`

ĐỒ ÁN THIẾT KẾ

ĐỀ TÀI: HỆ ĐIỀU HÀNH MÃ NGUỒN MỞ CHO IOT VÀ MẠNG CẢM BIẾN KHÔNG DÂY

Giảng viên hướng dẫn: PGS. TS. Nguyễn Quốc Cường

Nhóm sinh viên thực hiện

Họ và tên:	**MSSV**
Nguyễn Đình Khánh	20181549
Vũ Hoàng Mạnh	20191956

HÀ NỘI, 7/2023

LỜI MỞ ĐẦU

IoT đang là một lĩnh vực rất đáng quan tâm trong thời đại công nghệ 4.0, dự kiến sẽ có khoảng 28 tỷ thiết bị kết nối trong đó có 15 tỷ thiết bị kết nối IoT bao gồm thiết bị M2M (machine to machine). Khi các thiết bị ngày càng trở nên thông minh, yêu cầu về truyền thông giữa các vật (things) với nhau trong mạng IoT một cách hiệu quả ngày càng được chú trọng.
Chính vì thế, việc nghiên cứu, qua đó ứng dụng được những hệ thống và giao thức định tuyến trong mạng IoT để mạng đạt hiệu suất tốt nhất là việc làm rất quan trọng. Dựa trên cơ sở đó, Đồ án giới thiệu đề tài: "Hệ điều hành mã nguồn mở cho IoT và mạng cảm biến không dây"

Trong quá trình thực hiện Đồ án, chúng em nhận được sự hướng dẫn tận tình từ PGS.TS. Nguyễn Quốc Cường, qua những lần trao đổi là những lần chúng em học được rất nhiều, dần định hình được mục tiêu và ứng dụng của Đồ án, chúng em chân thành cảm ơn thầy.
Phần nội dung của Đồ án sẽ không thể tránh khỏi được những thiếu sót, chúng em mong nhận được những lời góp ý từ thầy.

MỤC LỤC

DANH MỤC HÌNH ẢNH 4
Yêu cầu của dự án 6
Giới thiệu thành viên của dự án 7
Kế hoạch thực hiện chung của dự án 8
Kế hoạch và nội dung thực hiện của từng thành viên 10
Tự đánh giá tỷ lệ đóng góp của từng thành viên trong dự án theo kế hoạch (trước khi thực hiện, thực hiện trong khi lên kế hoạch thực hiện dự án) 13
Chương 1. Tổng quan hệ điều hành contiki 14
1.1. Khái niệm 14
1.2. Đặc điểm riêng 14
1.3. Truyền thông 15
1.4. Cấu trúc 15
1.5. Hướng dẫn tải và cài đặt 16
Chương 2. Tổng quan về Cooja Simulation 18
2.1. Giới thiệu 18
2.2. Công cụ hỗ trợ mô phỏng hệ điều hành Contiki 18
2.3. Hướng dẫn cài đặt và chạy phần mềm 21
2.4. Ứng dụng Cooja Simulation và các ví dụ cơ bản 22
2.4.1 Mô phỏng Cooja cho người mới bắt đầu 22
2.4.2 Mô phỏng RPL-UDP trong Cooja 26
Chương 3. Giao thức RPL và ContikiRPL 35

3.1. Tổng quan RPL 35
3.1.1. Khái niệm, thuật ngữ sử dụng trong RPL 35
3.1.2. RPL Identifiers 38
3.1.3. Các loại bản tin trong định tuyến RPL 38
3.1.4. Quá trình khởi tạo mạng 43
3.1.5. Quá trình định tuyến upward 44
3.1.6. Quá trình định tuyến downward 45
3.2. Các hàm mục tiêu cơ bản trong RPL 48
3.2.1. The Minimum Rank with Hysteresis Objective Function – MRHOF 48
3.2.2. Objective Function 0 - OF0 57
3.3. Tổng quan về Contiki-RPL 58
Chương 4. Áp dụng mô phỏng bài báo khoa học 60
4.1. Abstract - Tổng quan 60
4.2. Introduction - Giới thiệu 60
4.3. Background và các nghiên cứu liên quan 63
4.4. Đề xuất thực thi MAB-Learning cho việc bảo tồn năng lượng và sự mất mát gói bản tin của mạng IoT 65
4.5. Kết quả thực nghiệm và thảo luận 67
4.6. Mô phỏng mô hình đánh giá năng lượng tiêu thụ trên các node trong bài báo 71
Thử nghiệm và Đánh giá 82
Hoàn thiện sản phẩm 82
Kết luận 82
TÀI LIỆU THAM KHẢO 89

DANH MỤC HÌNH ẢNH

Hình 1:Hình ảnh Contiki.....14
Hình 2:Cửa sổ Terminal.....16
Hình 3: Tải xuống contiki-3.0.zip.....16
Hình 4: Giải nén file tải xuống 17
Hình 5: Đổi tên thư mục thành contiki 17
Hình 6: Cài đặt các gói cần thiết.....17
Hình 7: Giao diện Cooja 18
Hình 8: Cửa sổ Time line.....19
Hình 9: Cửa sổ Mote output.....20
Hình 10: Cửa sổ network 21
Hình 11: Cửa sổ Terminal.....21
Hình 12: Chuyển thư mục làm việc đến tool Cooja 22
Hình 13: Chạy phần mềm mô phỏng Cooja trên Contiki 22
Hình 14: Phần mềm mô phỏng Cooja.....22
Hình 15: Chạy phần mềm mô phỏng Cooja trên contiki-ng.....22
Hình 16: Tạo tên thư mục 23
Hình 17: Chọn loại node là Sky Mote Type 23
Hình 18: Đoạn code Hello Word 24
Hình 19: Complie file code.....25
Hình 20: Lựa chọn số lượng node.....25
Hình 21: Bắt đầu truyền tin.....26
Hình 22: Cửa sổ Mote output cho Hello World.....26
Hình 23: Tạo tên cho file mô phỏng RPL_UDP.....27
Hình 24: Chọn loại node Sky Mote Type và complile file udp.server.....27
Hình 25 25:Chọn số node server.....29
Hình 26:Chọn loại node Sky và complile file udp.client.....30
Hình 27:Chọn số node client.....33

Hình 28:Kết quả nhận được trong Mote output 34
Hình 29:RPL DODAG 36
Hình 30: DODAG Sequence Number 37
Hình 31: DODAGVersionNumber 38
Hình 32:Bản tin DIO 39
Hình 33:Bản tin DAO 41
Hình 34:Bản tin DIS 42
Hình 35:Bản tin DAO-ACK 43
Hình 36:Chế độ không lưu trữ Non-storing 46
Hình 37:Chế độ lưu trữ Storing 47
Hình 38: Định các thông số trong MRHOF trên ContikiOS 54
Hình 39: Chỉ số liên kết node cha MRHOF trong ContikiOS 55
Hình 40: Chi phí đường truyền của node cha 55
Hình 41: Tính rank thông qua node cha 56
Hình 42: Lựa chọn node cha có thể sử dụng 56
Hình 43: Node cha tốt nhất 57
Hình 44:Contiki-RPL 58
Hình 45:Cấu trúc module của Contiki RPL 58
Hình 46:Cấu trúc RPL DAG với DODAG 64
Hình 47:Bảng thông số đưa vào mô phỏng 67
Hình 48: Biểu đồ tỷ lệ gói tin truyền được trong các mạng DAG có kích cỡ khác nhau 68
Hình 49:Biểu đồ số lượng chi phí điều khiển dio trong các mạng có kích thước DAG khác nhau 68
Hình 50:Biểu đồ tổng năng lượng tiêu thụ của CPU trong các mạng DAG có kích thước khác nhau 69
Hình 51:Biểu đồ tổng năng lượng tiêu thụ của các mạng DAG có kích thức khác nhau 69
Hình 52:Độ trễ của gói tin E2E trong các mạng DAG có kích thước khác nhau 70
Hình 53:Định tuyến RPL 71
Hình 54:Mote output 72
Hình 55: Điều chỉnh Makefile 73
Hình 56: Thêm Process năng lượng 73
Hình 57: Một phần trong Process Energest 74
Hình 58: Topo mạng trong Cooja 75
Hình 59: Mote Output cho mạng 75
Hình 60: Topology mạng 76
Hình 61:Năng lượng tiêu thụ các node trong mạng 77
Hình 62:Bảng tổng hợp các thông số mạng (1) 77
Hình 63:Bảng tổng hợp các thông số mạng (2) 78

TỔ CHỨC THỰC HIỆN

Yêu cầu của dự án

Nghiên cứu hệ điều hành mã nguồn mở cho thiết bị IoT, mạng cảm biến không dây

Yêu cầu	Chức năng, Thông số,..	Mức độ ưu tiên
Nghiên cứu hệ điều hành Contiki OS và mô phỏng trên Cooja	Các công cụ trong Cooja, khả năng mô phỏng của Cooja, điểm mạnh, điểm yếu của Cooja. Nghiên cứu hệ điều hành Contiki	1
Áp dụng triển khai trên phần cứng	MSP430, zigbee, 802.15.4	2

Giới thiệu thành viên của dự án

	Họ và tên: Nguyễn Đình Khánh MSSV: 20181549 Phụ trách công việc: Tìm hiểu Cooja, ContikiOS, nghiên cứu mô phỏng bài báo khoa học, nghiên cứu RPL, hàm mục tiêu trong RPL, triển khai ContikiOS cho phần cứng cơ bản, người lên kế hoạch cho dự án
(ảnh)	Họ và tên: Vũ Hoàng Mạnh MSSV: 20191956 Phụ trách công việc: Tìm hiểu Cooja, ContikiOS, nghiên cứu mô phỏng bài báo khoa học, nghiên cứu RPL, hàm mục tiêu trong RPL, triển khai ContikiOS cho phần cứng cơ bản

