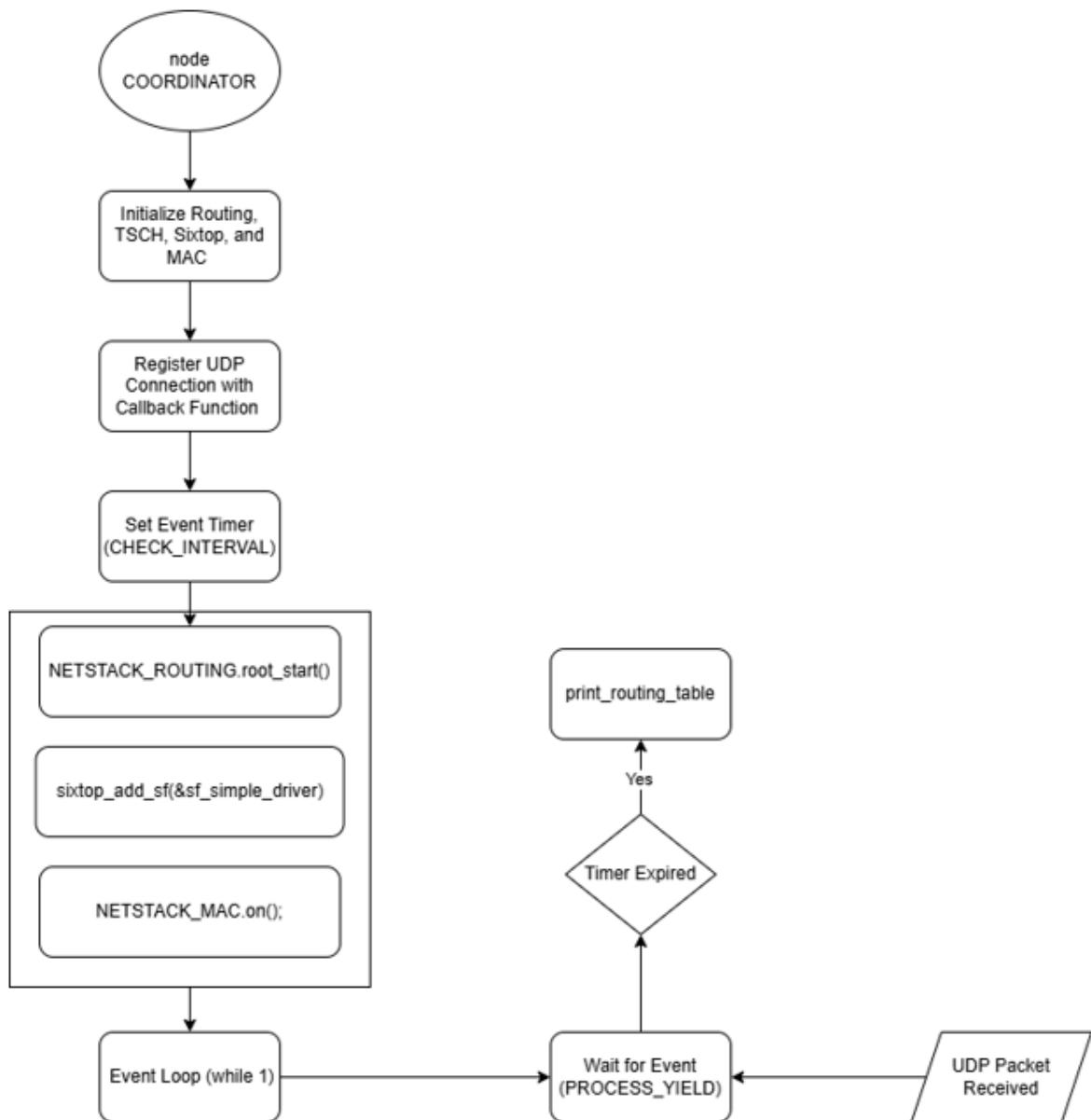
Bảng 4 Log TSCH và ý nghĩa mô tả

Để phân tích các log của 6top trong project, ta sẽ định nghĩa các thông số như sau

```
/* Enable 6top logging */
#define LOG_CONF_LEVEL_SIXTOP LOG_LEVEL_DBG
#define LOG_CONF_LEVEL_MAC LOG_LEVEL_DBG
```

#### 4.2.3. Lưu đồ giải thuật cho Coordinator

Coordinator thực hiện nhận gói tin gửi qua UDP thông qua hàm `udp_rx_callback()` và thực hiện phân tích gói tin để in ra bảng dữ liệu bao gồm Node ID, TX, RX, tỉ lệ PRR, RSSI, Ping, Pong, Packet loss, RTT và Nhiệt độ từ các node con.



Hình 25: Lưu đồ giải thuật Coordinator node

### **Coordinator\_process:**

Đây là process chính của nút điều phối và chịu trách nhiệm thực hiện như sau:

- **Khởi tạo:**

- In thông báo bắt đầu.
- Cấu hình nút làm gốc RPL (NETSTACK\_ROUTING.root\_start).
- Đăng ký chức năng điều phối của Sixtop (sixtop\_add\_sf).
- Bật lớp MAC (NETSTACK\_MAC.on).
- Đăng ký callback nhận gói UDP (simple\_udp\_register).

- **Chu kỳ kiểm tra:**

- Thiết lập bộ đếm thời gian (etimer).
- In bảng định tuyến và các dữ liệu nhận từ UDP định kì.
- Mỗi 5 giây, in bảng định tuyến (print\_routing\_table).

### Hàm *Udp\_rx\_callback*:

Hàm *udp\_rx\_callback* là hàm callback được gọi tự động mỗi khi nút nhận được một gói tin UDP. Nhiệm vụ của hàm này là xử lý dữ liệu nhận được từ các nút khác trong mạng, bao gồm việc phản hồi lại các yêu cầu PING và cập nhật các thông tin thống kê từ dữ liệu cảm biến nhận được. Được gọi khi nút điều phối nhận được gói tin UDP:

- Mục đích của hàm:

- Đối với gói tin PING/PONG: Cho phép nút điều phối phản hồi lại các yêu cầu PING từ các nút khác bằng cách gửi gói tin PONG, hỗ trợ việc đo lường thời gian trễ và kiểm tra kết nối.
- Đối với dữ liệu cảm biến: Thu thập và cập nhật các thông tin thống kê từ các nút trong mạng, bao gồm số lượng gói tin truyền nhận, nhiệt độ, RSSI, số lượng PING/PONG, thời gian trễ RTT, và lưu trữ chúng vào mảng *node\_stats* để sử dụng cho việc giám sát và quản lý mạng.

- Cập nhật thông tin nút:
  - Cập nhật tx\_count, tăng rx\_count.
  - Ghi địa chỉ nút và địa chỉ nút cha.
- Tính toán PRR với công thức PRR (Packet Reception Ratio) được định nghĩa như sau:

$$\text{PRR} = \frac{\text{rx\_count}}{\text{tx\_count}} \times 100\%$$

- Để lấy giá trị RSSI khi một gói tin được nhận từ các node khác, ta sử dụng hàm sau:

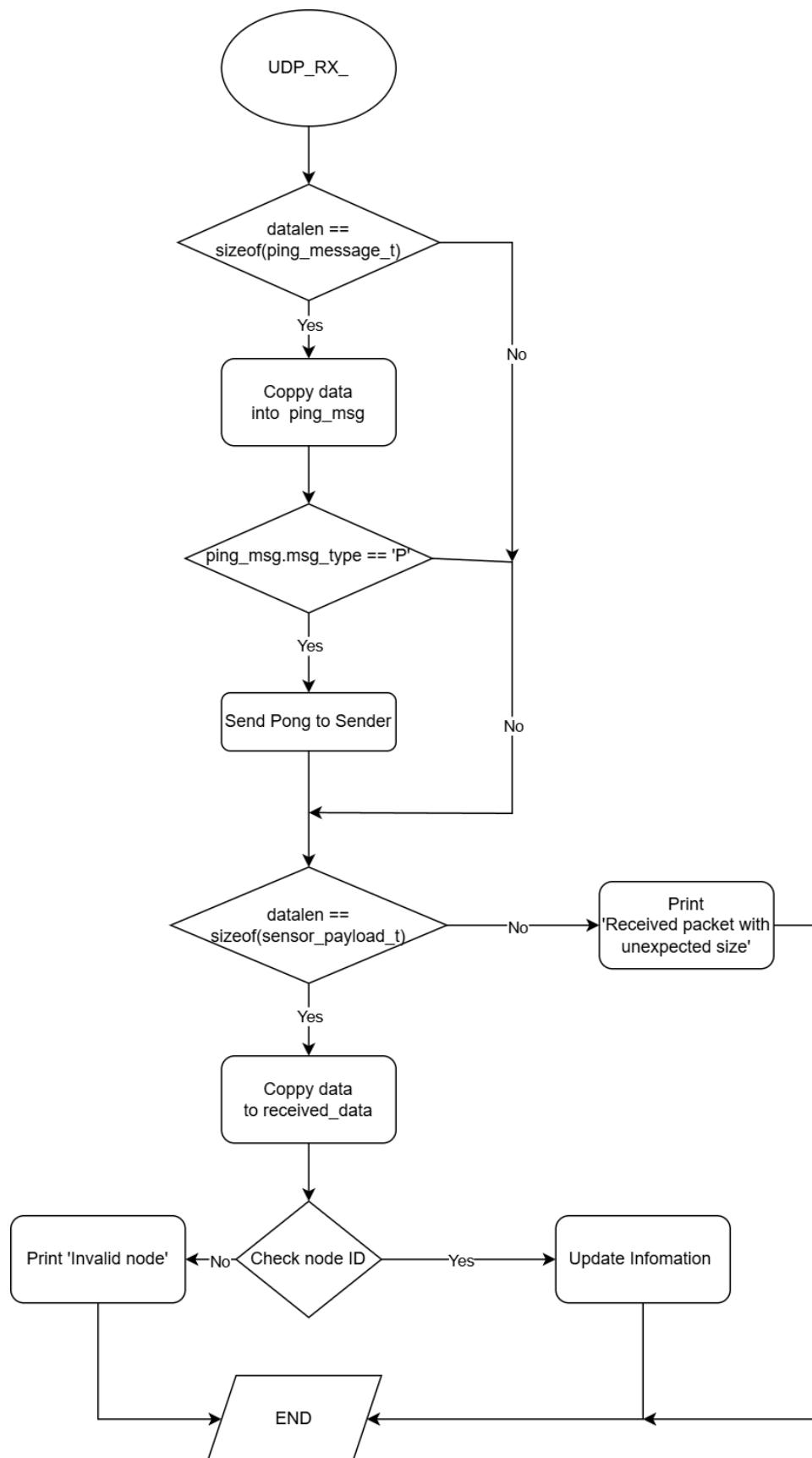
```
node_stats[index].rss = packetbuf_attr(PACKETBUF_ATTR_RSSI);
```

Hàm *print\_routing\_table*:

Hàm *print\_routing\_table* có nhiệm vụ in ra bảng định tuyến và các thông kê liên quan đến các nút trong mạng. Nó hiển thị thông tin về nút điều phối (coordinator), các nút con, và các số liệu thống kê như số lượng gói tin truyền nhận, tỷ lệ nhận gói tin (PRR), nhiệt độ, RSSI, số lượng PING/PONG, tỷ lệ mất gói tin (Packet Loss), và thời gian trễ vòng lặp (RTT).

- Mục đích của hàm:
  - Giám sát mạng: Cung cấp cái nhìn tổng quan về trạng thái của mạng và các nút thành viên, giúp người quản trị mạng có thể theo dõi hiệu suất và phát hiện các vấn đề kịp thời.
  - Phân tích hiệu suất: Thông qua các số liệu thống kê như PRR, Packet Loss, RSSI, và RTT, người dùng có thể đánh giá chất lượng kết nối giữa các nút, xác định nút nào có hiệu suất kém hoặc gặp sự cố.
  - Hỗ trợ trong việc tối ưu hóa mạng: Dựa trên các thông tin thu thập được, có thể thực hiện các biện pháp tối ưu hóa mạng, như điều chỉnh vị trí của các nút, thay đổi cấu hình mạng, hoặc nâng cấp phần cứng nếu cần thiết.