Kế hoạch thực hiện chung của dự án

Nội dung	Kết quả cần đạt	Thời gian (tuần)	Ghi chú
Tìm hiểu nội dung khái quát về dự án	Báo cáo tổng quan, đánh giá	T1-T3	
Lên phương án sơ bộ	Cài đặt các phần mềm cần thiết: Contiki, Cooja	T3-T4	ĐẠT
Làm quen với hệ điều hành Contiki, công cụ mô phỏng cooja Đọc hiểu khái quát bài báo cho dự án	Phân tích khả năng mô phỏng của cooja, lợi ích sử dụng cooja cùng với contiki Đọc hiểu sơ bộ nội dng bài báo	T4-T6	ĐẠT
Phân tích yêu cầu và mục đích của bài báo khoa học Mô phỏng bài toán trong bài báo	Mô hình hóa thành công bài báo trong bài toán và mô phỏng mô hình đó	T6-T8	Để mô hình hóa được bài toán trong bài báo, cần đọc hiểu kiến thức về mạng RPL, các hàm mục tiêu thực hiện trong mạng RPL. Để mô phỏng được năng lượng tiêu thụ như trong bài báo, cần tìm hiểu module Energest, module tính toán năng lượng được contiki hỗ trợ.
Viết tài liệu hướng dẫn sử dụng Cooja và Contiki	Đưa ra bản word hướng dẫn cài đặt Cooja, Contiki và mô phỏng mô hình bài báo	T8-T9	
Quay video hướng dẫn dựa trên bản word đã báo cáo	Cho ra video được chia làm các phần hướng dẫn nhỏ	T9-T10	Video: 1. https://www.youtube.com/watch?v=Pb2P6y2UsT 2. https://www.yout

			ube.com/watch?v=2xKL-rR7EgA 3. https://www.youtube.com/watch?v=JnTErHXQSa4&t=819s

Kế hoạch và nội dung thực hiện của từng thành viên

Nguyễn Đình Khánh

Nội dung	Mô tả (tính năng, thông số,…)	Kết quả cần đạt	Thời gian thực hiện (theo tuần)	Ghi chú
Tìm hiểu thêm về công cụ Cooja	Tìm hiểu các công cụ khác trong cooja simulation	Các công cụ trong mô phòng như Radio message, Breack point, Buffer View, RSSI base	Tuần 10 - 11	
Mô phỏng thêm sô lượng node trong mạng	Nâng số node lên 20-100 node như trong bài báo, đánh giá về năng lượng, CPU. Radio như trong bài báo. Viết lại file python	Nâng số node lên 20-100 node như trong bài báo, đánh giá về năng lượng, CPU. Radio như trong bài báo	Tuần 11-12	
Cách hàm mục tiêu MRHOF được thực thi trong Contiki	Mạng RPL dùng hàm mục tiêu để thiết lập DADOG, bảng định tuyến. Cần hiểu được cách triển khai code hàm mục tiêu thế nào	Hiểu được code thực thi hàm MRHOF	Tuần 12-13	
Thực thi Contiki trên phần cứng	Port contiki os vào vi điều khiển msp430 và thực hiện một số chương trình cơ bản	Thực hiện các chương trình như lấy năng lượng từ CPU, Udp-server	Tuần 13-14	
Thêm các	Thêm các node MSP430 để hiểu được	Dùng chương trình rpl-udp	Tuần 14-15	

node phần cứng	giao thức RPL thực thi trên phần cứng như thế nào	và module energest trong contiki		
Áp dụng thuật toán học máy	Thuật toán RL được áp dụng trong contiki os ở đâu và thực hiện thế nào	Code lại chương trình cho foldẻ rpl-lite để áp dụng thuật toán học máy nếu được	Tuần 15-16	
Làm báo cáo và trình bày kết thúc dự án	Tổng hợp kiên thức thông qua dự án	Làm Slide, báo cáo		

Vũ Hoàng Mạnh

Nội dung	Mô tả (tính năng, thông số,…)	Kết quả cần đạt	Thời gian thực hiện (theo tuần)	Ghi chú
Tìm hiểu thêm về công cụ Cooja	Tìm hiểu các công cụ khác trong cooja simulation	Các công cụ trong mô phòng như Radio message, Breack point, Buffer View, RSSI base	Tuần 10 - 11	
Mô phỏng thêm sô lượng node trong mạng	Nâng số node lên 20-100 node như trong bài báo, đánh giá về năng lượng, CPU. Radio như trong bài báo. Viết lại file python	Nâng số node lên 20-100 node như trong bài báo, đánh giá về năng lượng, CPU. Radio như trong bài báo	Tuần 11-12	
Cách hàm mục	Mạng RPL dùng hàm mục tiêu để thiết lập DADOG, bảng định	Hiểu được code thực thi hàm OF0, so	Tuần 12-13	

tiêu OF0 được thực thi trong Contiki	tuyến. Cần hiểu được cách triển khai code hàm mục tiêu thế nào	sánh với MRHOF		
Thực thi Contiki trên phần cứng	Port contiki os vào vi điều khiển msp430 và thực hiện một số chương trình cơ bản	Thực hiện các chương trình như lấy năng lượng từ CPU, Udp-server	Tuần 13-14	
Thêm các node phần cứng	Thêm các node MSP430 để hiểu được giao thức RPL thực thi trên phần cứng như thế nào	Dùng chương trình rpl-udp và module energest trong contiki	Tuần 14-15	
Áp dụng thuật toán học máy	Thuật toán RL được áp dụng trong contiki os ở đâu và thực hiện thế nào	Code lại chương trình cho foldẻ rpl-lite để áp dụng thuật toán học máy nếu được	Tuần 15-16	
Làm báo cáo và trình bày kết thúc dự án	Tổng hợp kiên thức thông qua dự án	Làm Slide, báo cáo		

Tự đánh giá tỷ lệ đóng góp của từng thành viên trong dự án theo kế hoạch (trước khi thực hiện, thực hiện trong khi lên kế hoạch thực hiện dự án)

Người thực hiện	Tỷ lệ	Giải quyết được những vấn đề gì của dự án (cần ghi rõ để có cơ sở đánh giá tỷ lệ)
Nguyễn Đình Khánh	xx%	Phụ trách công việc: Tìm hiểu Cooja, ContikiOS, nghiên cứu mô phỏng bài báo khoa học, nghiên cứu RPL, hàm mục tiêu trong RPL, triển khai ContikiOS cho phần cứng cơ bản, người lên kế hoạch cho dự án
Vũ Hoàng Mạnh	xx%	Tìm hiểu Cooja, ContikiOS, nghiên cứu mô phỏng bài báo khoa học, nghiên cứu RPL, hàm mục tiêu trong RPL, triển khai ContikiOS cho phần cứng cơ bản

Chương 1. Tổng quan hệ điều hành contiki

Với số lượng dày đặc các nút cảm biến, việc kiểm nghiêm và đánh giá chính xác hiệu quả hoạt động là một trong những thách thức của mạng cảm biến không dây trong quá trình triển khai trên thực tế. Nhằm khắc phục những khó khăn này, nhiều hướng nghiên cứu đã tập trung vào phát triển những công cụ mô phỏng cách thức hoạt động các nút trong hệ thống mạng cảm biến. Trong phần này sẽ trình bày về hệ điều hành contiki.

1.1. Khái niệm

Hệ điều hành contiki là hệ điều hành mã nguồn mở, được nghiên cứu, thiết kế và phát triển bởi một nhóm các nhà phát triển từ viện khoa học máy tính Thụy Điển, người đứng đầu là Adam Dunkels. Nhóm phát triển Contiki gồm nhiều thành viên đến từ SICS, CISCO, cùng nhiều tổ chức và các trường đại học khác trên thế giới. Hệ điều hành Contiki được thiết kế cho các vi điều khiển có bộ nhớ nhỏ, với thông số 2KB RAM và 40KB ROM. Nhờ đó, Contiki được sử dụng cho các hệ thống nhúng và các ứng dụng trong mạng cảm biến không dây. Contiki bắt đầu được nghiên cứu từ năm 2001 và phát hành phiên bản đầu tiên Contiki 1.0 năm 2003.

Trong thực tế, Contiki đã được ứng dụng trong nhiều dự án như giám sát đường hầm xe lửa, theo dõi nước trong biển Baltic,… Nhiều cơ chế, ý tưởng trong Contiki đã được ứng dụng rộng rãi trong công nghiệp. Điển hình như mô hình uIP được phát hành năm 2001 đã được sử dụng trong hệ thống ứng dụng của hàng trăm công ty trong các lĩnh vực hàng hải, thông tin vệ tinh, khai thác dầu mỏ,…; mô hình Protothreads được công bố lần đầu tiên năm 2005, đến nay đã được sử dụng trong nhiều ứng dụng như bộ giải mã kỹ thuật số và thiết bị cảm biến rung không dây.

Hình 1:Hình ảnh Contiki

1.2. Đặc điểm riêng

Hệ điều hành Contiki được lập trình bằng ngôn ngữ C, hoạt động dựa trên cơ chế event - driven và có những đặc điểm phù hợp với các hệ thống nhúng và mạng cảm biến không dây:

- Contiki được chia thành nhiều modul hoạt động độc lập. Nhờ đó các ứng dụng có thể sử dụng các modul một cách linh động và chỉ load những modul cần thiết.
- Cơ chế hoạt động điều khiển sự kiện làm giảm năng lượng tiêu hao và hạn chế dung lượng bộ nhớ cần sử dụng.
- Có thể sử dụng IP trong mạng cảm biến thông qua uIP stack được xây dựng dựa trên nền TCP/IP.
- Có những modul cho phép ước lượng và quản lý năng lượng một cách hiệu quả.
- Các giao thức tương tác giữa các lớp và các node trong mạng dễ dàng hơn.
- Sử dụng RIME stack phục vụ các giao thức dành cho mạng năng lượng thấp một cách hiệu quả.

Bên cạnh đó, Contiki còn cung cấp những công cụ hỗ trợ mô phỏng với giao diện đơn giản, dễ sử dụng và hỗ trợ tốt những thiết bị trong thực tế, phục vụ những mục đích nghiên cứu, mô phỏng và triển khai những giao thức mới.