Dưới đây là lưu đồ giải thuật của hàm *udp\_rx\_callback()*



Hình 26: Lưu đồ hàm xử lý UDP packet received

#### **4.2.4. Lưu đồ giải thuật Node sensor**

##### **Sensor\_node\_process:**

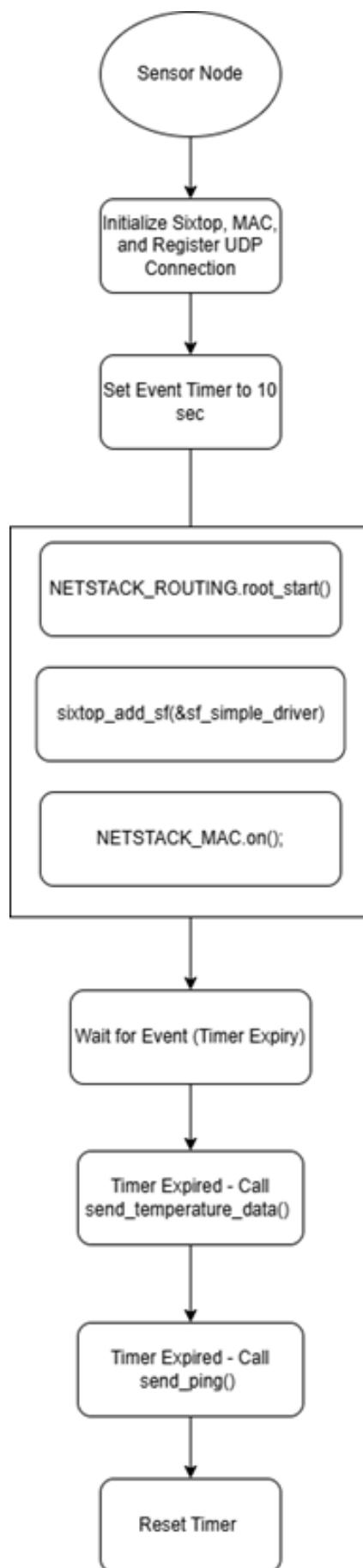
Process chính của nút cảm biến::

- **Khởi tạo:**

- Thêm chức năng lập lịch Sixtop (sixtop\_add\_sf).
- Bật lớp MAC TSCH (NETSTACK\_MAC.on).
- Đăng ký UDP (simple\_udp\_register).

- **Gửi dữ liệu định kỳ:**

- Sử dụng bộ đếm thời gian (etimer), gửi dữ liệu cứ mỗi 10 giây.
- Gọi hàm send\_temperature\_data để gửi dữ liệu đến nút điều phối.
- Gọi hàm send\_ping để Ping liên tục đến Coordinator, thực hiện đo tỉ lệ mất gói tin và Round Trip Time (RTT) độ trễ của gói tin



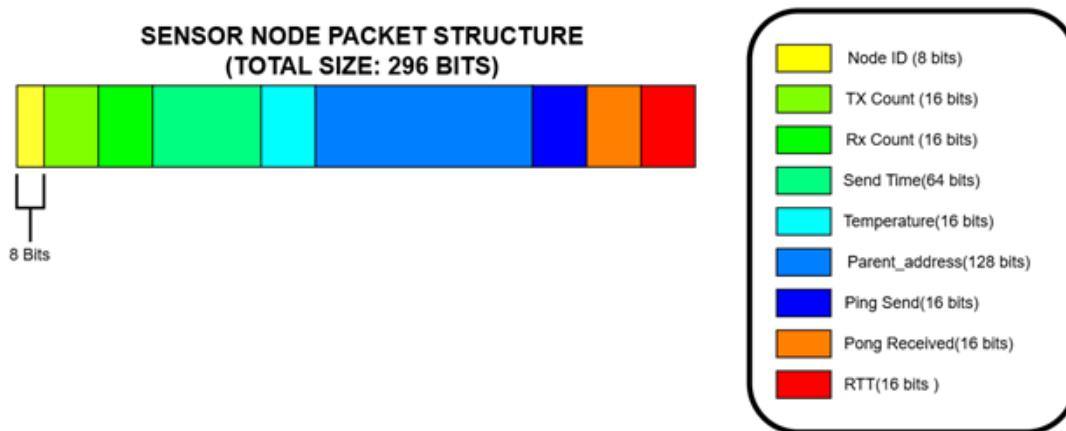
Hình 27: Lưu đồ giải thuật Node sensor



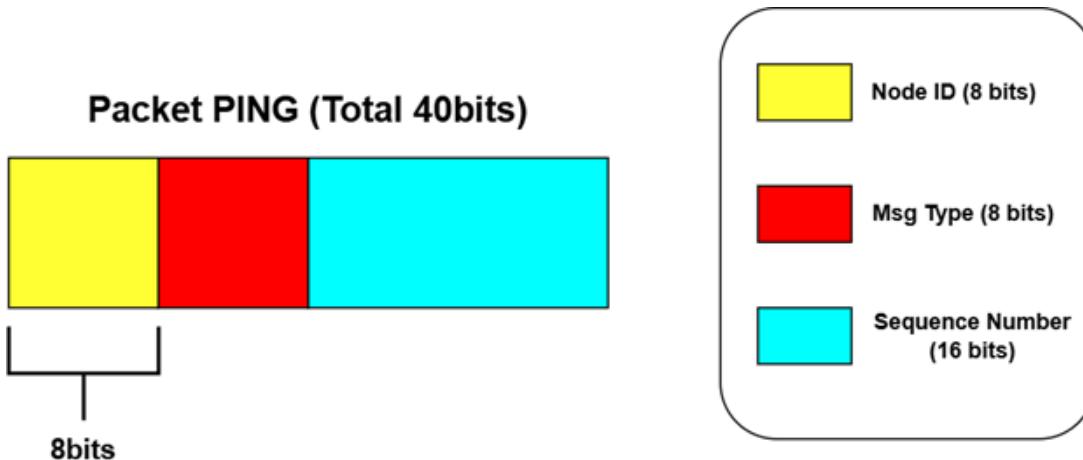
Hình 28: Lưu đồ giải thuật hàm gửi data và hàm ping

Hàm chịu trách nhiệm gửi packet dữ liệu là send\_temperature\_data(). Hàm này xây dựng gói tin (packet) chứa dữ liệu và gửi nó tới node coordinator bằng cách sử dụng hàm simple\_udp\_sendto()

Hàm này sử dụng cấu trúc sensor\_payload\_t để lưu dữ liệu cần gửi. Các trường của cấu trúc được điền đầy đủ với các thông tin như:



**Hình 29:** Cấu trúc gói tin gửi dữ liệu của Node sensor



**Hình 30:** Cấu trúc gói tin Ping của Node và Coordinator

Về lưu trữ định tuyến tại từng node

- Mỗi node duy trì bảng định tuyến cục bộ của mình để biết parent
- Node gửi thông tin parent, số lượng truyền/nhận, và các số liệu khác về Coordinator thông qua payload UDP.
- Coordinator sử dụng hàm NETSTACK\_ROUTING.get\_root\_ipaddr() để lấy địa chỉ IP root và in thông tin định tuyến.
- Node không truyền thông tin định tuyến trong header của gói tin. Thay vào đó, RPL xây dựng bảng định tuyến dựa trên các DIO (DODAG Information Object), DAO (Destination Advertisement Object), và DIS (DODAG Information Solicitation).

Một số hàm đặc trưng khác trong code được liệt kê bên dưới như sau:

Lấy địa chỉ IP của Coordinator:

```
NETSTACK_ROUTING.get_root_ipaddr(&dest_ipaddr);
```

Nếu không lấy được địa chỉ, hàm sẽ log lỗi:

```
LOG_ERR("Failed to get coordinator IP address.\r\n");
```

Gói tin được gửi đến địa chỉ IP của coordinator bằng cách sử dụng:

```
simple_udp_sendto(&udp_conn, &payload, sizeof(payload), &dest_ipaddr);
```

Trong đó:

**udp\_conn** : Kết nối UDP đã được khởi tạo.

**&payload** : Con trỏ tới gói tin đã xây dựng.

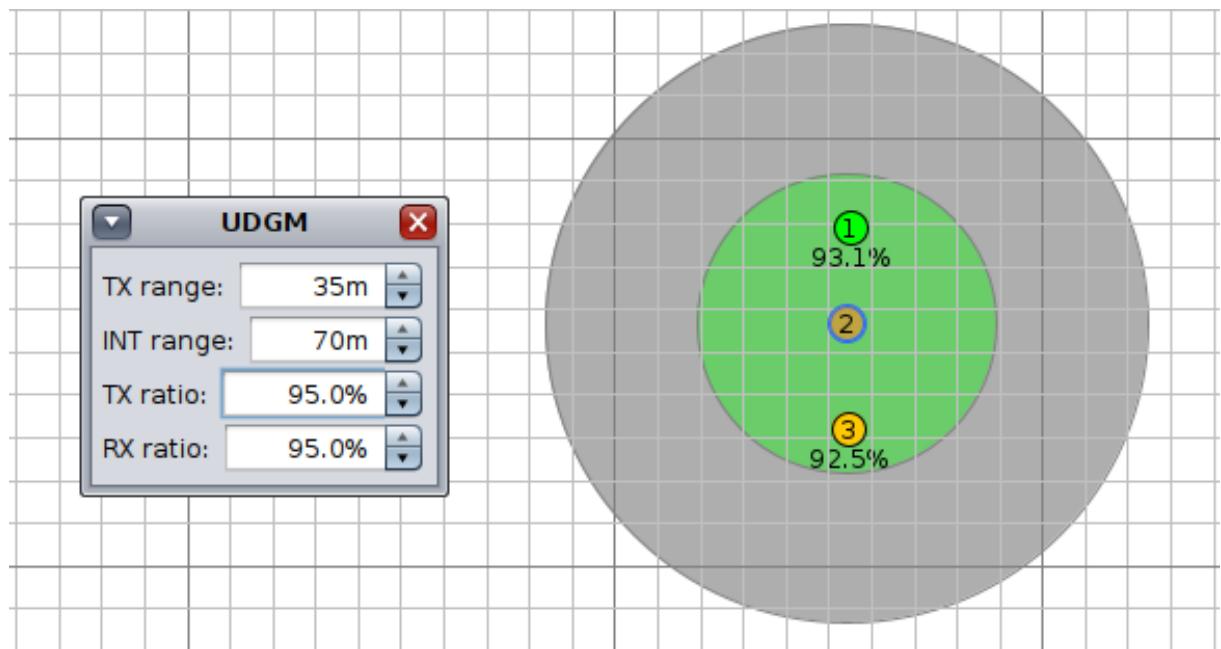
**sizeof(payload)** : Kích thước của gói tin.

**&dest\_ipaddr** : Địa chỉ IP của coordinator (đích đến).

### 4.3. Mô phỏng project trên cooja

Triển khai chương trình mô phỏng với Cooja, mạng mô phỏng sẽ được thực hiện bao gồm: 1 Coordinator và 2 Node sensor để tương ứng với chạy thực tế.

Thực hiện mô phỏng triển khai mạng chạy Multihop, Node 2 sẽ là parent của node 3 để gửi dữ liệu về Coordinator 1.



Hình 31: Mô hình mạng mô phỏng

Mô phỏng cài đặt thông số truyền trong 30m với tỉ lệ thu phát thành công thực tế đạt hơn 90% để có kết quả mô phỏng sát với thực tế khi chạy trên phần cứng hơn.

Dưới đây là kết quả mô phỏng.