1.3. Truyền thông

Contiki hỗ trợ các giao thức tiêu chuẩn và các giao thức kích hoạt gần đây cho IoT

1. uIP (dành cho Ipv4) – Việc triển khai TCP/IP này hỗ trợ các bộ vi điều khiển 8-bit và 16-bit.
2. uIPv6 (dành cho IPv6) − Đây là phần mở rộng IPv6 tương thích hoàn toàn với uIP.
3. Rime − Ngăn xếp thay thế này cung cấp giải pháp khi IPv4 hoặc IPv6 bị cấm. Nó cung cấp một bộ nguyên thủy cho các hệ thống năng lượng thấp.
4. 6LoWPAN − Đây là viết tắt của IPv6 trên mạng khu vực cá nhân không dây công suất thấp. Nó cung cấp công nghệ nén để hỗ trợ mạng không dây tốc độ dữ liệu thấp cần thiết cho các thiết bị có tài nguyên hạn chế.
5. RPL - Giao thức IPv6 véc tơ khoảng cách này dành cho LLN (mạng công suất thấp và mất dữ liệu) cho phép tìm thấy đường dẫn tốt nhất có thể trong một mạng phức hợp gồm các thiết bị có khả năng đa dạng.
6. CoAP − Giao thức này hỗ trợ giao tiếp cho các thiết bị đơn giản, điển hình là các thiết bị yêu cầu giám sát từ xa nặng.

1.4. Cấu trúc

Bất kỳ bản Contiki nào cũng gồm 7 thư mục : apps, core, cpu, docs, example, platform và tools.

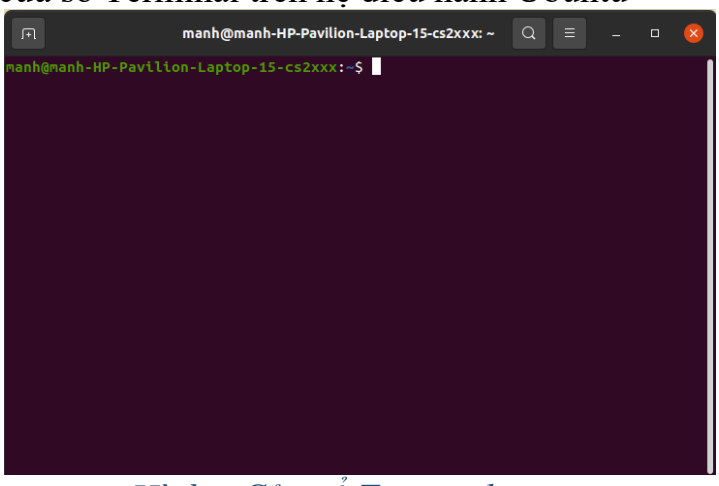
- “apps” chứa các tập tin nguồn của các tiện ích phát triển cho Contiki. Chúng có sẵn để sử dụng và bao gồm các thiết lập cơ bản của các ứng dụng cho mạng cảm biến không dây . ứng dụng tiêu biểu trong thư mục này là trình duyệt web, máy chủ Web, FTP, email, và máy tính.
- “core”, như tên gọi cho thấy, nó chứa các hạt nhân của hệ điều hành Contiki. Nó chứa khoảng 300 file, gần một nửa trong số đó là tập tin tiêu đề chứa các khai báo và còn lại là các tập tin nguồn chứa cài đặt.

- “cpu” chứa bộ xử lý cụ thể việc thực hiện các chức năng khác nhau được sử dụng trong hệ điều hành.
- “docs” được sử dụng trong việc chuẩn bị tài liệu cho Contiki. Nó chứa thông tin sẽ được sử dụng bởi một hệ thống tài liệu điển hình như Doxygen. “examples” chứa các chương trình ví dụ đơn giản bắt đầu với “Hello-world”, mà phục vụ như là bước đầu tiên hướng tới Contiki lập trình.
- “platform” bao gồm thông tin cụ thể liên quan đến nền tảng Node cảm biến như ESB, Sky, vv "native" là một nền tảng đặc biệt là xây dựng một hệ thống toàn bộ Contiki như một chương trình chạy trên hệ thống phát triển.
- “tools” là thư mục mà các công cụ phần mềm đặc biệt được lưu trữ. 'Cooja' là một Java dựa trên mô phỏng cho Contiki.Thư mục này cũng chứa các công cụ nền tảng cụ thể. ví dụ điển hình là các công cụ cho Sky.

1.5. Hướng dẫn tải và cài đặt

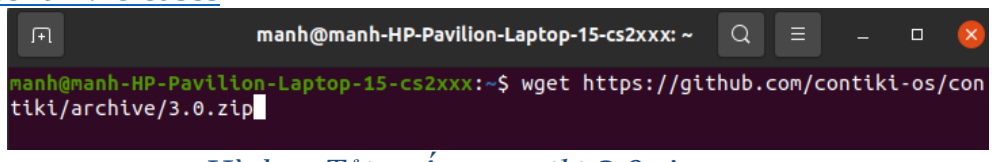
- Cài đặt hệ điều hành Ubuntu
- Link tham khảo: [Hướng dẫn cài đặt Ubuntu 20.04 LTS mới nhất song song với Windows cực dễ - YouTube]
- Cài đặt Contiki trên hệ điều hành Ubuntu
- Link tham khảo: [Installation - Contiki (usc.edu)]

Bước 1: Mở cửa sổ Terminal trên hệ điều hành Ubuntu



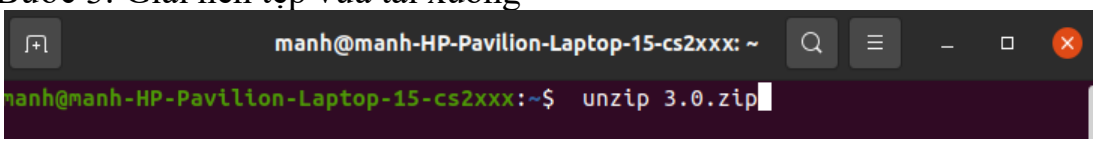
Hình 2:Cửa sổ Terminal

Bước 2: Tải xuống contiki-3.0.zip từ [https://github.com/contiki-os/contiki/releases]



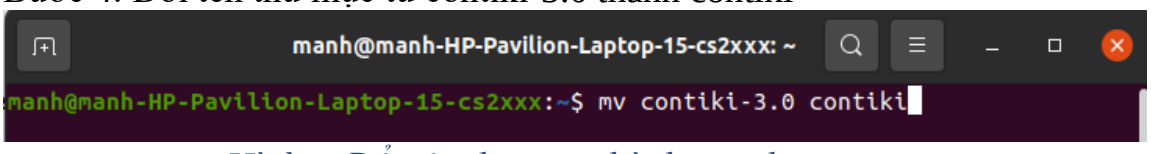
Hình 3: Tải xuống contiki-3.0.zip

Bước 3: Giải nén tệp vừa tải xuống



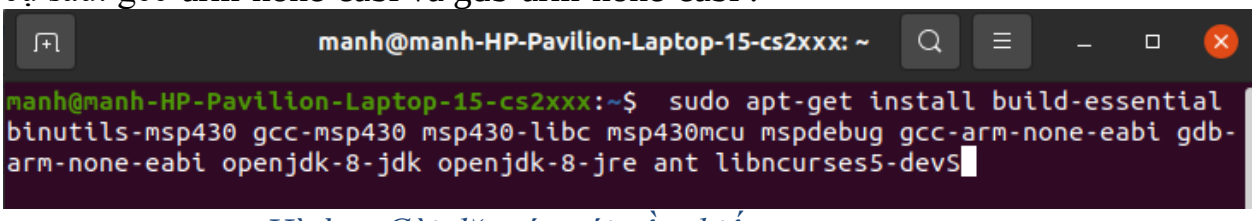
Hình 4: Giải nén file tải xuống

Bước 4: Đổi tên thư mục từ contiki-3.0 thành contiki



Hình 5: Đổi tên thư mục thành contiki

Bước 5: Cài đặt tất cả các gói cần thiết để biên dịch và chạy ContikiOS. Nếu bạn đang sử dụng nền tảng dựa trên MSP430 (chẳng hạn như Tmote), bạn chỉ cần các gói sau: binutils-msp430 , gcc-msp430 , msp430-libc , msp430mcu và mspdebug . Nếu bạn định viết mã cho các nền tảng dựa trên CC2538 (chẳng hạn như OpenMote), bạn cũng nên cài đặt các chuỗi công cụ sau: gcc-arm-none-eabi và gdb-arm-none-eabi .

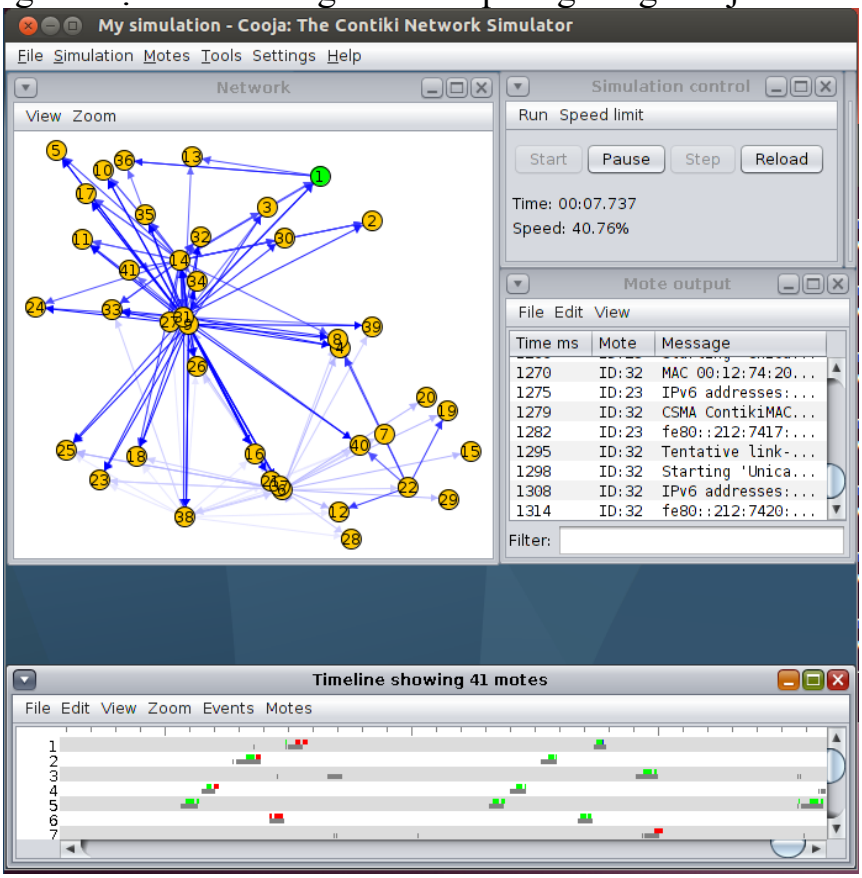


Hình 6: Cài đặt các gói cần thiết

Chương 2. Tổng quan về Cooja Simulation

2.1. Giới thiệu

Cooja là phần mềm mô phỏng hệ thống mạng được tích hợp trong hệ điều hành Contiki. Công cụ này cho phép người sử dụng thay đổi các thông số như vị trí, phạm vi kết nối, tỉ lệ truyền gói thành công,… Nhờ đó người sử dụng có thể mô phỏng và đánh giá kết quả một cách hiệu quả hơn. Dưới đây là giao diện của chương trình mô phỏng bằng Cooja:



Hình 7: Giao diện Cooja

Từ hình có thể nhận thấy giao diện của chương trình than thiện và dễ sử dụng, với một màn hình cho phép hiển thị các quá trình hoạt động của node, có khả năng thay đổi vị trí, phạm vi phủ sóng của mỗi node. Bên cạnh đó Cooja cung cấp một số các cửa sổ theo dõi sự kiện như Log listener, Radio listener cho phép người sử dụng tìm kiếm những sự kiện theo một số thông số nhất định, theo dõi sự giao tiếp giữa một số node cụ thể, ….Có thể nói, đây là một công cụ mô phỏng khá trực quan và dễ sử dụng, phục vụ tốt cho quá trình nghiên cứu, mô phỏng, đánh giá.