2:53:20.382 ID:1 Coordinator: fd00::201:1:1:1
2:53:20.382 ID:1 Routing and Node Statistics:
2:53:20.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 1040   RX: 1031   PRR: 99%   Temp: 20C   RSSI: -63   PING Sent: 1030   PONG Received: 1029   Packet Loss: 0%   RTT: 180 ms
2:53:20.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 1039   RX: 1028   PRR: 99%   Temp: 20C   RSSI: -63   PING Sent: 1028   PONG Received: 1027   Packet Loss: 0%   RTT: 237 ms
2:53:21.058 ID:1 [INFO: Coordinator] Node 3   TX: 1040   RX: 1029   PRR: 99%   Temp: 24C   RSSI: -63   PING Sent: 1029   PONG Received: 1028   RTT: 777 ms
2:53:21.198 ID:1 [INFO: Coordinator] PONG sent to Node 3 [fd00::201:1:1:1] seq_num: 1030
2:53:25.382 ID:1 Coordinator: fd00::201:1:1:1
2:53:25.382 ID:1 Routing and Node Statistics:
2:53:25.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 1040   RX: 1031   PRR: 99%   Temp: 20C   RSSI: -63   PING Sent: 1030   PONG Received: 1029   Packet Loss: 0%   RTT: 180 ms
2:53:25.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 1040   RX: 1029   PRR: 99%   Temp: 24C   RSSI: -63   PING Sent: 1029   PONG Received: 1028   Packet Loss: 0%   RTT: 777 ms
2:53:30.298 ID:1 [INFO: Coordinator] Node 2   TX: 1041   RX: 1032   PRR: 99%   Temp: 25C   RSSI: -63   PING Sent: 1031   PONG Received: 1030   RTT: 190 ms
2:53:30.368 ID:1 [INFO: Coordinator] PONG sent to Node 2 [fd00::201:1:1:1] seq_num: 1032
2:53:30.382 ID:1 Coordinator: fd00::201:1:1:1
2:53:30.382 ID:1 Routing and Node Statistics:
2:53:30.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 1041   RX: 1032   PRR: 99%   Temp: 25C   RSSI: -63   PING Sent: 1031   PONG Received: 1030   Packet Loss: 0%   RTT: 190 ms
2:53:30.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 1040   RX: 1029   PRR: 99%   Temp: 24C   RSSI: -63   PING Sent: 1029   PONG Received: 1028   Packet Loss: 0%   RTT: 777 ms
2:53:31.498 ID:1 [INFO: Coordinator] Node 3   TX: 1041   RX: 1030   PRR: 99%   Temp: 23C   RSSI: -63   PING Sent: 1030   PONG Received: 1029   RTT: 577 ms
2:53:31.558 ID:1 [INFO: Coordinator] PONG sent to Node 3 [fd00::201:1:1:1] seq_num: 1031
2:53:35.382 ID:1 Coordinator: fd00::201:1:1:1
2:53:35.382 ID:1 Routing and Node Statistics:
2:53:35.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 1041   RX: 1032   PRR: 99%   Temp: 25C   RSSI: -63   PING Sent: 1031   PONG Received: 1030   Packet Loss: 0%   RTT: 190 ms
2:53:35.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 1041   RX: 1030   PRR: 99%   Temp: 23C   RSSI: -63   PING Sent: 1030   PONG Received: 1029   Packet Loss: 0%   RTT: 577 ms
2:53:35.382 ID:1 Coordinator: fd00::201:1:1:1
2:53:35.382 ID:1 Routing and Node Statistics:
2:53:35.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 1041   RX: 1032   PRR: 99%   Temp: 25C   RSSI: -63   PING Sent: 1031   PONG Received: 1030   Packet Loss: 0%   RTT: 190 ms
2:53:35.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 1041   RX: 1030   PRR: 99%   Temp: 23C   RSSI: -63   PING Sent: 1030   PONG Received: 1029   Packet Loss: 0%   RTT: 577 ms
2:53:40.382 ID:1 [INFO: Coordinator] Node 2   TX: 1042   RX: 1031   PRR: 99%   Temp: 26C   RSSI: -63   PING Sent: 1032   PONG Received: 1031   RTT: 200 ms
2:53:40.518 ID:1 [INFO: Coordinator] PONG sent to Node 2 [fd00::201:1:1:1] seq_num: 1033
2:53:41.218 ID:1 [INFO: Coordinator] Node 3   TX: 1042   RX: 1031   PRR: 99%   Temp: 20C   RSSI: -63   PING Sent: 1031   PONG Received: 1030   RTT: 1007 ms
2:53:41.649 ID:1 [INFO: Coordinator] PONG sent to Node 3 [fd00::201:1:1:1] seq_num: 1032
2:53:45.382 ID:1 Coordinator: fd00::201:1:1:1
2:53:45.382 ID:1 Routing and Node Statistics:
2:53:45.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 1042   RX: 1033   PRR: 99%   Temp: 26C   RSSI: -63   PING Sent: 1032   PONG Received: 1031   Packet Loss: 0%   RTT: 200 ms
2:53:45.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 1042   RX: 1031   PRR: 99%   Temp: 20C   RSSI: -63   PING Sent: 1031   PONG Received: 1030   Packet Loss: 0%   RTT: 1007 ms

Filter: ID:1

**Hình 32:** Kết quả mô phỏng với hơn 1000 gói

10:45.382 ID:1 Coordinator: fd00::201:1:1:1
10:45.382 ID:1 Routing and Node Statistics:
10:45.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 64   RX: 55   PRR: 85%   Temp: 22C   RSSI: -63   PING Sent: 54   PONG Received: 53   Packet Loss: 1%   RTT: 150 ms
10:45.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 64   RX: 53   PRR: 82%   Temp: 25C   RSSI: -63   PING Sent: 53   PONG Received: 52   Packet Loss: 1%   RTT: 817 ms
10:50.268 ID:1 [INFO: Coordinator] Node 2   TX: 65   RX: 56   PRR: 86%   Temp: 27C   RSSI: -63   PING Sent: 55   PONG Received: 54   RTT: 650 ms
10:50.338 ID:1 [INFO: Coordinator] PONG sent to Node 2 [fd00::201:1:1:1] seq_num: 56
10:50.382 ID:1 Coordinator: fd00::201:1:1:1
10:50.382 ID:1 Routing and Node Statistics:
10:50.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 65   RX: 56   PRR: 86%   Temp: 27C   RSSI: -63   PING Sent: 55   PONG Received: 54   Packet Loss: 1%   RTT: 650 ms
10:50.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 64   RX: 53   PRR: 82%   Temp: 25C   RSSI: -63   PING Sent: 53   PONG Received: 52   Packet Loss: 1%   RTT: 817 ms
10:51.248 ID:1 [INFO: Coordinator] Node 3   TX: 65   RX: 54   PRR: 89%   Temp: 26C   RSSI: -63   PING Sent: 54   PONG Received: 53   RTT: 547 ms
10:51.318 ID:1 [INFO: Coordinator] PONG sent to Node 3 [fd00::201:1:1:1] seq_num: 55
10:55.382 ID:1 Coordinator: fd00::201:1:1:1
10:55.382 ID:1 Routing and Node Statistics:
10:55.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 65   RX: 56   PRR: 86%   Temp: 27C   RSSI: -63   PING Sent: 55   PONG Received: 54   Packet Loss: 1%   RTT: 650 ms
10:55.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 64   RX: 53   PRR: 82%   Temp: 25C   RSSI: -63   PING Sent: 53   PONG Received: 52   Packet Loss: 1%   RTT: 817 ms
11:00.348 ID:1 [INFO: Coordinator] Node 2   TX: 66   RX: 57   PRR: 86%   Temp: 26C   RSSI: -63   PING Sent: 56   PONG Received: 55   RTT: 170 ms
11:00.392 ID:1 [INFO: Coordinator] PONG sent to Node 2 [fd00::201:1:1:1] seq_num: 57
11:00.392 ID:1 Coordinator: fd00::201:1:1:1
11:00.392 ID:1 Routing and Node Statistics:
11:00.392 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 66   RX: 57   PRR: 86%   Temp: 28C   RSSI: -63   PING Sent: 56   PONG Received: 55   Packet Loss: 1%   RTT: 170 ms
11:00.392 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 66   RX: 55   PRR: 83%   Temp: 26C   RSSI: -63   PING Sent: 54   PONG Received: 53   Packet Loss: 1%   RTT: 547 ms
11:00.978 ID:1 [INFO: Coordinator] Node 3   TX: 66   RX: 54   PRR: 83%   Temp: 26C   RSSI: -63   PING Sent: 54   PONG Received: 53   RTT: 547 ms
11:01.328 ID:1 [INFO: Coordinator] PONG sent to Node 3 [fd00::201:1:1:1] seq_num: 55
11:05.382 ID:1 Coordinator: fd00::201:1:1:1
11:05.382 ID:1 Routing and Node Statistics:
11:05.382 ID:1 Node ID 2 [fd00::202:2:2:2] via fe80::201:1:1:1   TX: 66   RX: 57   PRR: 86%   Temp: 28C   RSSI: -63   PING Sent: 56   PONG Received: 55   Packet Loss: 1%   RTT: 170 ms
11:05.382 ID:1 Node ID 3 [fd00::203:3:3:3] via fe80::202:2:2:2   TX: 66   RX: 55   PRR: 83%   Temp: 26C   RSSI: -63   PING Sent: 55   PONG Received: 54   Packet Loss: 1%   RTT: 557 ms
11:10.288 ID:1 [INFO: Coordinator] Node 2   TX: 67   RX: 58   PRR: 86%   Temp: 25C   RSSI: -63   PING Sent: 57   PONG Received: 56   RTT: 160 ms
11:10.358 ID:1 [INFO: Coordinator] PONG sent to Node 2 [fd00::201:1:1:1] seq_num: 58

**Hình 33:** Kết quả mô phỏng ở những gói tin thấp

Thực hiện đo độ trễ gói tin thông qua tính giá trị trung bình của các RTT ta được kết quả như sau:

Node ID 2: minRTT=150 ms / aveRTT=266.82 ms / maxRTT=3780 ms
Node ID 3: minRTT=357 ms / aveRTT=857.60 ms / maxRTT=4907 ms

Nhận xét kết quả mô phỏng:

- Các node đã chạy được Multihop
- Ở những gói tin đầu, do mạng chưa ổn định nên độ mất gói sẽ cao, sau khi mạng ổn định ở khoảng sau 100 gói thì tỉ lệ mất gói xuống rất thấp, đạt 1%

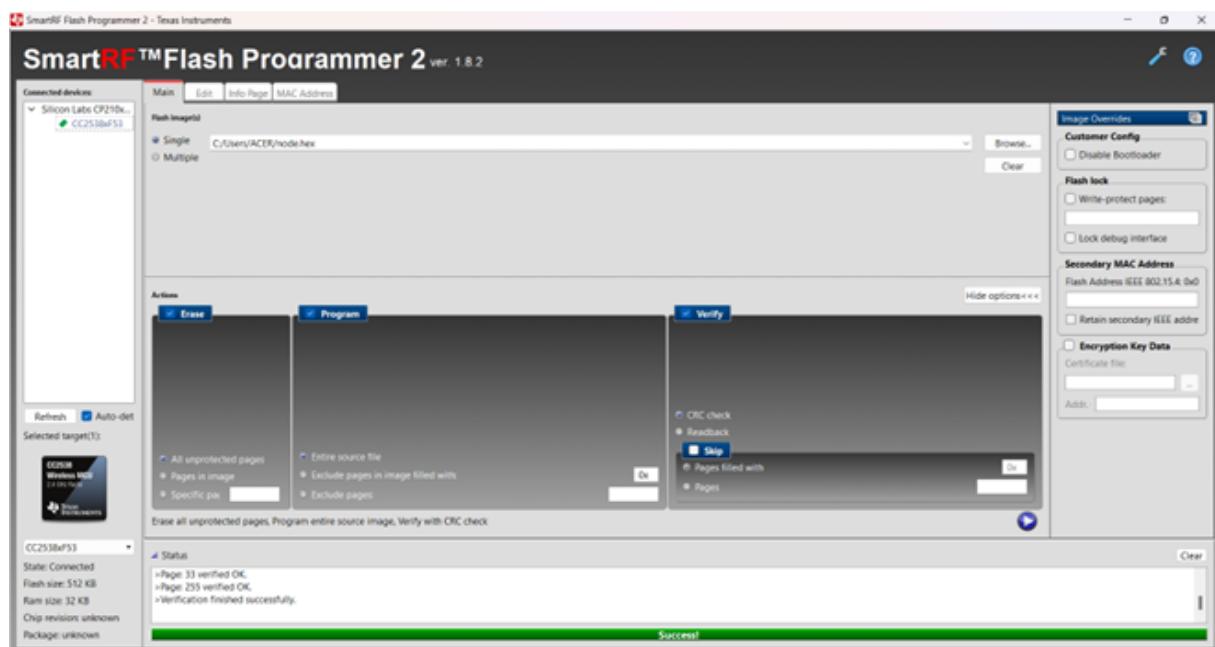
- Độ trễ trung bình gói tin đối với node 2 chỉ 0,2ms và đối với node 3 là 0,8ms phù hợp cho mạng cảm biến truyền dữ liệu hiện tại, không yêu cầu tính trễ thấp.
- Đối với mô phỏng ở hơn 100 gói tin thì tỉ lệ mất gói là dưới 1%, tỉ lệ nhận được dữ liệu data từ các node sensor là 99% cho thấy mạng chạy khá ổn định, có độ tin cậy cao.