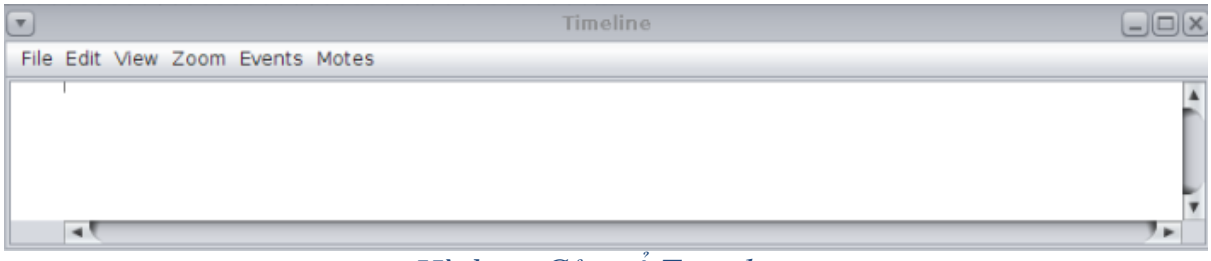
2.2. Công cụ hỗ trợ mô phỏng hệ điều hành Contiki

a. **Timeline window**:

Hiển thị các sự kiện và hoạt động của các node trong mô phỏng. Cửa sổ Timeline trong Cooja cho phép phân tích các sự kiện và hoạt động diễn ra trong mạng mô phỏng theo thời gian một các thuận tiện và trực quan. Có thể xem các

sự kiện như việc nhận và gửi gói tin, các hoạt động mạng như định tuyến và cập nhật bảng định tuyến,…

- Hiển thị sự kiện: Hiển thị các chỉ thị theo thời gian, bao gồm gửi và nhận gói tin, thay đổi trạng thái node và các hoạt động mạng khác
- Điều hướng và thu/phóng: Có thể di chuyển trên dải thời gian để xem các sự kiện trong khoảng thời gian cụ thể. Phóng to/thu nhỏ để tăng hoặc giảm độ chi tiết hiển thị
- Gán nhãn và màu sắc: Các sự kiện trên Timeline được gán nhãn để dễ dàng nhận biết và theo dõi. Sử dụng các màu sắc để phân biệt các sự kiện
- Có thể xem thông tin chi tiết về mỗi sự kiện bằng cách nhấp chuột nó trên timeline. Thông tin này có thể bao gồm thời gian, loại sự kiện, địa chỉ nguồn và đích, và các thông số khác tùy thuộc vào loại sự kiện



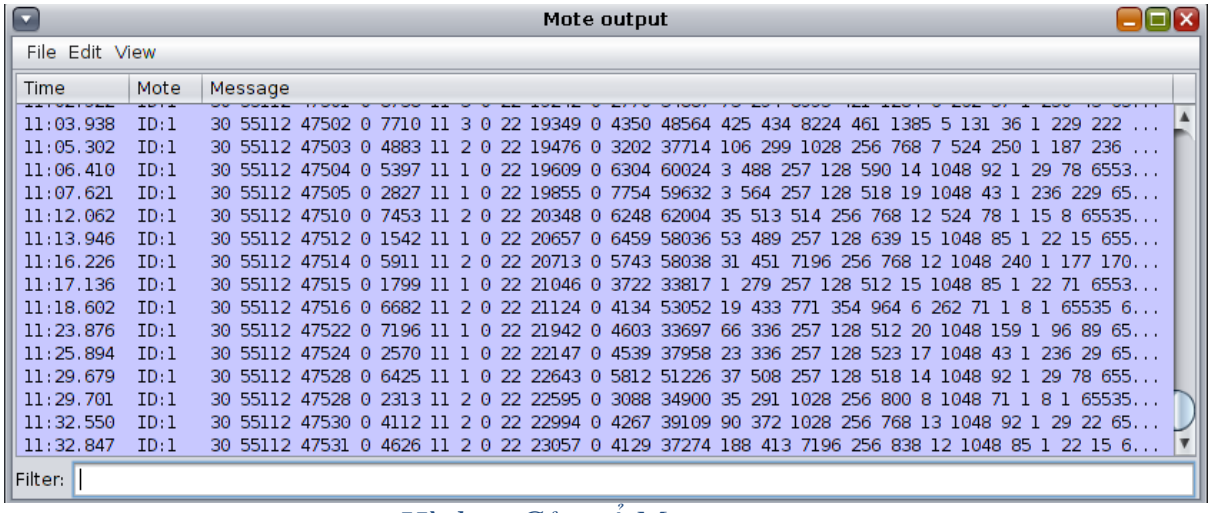
Hình 8: Cửa sổ Time line

b. **Mote Output:**

Khi chạy một mô phỏng trong Cooja, mỗi node trong mạng mô phỏng có thể in ra thông tin và dữ liệu đầu ra thông qua giao diện đầu ra chuẩn (Standard output) của chương trình node. Cửa sổ "mote output" hiện thị nội dung của giao diện đầu ra chuẩn này để có thể theo dõi và kiểm tra thông tin và dữ liệu được in ra từ các node

- Hiển thị đầu ra của node: nó hiển thị thông tin và dữ liệu được in ra từ các node trong mô phỏng. Có thể thấy các thông báo, lỗi, ghi chú và dữ liệu khác mà các node gửi ra
- Tự động cuộn và tìm kiếm: Khi có nhiều đầu ra, cửa sổ sẽ tự động cuộn xuống để hiển thị các dòng mới nhất. Có thể tìm kiếm thông tin cụ thể trong cửa sổ để nhanh chóng định vị các thông báo cần tìm
- Ghi lại đầu ra: Có thể ghi lại đầu ra của các node vào một tệp tin để theo dõi và phân tích sau này

Tool này hữu ích để gỡ lỗi, kiểm tra kết quả và hiểu rõ hơn về hành vi của các node trong quá trình mô phỏng



Hình 9: Cửa sổ Mote output

c. Network:

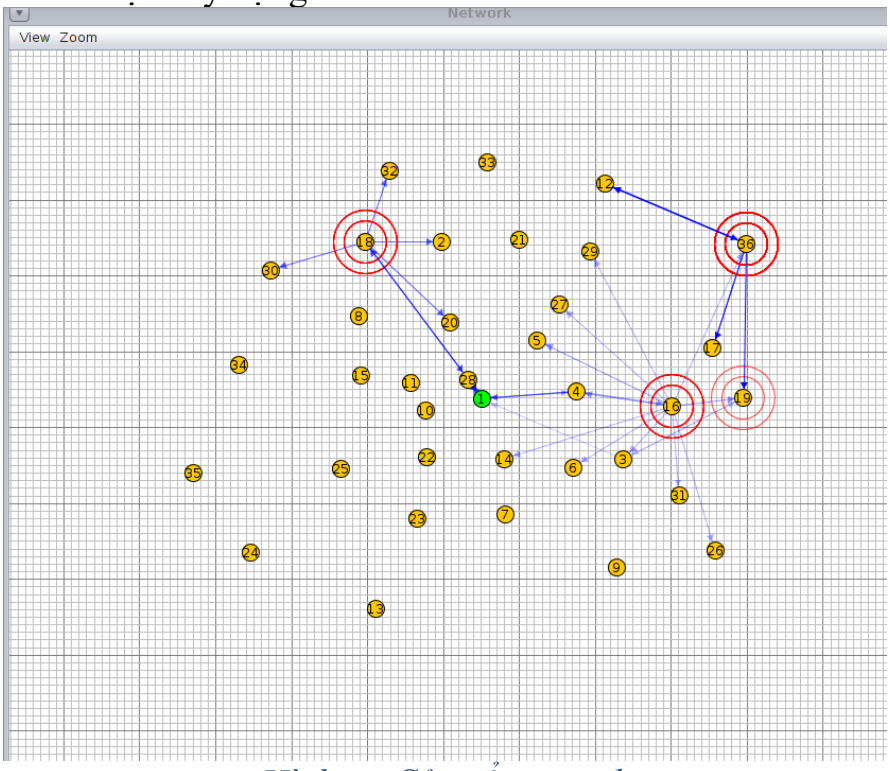
Là một cửa sổ hiển thị mạng cảm biến không dây được mô phỏng. Nó cung cấp một giao diện đồ họa để cấu hình, quan sát và quản lý các node và liên kết trong mạng mô phỏng. Cung cấp cái nhìn tổng quan về cấu trúc mạng, quản lý các thành phần của mạng trong quá trình mô phỏng.

- Hiển thị mạng: Hiển thị biểu đồ mạng với các node và liên kết giữa các node. Có thể thấy vị trí và tên của từng node trong mạng, cũng như các liên kết giữa chúng
- Quản lý node: Thêm, xóa và chỉnh sửa các liên kết giữa các node trong mạng mô phỏng. Hỗ trợ tính năng đặt vị trí và tên cho từng node, cũng như cấu hình các thuộc tính liên kết thiết lập giữa chúng
- Quản lý liên kiết: Tạo, xóa và chỉnh sửa liên kết giữa các node trong mạng, thiết lập các thông số và cấu hình cho liên kết, chẳng hạn như khoảng cách, tốc độ truyền, chất lượng kết nối
- Xem thông tin chi tiết: Xem thông tin chi tiết về từng node và liên kết bằng cách nhấp vào node trong cửa sổ. Thông tin này bao gồm địa chỉ, tên, thuộc tính, và trạng thái của node, cũng như các thông số và cấu hình của liên kết

Menu view cung cấp các tùy chọn hiển thị khác nhau. Một số tính năng mạnh được sử dụng nhiều:

- Mode IDs: Hiển thị ID của các node trong mô phỏng, cung cấp hình ảnh trực quan để nhận diện các node trong mạng
- Mote type: Sử dụng các màu sắc khác nhau để hiện thỉ sự khác biệt giữa các loại node khác nhau ví dụ sink có màu xanh và các client sẽ có màu vàng như hình dưới
- 10m background grid: cực kỳ hữu ích để mang lại cảm giác về quy mô bố cục mạng. Hiển thị một lưới các ô vuông 100m2 thay vì nền trống
- Radio traffic: Tùy chọn hữu ích trong việc tạo hoạt ảnh cho mạng khi mô phỏng đang chạy. Sử dụng tùy chọn này sẽ hiện thị việc trao đổi thông tin

thông báo giữa các node, cho phép quan sát việc lựa chọn parent và cách DODAG được xây dựng.

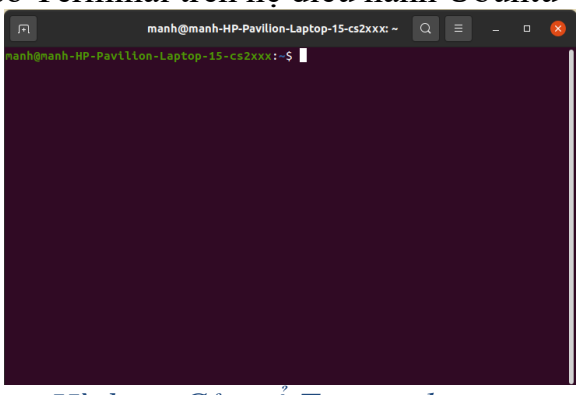


Hình 10: Cửa sổ network

2.3. Hướng dẫn cài đặt và chạy phần mềm

- Cài đặt hệ điều hành Ubuntu (đã có trong chương 1)
- Cài đặt Contiki trên hệ điều hành Ubuntu (đã có trong chương 1)
- Cài đặt và chạy phần mềm mô phỏng Cooja

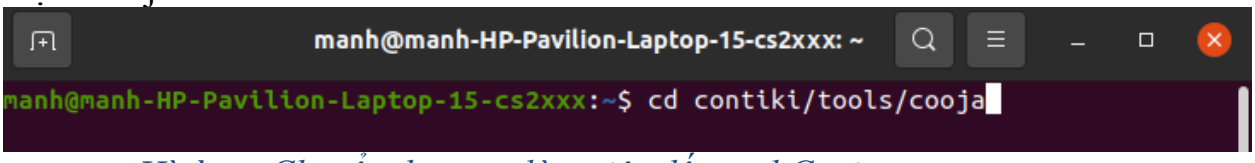
  Bước 1: Mở cửa sổ Terminal trên hệ điều hành Ubuntu



Hình 11: Cửa sổ Terminal

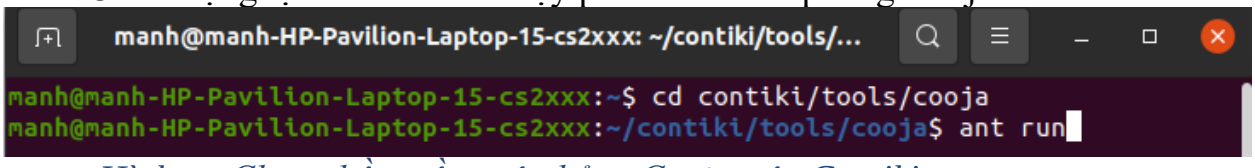
Bước 2: Sử dụng lệnh cd contiki/tools/cooja. Khi chạy lệnh "cd contiki/tools/cooja", tức là đang yêu cầu Terminal chuyển thư mục làm việc hiện tại thành "contiki/tools/cooja". Sau khi thực hiện lệnh này thành

công, các lệnh tiếp theo bạn chạy trong Terminal sẽ hoạt động trong thư mục "cooja".



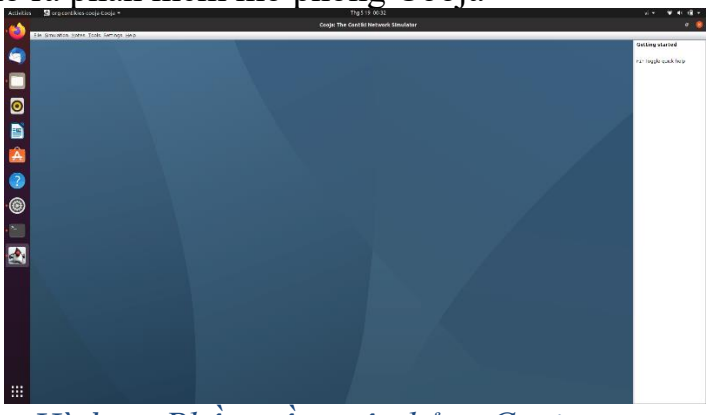
Hình 12: Chuyển thư mục làm việc đến tool Cooja

Bước 3: Sử dụng lệnh ant run để chạy phần mềm mô phỏng Cooja



Hình 13: Chạy phần mềm mô phỏng Cooja trên Contiki

Kết quả sẽ cho ra phần mềm mô phỏng Cooja



Hình 14: Phần mềm mô phỏng Cooja

Lưu ý nếu muốn sử dụng phần mềm Cooja của contiki-ng thì dùng các lệnh sau:

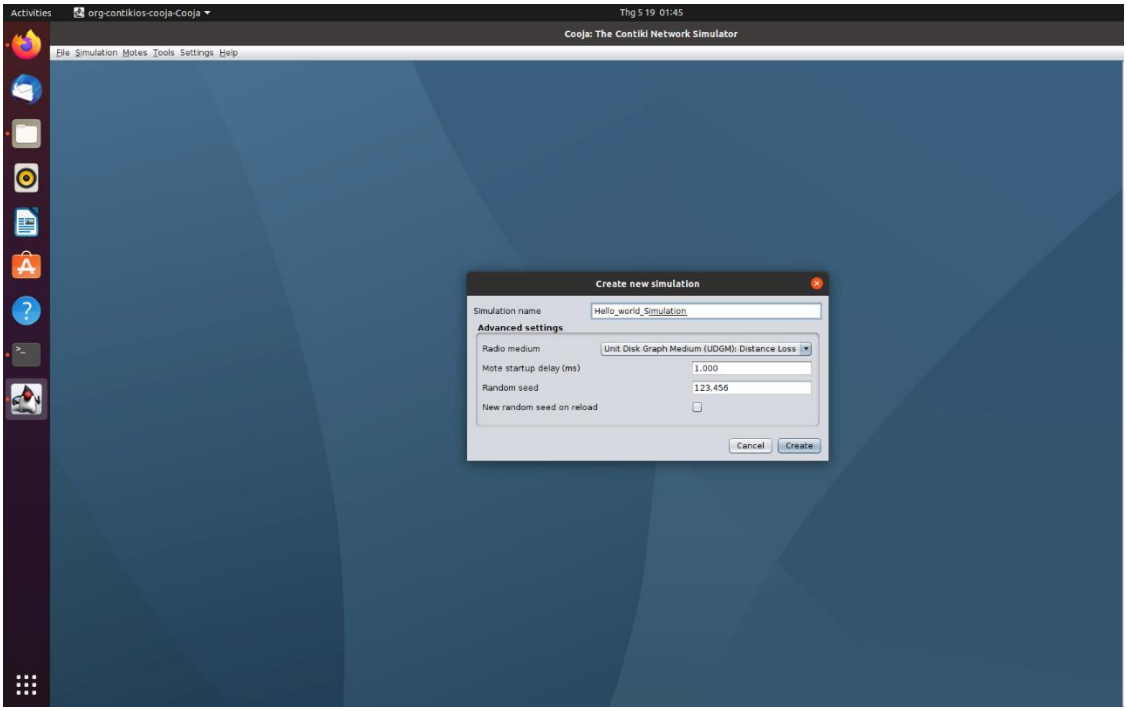
Hình 15: Chạy phần mềm mô phỏng Cooja trên contiki-ng

2.4. Ứng dụng Cooja Simulation và các ví dụ cơ bản

2.4.1 Mô phỏng Cooja cho người mới bắt đầu

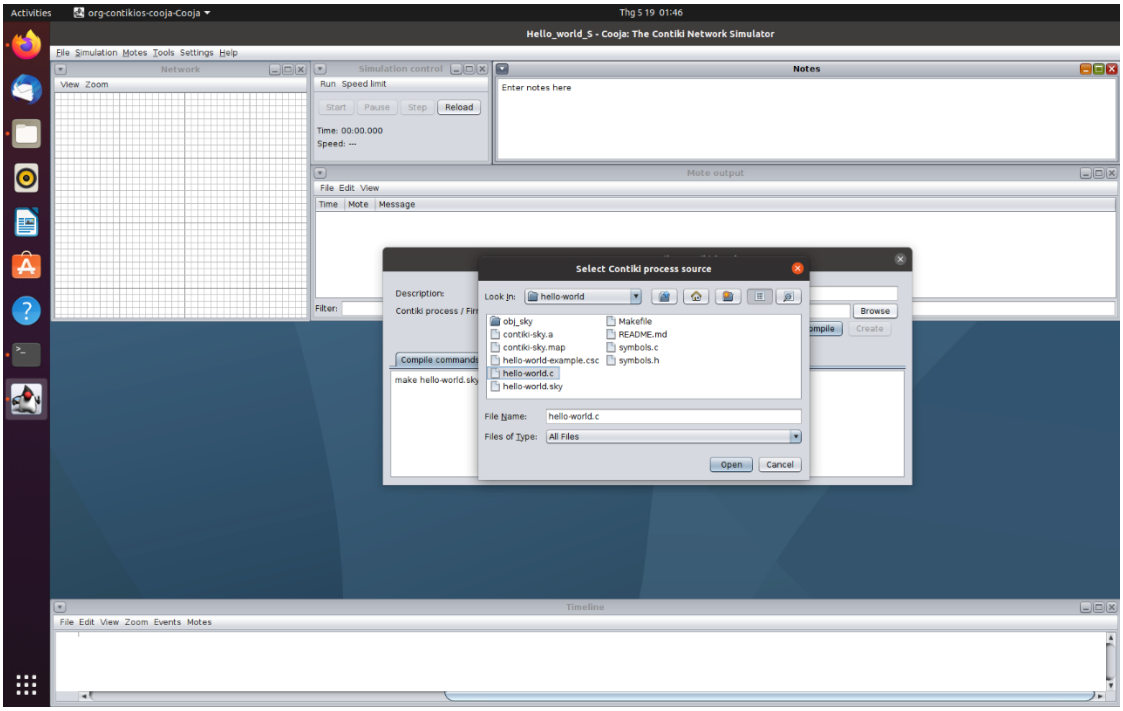
- Link tham khảo: #Contiki-os #Cooja Mô phỏng cho quy trình chào thế giới đơn giản #Technosilent - YouTube