## PHẦN 5: Thực hiện triển khai mạng 6TiSCH trên phần cứng CC2538dk

### 5.1. Thực hiện nạp lên phần cứng CC2538dk

Nạp chương trình Coordinator và Node lên phần cứng CC2538. Sử dụng phần mềm Flash Programmer của TI để nạp file .hex

Đưa CC2538 vào chế độ Bootloader và thực hiện nạp file .hex vào, kết quả sau khi nạp sẽ được hiển thị thành công tại Status như bên dưới



Hình 34: Kết quả nạp code lên CC2538dk thành công

### 5.2. Chạy thực nghiệm kết quả mô phỏng

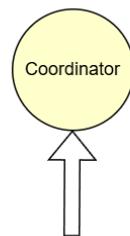
Tiếp theo, bố trí sơ đồ mạng cảm biến tương tự như mô phỏng với 3 node , 1 node làm Coordinator và 2 node còn lại tham gia vào mạng.

Bố trí sơ đồ thực tế được đo tại sân bóng KTX khu A ĐHQG TPHCM, với các node cảm biến được đặt cách nhau 28m, đây là khoảng cách ngoài trời có ít vật cản và người qua lại. Sơ đồ bố trí trên bản đồ được chụp từ Google map như ảnh.

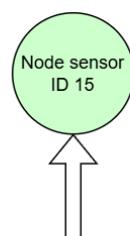


Hình 35: Hình ảnh bố trí thực tế

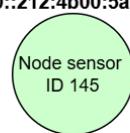
fe80::212:4b00:5a9:8f04



fe80::212:4b00:5a9:8f0f



fd00::212:4b00:5a9:8f0f



Hình 36: Sơ đồ minh họa bố trí phần cứng thực tế

Dữ liệu từ Coordinator sẽ được in ra Serial hiển thị để tiện theo dõi, xuất file dữ liệu đầu ra dưới dạng file .log để thực hiện tính toán thêm các giá trị như độ trễ trung bình(RTT average). Dưới đây là kết quả đầu ra của coordinator

```

Routing and Node Statistics:
Node ID 15 [fd00::212:4b00:5a9:8f04] via fe80::212:4b00:5a9:8f04 | TX: 444 | RX: 389 | PRR: 87% | Temp: 21C | RSSI: -70 | PING Sent: 426 | PONG Received: 407 | Packet Loss: 4% | RTT: 297 ms
Node ID 145 [fd00::212:4b00:5a9:8f0f] via fe80::212:4b00:5a9:8f0f | TX: 405 | RX: 372 | PRR: 91% | Temp: 29C | RSSI: -70 | PING Sent: 400 | PONG Received: 378 | Packet Loss: 5% | RTT: 1208 ms
[INFO: Coordinator] Node 15 | TX: 445 | RX: 390 | PRR: 87% | Temp: 25C | RSSI: -70 | PING Sent: 427 | PONG Received: 408 | RTT: 1417 ms
[INFO: Coordinator] PONG sent to Node 15 [fd00::212:4b00:5a9:8f04] seq_num: 428
Coordinator: fd00::212:4b00:5a9:8f04
Routing and Node Statistics:
Node ID 15 [fd00::212:4b00:5a9:8f04] via fe80::212:4b00:5a9:8f04 | TX: 445 | RX: 390 | PRR: 87% | Temp: 25C | RSSI: -70 | PING Sent: 427 | PONG Received: 408 | Packet Loss: 4% | RTT: 1417 ms
Node ID 145 [fd00::212:4b00:5a9:8f01] via fe80::212:4b00:5a9:8f0f | TX: 405 | RX: 372 | PRR: 91% | Temp: 29C | RSSI: -70 | PING Sent: 400 | PONG Received: 378 | Packet Loss: 5% | RTT: 1208 ms
[INFO: Coordinator] Node 15 | TX: 446 | RX: 391 | PRR: 87% | Temp: 27C | RSSI: -80 | PING Sent: 428 | PONG Received: 409 | RTT: 647 ms
[INFO: Coordinator] PONG sent to Node 15 [fd00::212:4b00:5a9:8f04] seq_num: 429
Coordinator: fd00::212:4b00:5a9:8f04
Routing and Node Statistics:
Node ID 15 [fd00::212:4b00:5a9:8f0f] via fe80::212:4b00:5a9:8f04 | TX: 445 | RX: 390 | PRR: 87% | Temp: 25C | RSSI: -70 | PING Sent: 427 | PONG Received: 408 | Packet Loss: 4% | RTT: 1417 ms
Node ID 145 [fd00::212:4b00:5a9:8f01] via fe80::212:4b00:5a9:8f0f | TX: 405 | RX: 372 | PRR: 91% | Temp: 29C | RSSI: -70 | PING Sent: 400 | PONG Received: 378 | Packet Loss: 5% | RTT: 1208 ms
[INFO: Coordinator] Node 15 | TX: 446 | RX: 391 | PRR: 87% | Temp: 27C | RSSI: -80 | PING Sent: 428 | PONG Received: 409 | RTT: 647 ms
Node ID 145 [fd00::212:4b00:5a9:8f0f] via fe80::212:4b00:5a9:8f04 | TX: 446 | RX: 391 | PRR: 87% | Temp: 27C | RSSI: -80 | PING Sent: 428 | PONG Received: 409 | Packet Loss: 4% | RTT: 647 ms
Node ID 145 [fd00::212:4b00:5a9:8f01] via fe80::212:4b00:5a9:8f0f | TX: 405 | RX: 372 | PRR: 91% | Temp: 29C | RSSI: -70 | PING Sent: 400 | PONG Received: 378 | Packet Loss: 5% | RTT: 1208 ms

```

Hình 37: Kết quả sau khi chạy thực tế

### 5.3. Nhận xét kết quả thực tế

Kết quả chạy thực tế đạt gần với kết quả mô phỏng trên cooja, các node thực hiện gửi được dữ liệu đến coordinator và chạy multi-hop được thấy rõ thông qua bảng định tuyến với node ID 145 chuyển tiếp đến node ID 15 trước khi đến Coordinator.