Bước 1: Tạo file mô phỏng và đặt tên



Hình 16: Tạo tên thư mục

Bước 2: Chọn loại node là Sky Mote Type, sau đó chọn file hello-word.c vào Process để triển khai tiến trình trong mạng cảm biến không dây



Hình 17: Chọn loại node là Sky Mote Type

Phân tích đoạn code “Hello-world.c”

```
#include "contiki.h"

#include <stdio.h> /* For printf() */
/*---------------------------------------------------------------------
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*---------------------------------------------------------------------
PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();

  printf("Hello, world\n");

  PROCESS_END();
}
/*---------------------------------------------------------------------
```

Hình 18: Đoạn code Hello Word

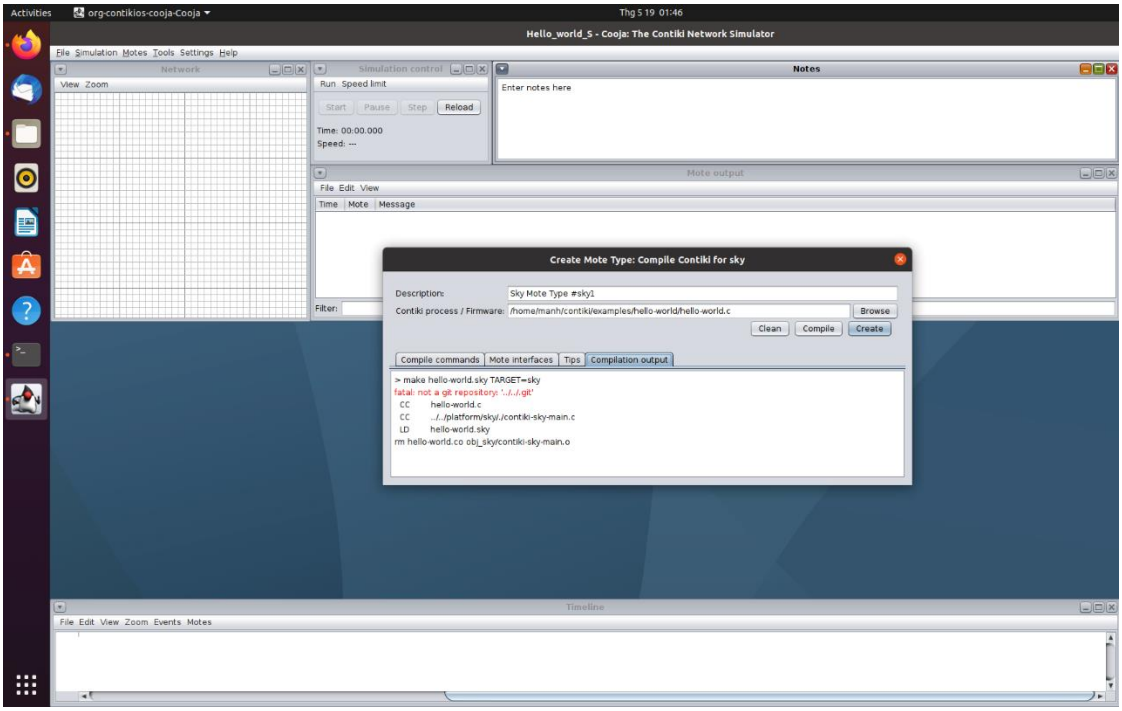
Đoạn code trên là một ứng dụng Contiki rất đơn giản, được sử dụng để hiển thị thông điệp "Hello, world" trên nền tảng Contiki.

Dưới đây là một giải thích từng phần trong đoạn code:

- Đoạn mã bắt đầu với các thông tin về quyền sở hữu và giấy phép của tệp mã nguồn.
- Tiếp theo, tệp mã nguồn bao gồm các tiêu đề và thư viện cần thiết. Trong trường hợp này, tệp "contiki.h" được bao gồm để sử dụng các chức năng của Contiki và "stdio.h" được bao gồm để sử dụng hàm printf().
- Tiếp theo là khai báo một quy trình Contiki được đặt tên là "hello_world_process". Quy trình này được sử dụng để thực thi mã chương trình chính.
- Trong phần "AUTOSTART_PROCESSES", quy trình "hello_world_process" được đăng ký để tự động khởi động khi Contiki được khởi chạy.
- Sau đó, khai báo quy trình "hello_world_process" bắt đầu.
- Trong quy trình này, PROCESS_BEGIN() được sử dụng để bắt đầu quy trình.
- Tiếp theo, hàm printf() được sử dụng để in ra thông điệp "Hello, world".
- Cuối cùng, PROCESS_END() được sử dụng để kết thúc quy trình.

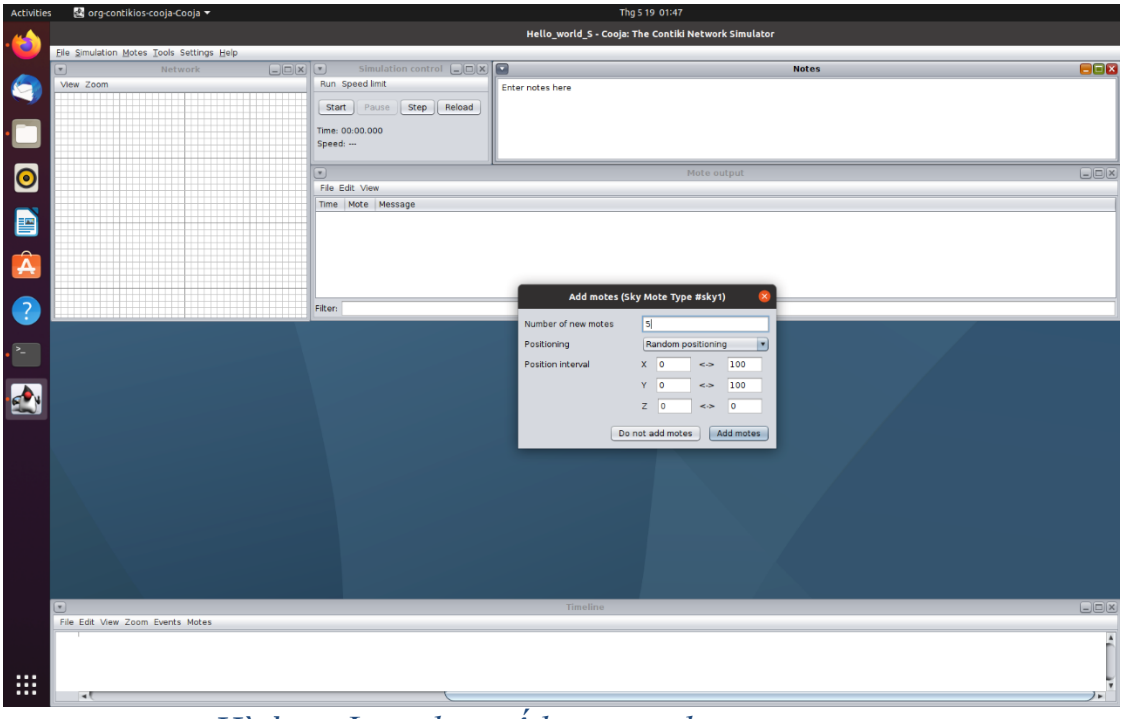
Tổng quan, đoạn mã trên chỉ đơn giản là in ra thông điệp "Hello, world" khi Contiki được khởi chạy.

Bước 3: Compile file .c vừa nhập để xem process có thể triển khai không, nếu có thì nút “Create” sẽ sáng lên



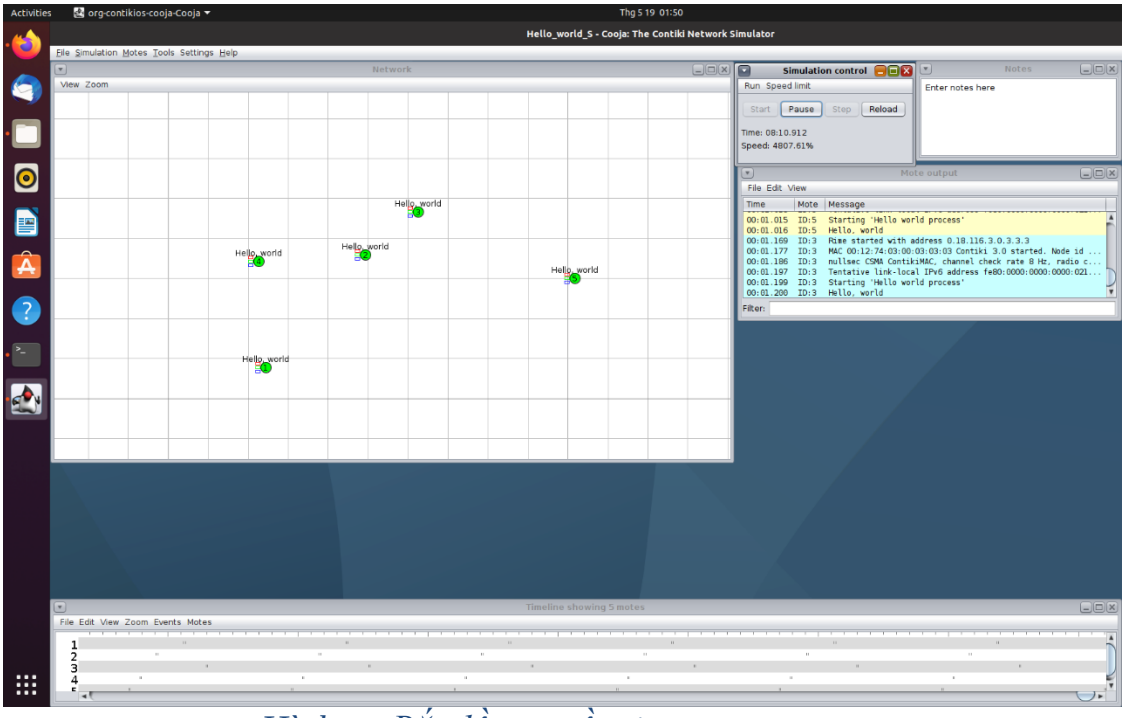
Hình 19: Complie file code

Bước 4: Nhập số node để mô phỏng mạng cảm biến không dây, trong đây ta chọn 5 node



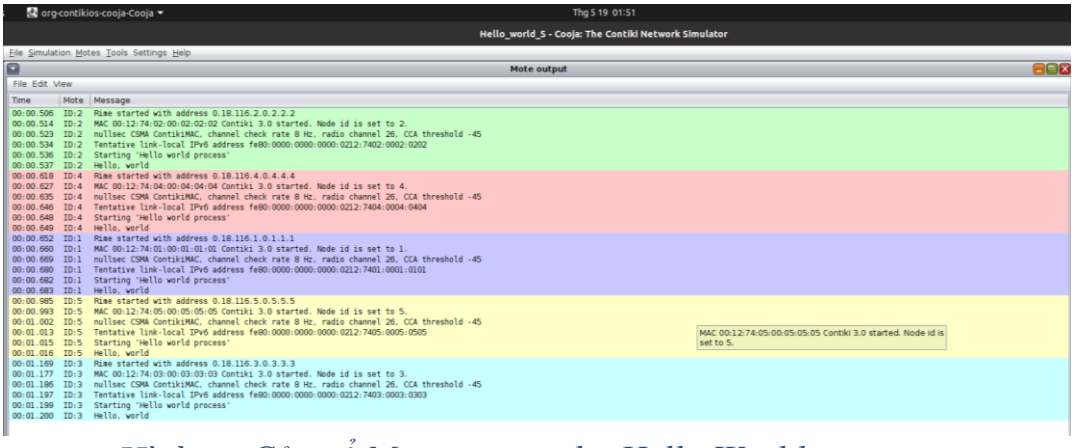
Hình 20: Lựa chọn số lượng node

Bước 5: Nhấn nút Start để bắt đầu quá trình truyền tin



Hình 21: Bắt đầu truyền tin

Bước 6: Đánh giá kết quả mô phỏng nhận được trong cửa sổ Output. Ta có thể thấy các node đều được xác định địa chỉ, đặt tên và cuối cùng là thực hiện quá trình truyền tin "Hello Word"

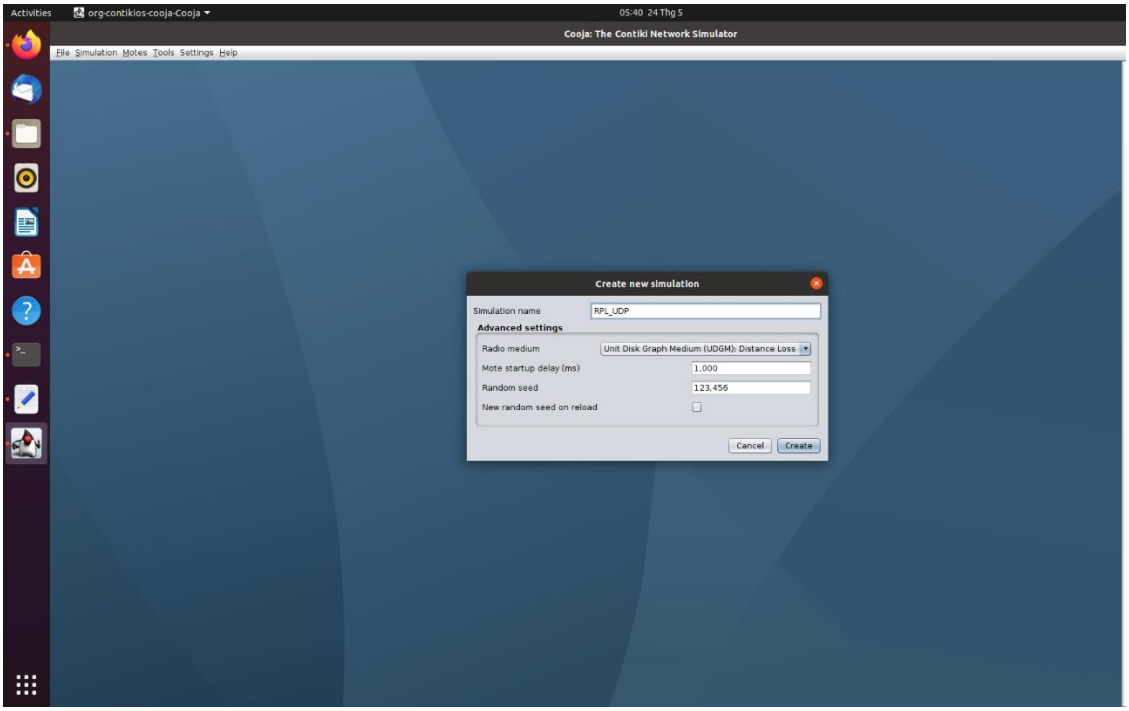


Hình 22: Cửa sổ Mote output cho Hello World

2.4.2 Mô phỏng RPL-UDP trong Cooja

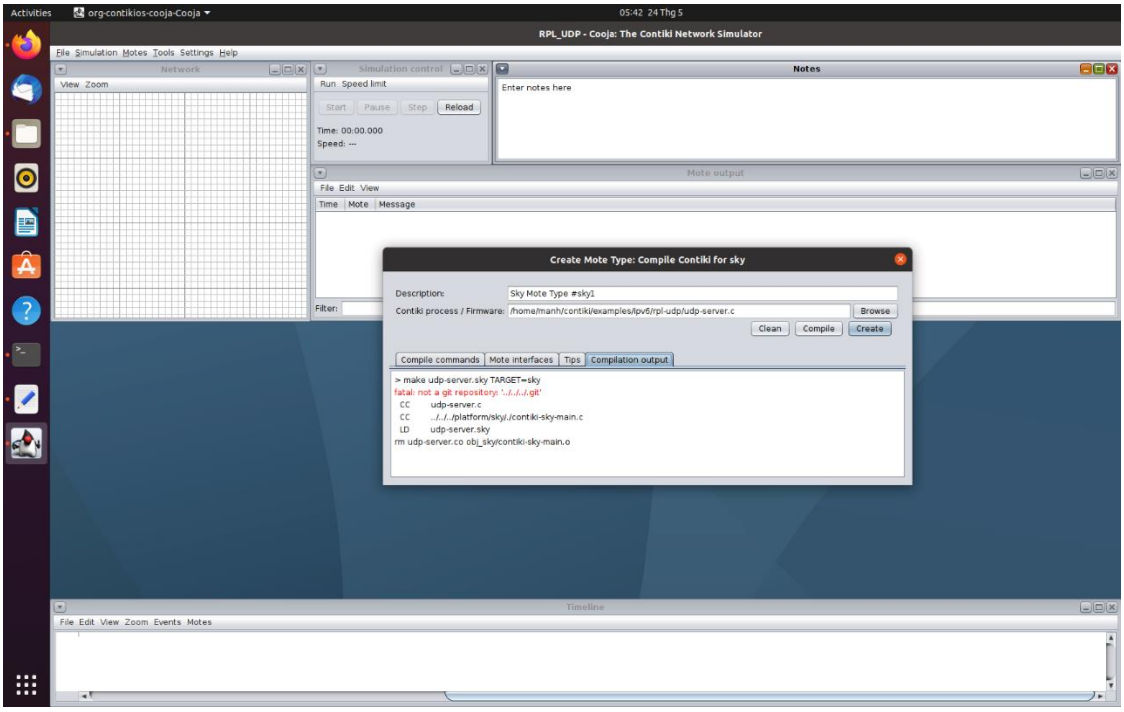
- Link tham khảo: [Running a RPL network in Cooja — Contiki-NG documentation]

Bước 1: Tạo file mô phỏng và đặt tên



Hình 23: Tạo tên cho file mô phỏng RPL_UDP

Bước 2: Chọn loại node là Sky Mote Type, sau đó chọn và Compile file "udp-server.c" trong mục Contiki process để triển khai tiến trình cho loại node server trong mạng cảm biến không dây. Nhiệm vụ chính của loại node này là lắng nghe và xử lý các gói tin UDP từ client và có thể gửi lại phản hồi



Hình 24: Chọn loại node Sky Mote Type và complile file udp.server

Phân tích đoạn code trong “udp-server.c”

```
#include "contiki.h"
#include "net/routing/routing.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"

#include "sys/log.h"
#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_INFO

#define WITH_SERVER_REPLY  1
#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678
```