Kết quả thu được cường độ tín hiệu:

- Đối với node ID 15 là -80 và ID 145 là -70 mặc dù node ID 145 được bố trí xa hơn là vì giá trị -70 được lấy từ node parent của nó là ID 15 và chuyển tiếp đến root để in ra nên có giá trị như

Thông kê kết quả PING dữ liệu để đánh giá độ tin cậy mạng:

- PING Sent (ping\_sent\_count): Số lượng gói Ping đã gửi.
- PONG Received (pong\_received\_count): Số lượng gói Pong nhận được.
- Packet Loss:

$$\text{Packet Loss} = \left( 1 - \frac{\text{PONG Received}}{\text{PING Sent}} \right) \times 100\%$$

- RTT (Round Trip Time): Thời gian để nhận được gói Pong kể từ khi gửi gói Ping.

Kết quả thực hiện Ping gói tin trong mạng:

- Node ID 15 thực hiện Ping 428 lần và nhận phản hồi Pong 409 lần, tỉ lệ mất gói của lần Ping này là 4%.
- Node ID 145 thực hiện Ping thông qua node ID15 đến Coordinator, Ping 400 lần và nhận 378 phản hồi Pong, tỉ lệ mất gói 5%.
- Tỉ lệ nhận dữ liệu nhiệt độ song song cùng với Ping là 87% đối với ID 15 và 91% đối với ID 145.

Các giá trị về tỉ lệ mất gói sẽ thấp hơn nếu mạng chạy ổn định với số gói tin truyền lớn hơn, nhưng kết quả trên cũng có thể thấy rằng mạng chạy có tính ổn định và tỉ lệ mất gói không quá cao.

Sau khi đã tiến hành đo mất gói, ta thực hiện đo độ trễ gói tin (RTT) thông qua chương trình Python (Ave\_RTT.py) tính giá trị Round-Trip Time (RTT) để xác định độ trễ Min , Average, và Max của từng node.

Dưới đây là thông số RTT sau khi chạy dữ liệu của đầu ra Coordinator:

Node ID 15: minRTT=225 ms / aveRTT=334.12 ms / maxRTT=4567 ms

Node ID 145: minRTT=438 ms / aveRTT=1400.80 ms / maxRTT=8908 ms

Khi chạy kết quả thực tế, độ trễ trung bình của node ID 15 là 334,12ms và node ID 145 là 1400,80ms. Giá trị độ trễ sẽ tăng dần theo khoảng cách hoặc thông qua nhiều tuyến.

Vì mạng cảm biến hiện tại không có yêu cầu độ trễ thấp nên với độ trễ trên vẫn phù hợp với các yêu cầu phục vụ theo dõi, báo cáo định kỳ, không đòi hỏi hành động ngay lập tức.

## PHẦN 6: Kết luận và hướng phát triển

### 6.1. Kết luận

Đồ án đã đạt được mục tiêu là tìm hiểu và xây dựng được một mạng 6TiSCH, thực hiện mô phỏng và chạy thực tế được một mạng với đầy đủ các giao thức Netstack đã đề ra bao gồm TSCH, 6top và 6TiSCH. Hệ thống chạy ổn định ở môi trường điều kiện kiểm tra.

Quá trình thực hiện hiện Đồ án lần này mang ý nghĩa lớn, mang lại kiến thức và kĩ năng trong lĩnh vực IoT và hệ thống nhúng, nâng cao kĩ năng lập trình C cho hệ điều Contiki-NG, bên cạnh đó rèn luyện tư duy cũng như khả năng sửa lỗi khi gặp vấn đề. Phát triển kĩ năng mềm như viết Latex, tự xử lý vấn đề gấp phải, tìm kiếm tài liệu tham khảo.

Bên cạnh những kiến thức và kĩ năng học được, vẫn còn tồn tại một số hạn chế như sau:

- Hệ thống chỉ mới thu thập thông tin, chưa có phần điều khiển từ người dùng.
- Hệ thống chỉ mới đánh giá về độ ổn định của mạng 6Tisch ở số lượng node nhỏ, chưa được kiểm tra với những trường hợp khác đối với mạng như tải lớn, nhiều cao,...
- Chưa xây dựng được một hệ xử lý và quản lý dữ liệu dành cho người dùng như webserver ...

### 6.2. Hướng phát triển

Hướng phát triển của đồ án trong tương lai:

- Xây dựng mạng cảm biến với số lượng node cao, ứng dụng vào môi trường có nhiều nhiễu, cần tiết kiệm năng lượng.
- Đo đạc và kiểm tra hiệu năng của hệ thống một cách khoa học.
- Phát triển thuật toán kiểm tra dữ liệu theo hướng sự kiện, thay vì sử dụng theo chu kì.

- Xây dựng web và một màn hình hiển thị trên gateway để tiện quan sát theo dõi.

## PHẦN 7. Tài liệu tham khảo

- [1 ] Oikonomou, George, et al. "The Contiki-NG open source operating system for next generation IoT devices." SoftwareX 18 (2022): 101089.
- [2 ] Tanaka, Yasuyuki, Toshio Ito, and Fumio Teraoka. "6TiSCH Scheduling Function Design Suite founded on Contiki-NG." Journal of information processing 30 (2022): 669-678.
- [3 ] Kharb, Seema, and Anita Singhrova. "A survey on network formation and scheduling algorithms for time slotted channel hopping in industrial networks." Journal of Network and Computer Applications 126 (2019): 59-87.
- [4 ] Winter, Tim. "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks". tools.ietf.org. Retrieved 2018-10-13.
- [5 ] Kurniawan, Agus. "Practical Contiki-NG." Pract. Contiki-NG (2018).
- [6 ] Vo Que Son. 2024. "Design and develop IOT applications" [Lecture slides]. HK241. HCMUT.