Đầu tiên, tệp tin header cần thiết được import vào: **"contiki.h"** là một tệp tin header chính của hệ điều hành Contiki; **"net/routing/routing.h"** cung cấp các hàm và cấu trúc để quản lý định tuyến; **"net/netstack.h"** cung cấp giao diện chung cho ngăn xếp mạng; **"net/ipv6/simple-udp.h"** cung cấp các hàm và cấu trúc cho giao thức UDP. Tiếp theo, có một số định nghĩa tiền xử lý cho ghi log: **"#define LOG_MODULE"** xác định mô-đun log là "App"; **"#define LOG_LEVEL"** xác định cấp độ log là "LOG_LEVEL_INFO" (Thông tin). Các macro được định nghĩa: **WITH_SERVER_REPLY** xác định xem server có trả lời lại cho client hay không; **UDP_CLIENT_PORT** là cổng mà client sẽ sử dụng để gửi dữ liệu tới server; **UDP_SERVER_PORT** là cổng mà server lắng nghe để nhận dữ liệu từ client.

```
static struct simple_udp_connection udp_conn;

PROCESS(udp_server_process, "UDP server");
AUTOSTART_PROCESSES(&udp_server_process);
/*---------------------------------------------------------------------------*/
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)
{
  LOG_INFO("Received request '%.*s' from ", datalen, (char *) data);
  LOG_INFO_6ADDR(sender_addr);
  LOG_INFO_("\n");
#if WITH_SERVER_REPLY
  /* send back the same string to the client as an echo reply */
  LOG_INFO("Sending response.\n");
  simple_udp_sendto(&udp_conn, data, datalen, sender_addr);
#endif /* WITH_SERVER_REPLY */
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_server_process, ev, data)
{
  PROCESS_BEGIN();

  /* Initialize DAG root */
  NETSTACK_ROUTING.root_start();

  /* Initialize UDP connection */
  simple_udp_register(&udp_conn, UDP_SERVER_PORT, NULL,
                      UDP_CLIENT_PORT, udp_rx_callback);

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

Tiếp theo, có một kết nối UDP đơn giản được khởi tạo.

- Biến udp_conn được khai báo là một struct simple_udp_connection.
- Hàm simple_udp_register được sử dụng để đăng ký kết nối UDP này.
- Đăng ký kết nối UDP bao gồm cổng UDP của server, cổng UDP của client và một con trỏ tới hàm callback (udp_rx_callback) sẽ được gọi khi có dữ liệu được nhận từ client.

Tiếp theo, có một quy trình (PROCESS) được định nghĩa cho udp_server.

- Quy trình bắt đầu bằng PROCESS_BEGIN() và kết thúc bằng PROCESS_END().

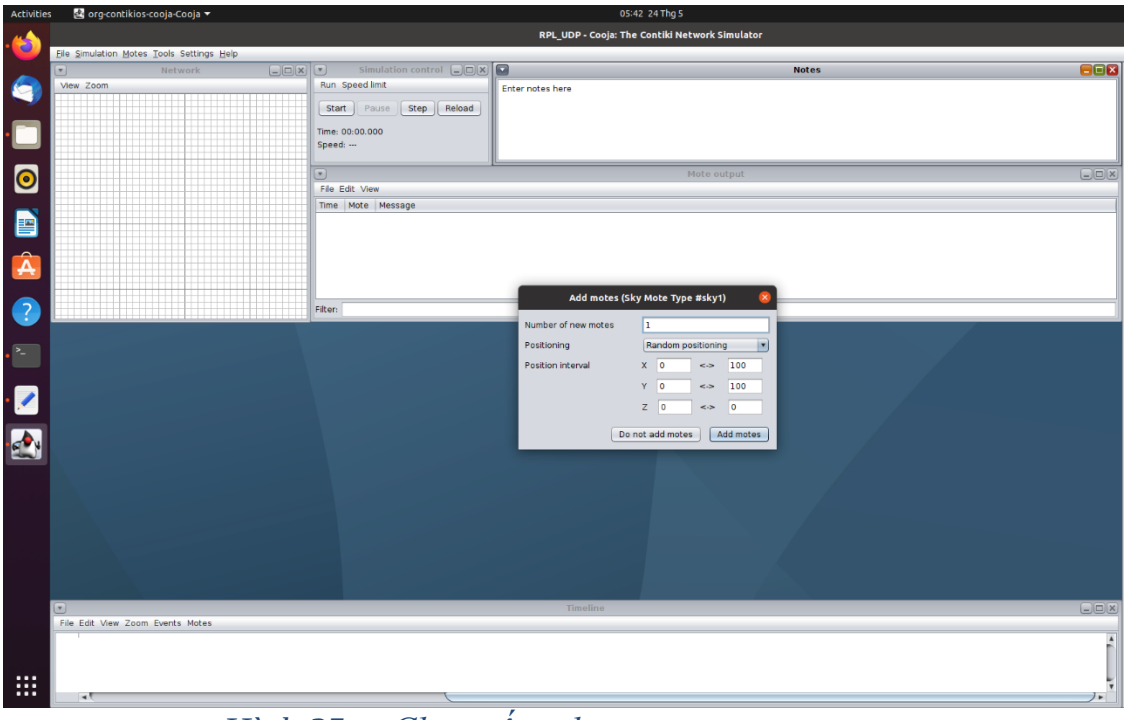
- Trong quy trình này, đầu tiên DAG root được khởi tạo bằng cách gọi NETSTACK_ROUTING.root_start().
- Sau đó, kết nối UDP được khởi tạo bằng cách gọi simple_udp_register().
- Cuối cùng, quy trình kết thúc.

Cuối cùng, có một hàm callback udp_rx_callback được định nghĩa.

- Hàm này được gọi khi có dữ liệu được nhận từ client.
- Nó in ra nội dung của gói tin nhận được và địa chỉ của client gửi gói tin đó.
- Nếu WITH_SERVER_REPLY được định nghĩa, server sẽ gửi lại cùng một chuỗi dữ liệu cho client bằng cách gọi simple_udp_sendto().

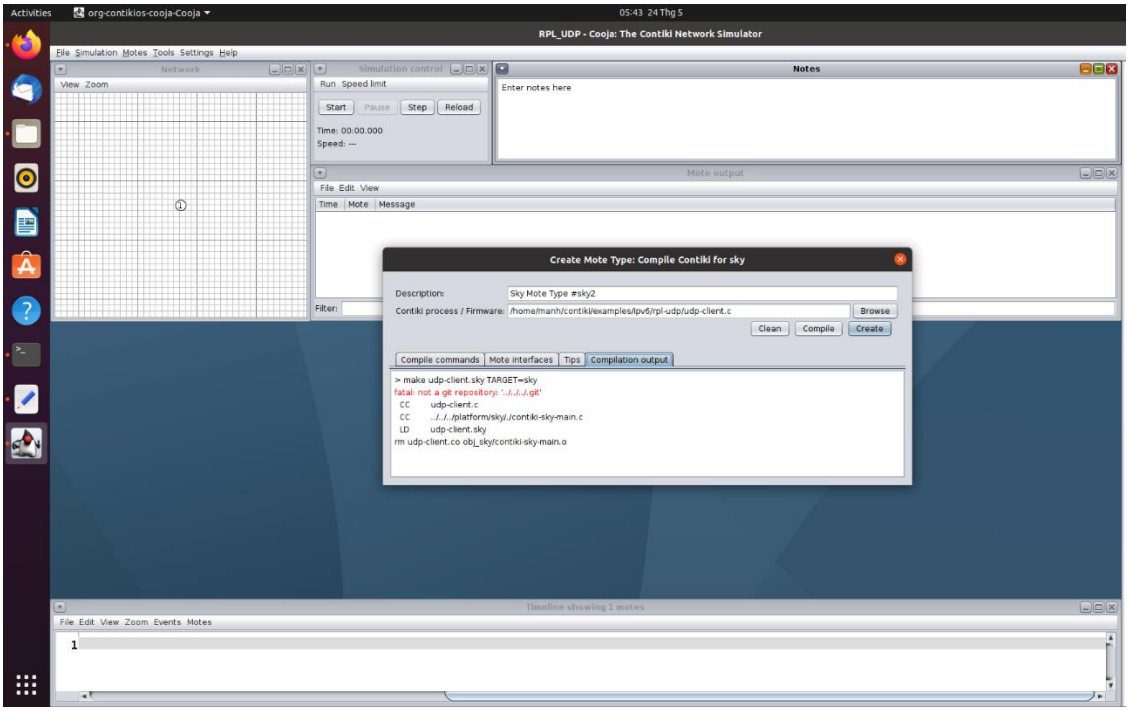
Tổng kết lại, đoạn mã trên đề cập đến việc triển khai một server UDP đơn giản trong hệ điều hành Contiki. Nó đăng ký một kết nối UDP và lắng nghe dữ liệu từ client, sau đó gửi lại cùng một dữ liệu cho client (nếu WITH_SERVER_REPLY được định nghĩa).

Bước 3: Chọn số node loại udp-server là 1 node



Hình 25 25:Chọn số node server

Bước 4: Chọn loại node là Sky Mote Type, sau đó chọn và Compile file "udp-client.c" trong mục Contiki process để triển khai tiến trình cho loại node client trong mạng cảm biến không dây. Nhiệm vụ chính của loại node này là sẽ gửi các gói tin UDP chứa dữ liệu "hello" đến nút gốc của mạng.



Hình 26:Chọn loại node Sky và complile file udp.client

Phân tích đoạn code trong “udp-client.c”

```
#include "contiki.h"
#include "net/routing/routing.h"
#include "random.h"
#include "net/netstack.h"
#include "net/ipv6/simple-udp.h"
#include <stdint.h>
#include <inttypes.h>

#include "sys/log.h"
#define LOG_MODULE "App"
#define LOG_LEVEL LOG_LEVEL_INFO

#define WITH_SERVER_REPLY  1
#define UDP_CLIENT_PORT 8765
#define UDP_SERVER_PORT 5678

#define SEND_INTERVAL           (60 * CLOCK_SECOND)

static struct simple_udp_connection udp_conn;
static uint32_t rx_count = 0;
```

Đầu tiên, các thư viện cần thiết được import: **contiki.h:** Thư viện chính của Contiki OS; **net/routing/routing.h:** Thư viện định tuyến mạng; **random.h:** Thư viện để tạo số ngẫu nhiên; **net/netstack.h:** Thư viện giao tiếp mạng.; **net/ipv6/simple-udp.h:** Thư viện UDP đơn giản; **stdint.h và inttypes.h:** Thư viện để định nghĩa kiểu dữ liệu số nguyên.

Định nghĩa các hằng số và biến: **LOG_MODULE và LOG_LEVEL:** Định nghĩa module và mức độ log; **WITH_SERVER_REPLY:** Xác định liệu ứng dụng có nhận phản hồi từ máy chủ hay không; **UDP_CLIENT_PORT và UDP_SERVER_PORT:** Cổng UDP của ứng dụng khách và máy chủ; **SEND_INTERVAL:** Khoảng thời gian giữa các lần gửi yêu cầu.

```
PROCESS(udp_client_process, "UDP client");
AUTOSTART_PROCESSES(&udp_client_process);
/*---------------------------------------------------------------------------*/
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)
{

  LOG_INFO("Received response '%.*s' from ", datalen, (char *) data);
  LOG_INFO_6ADDR(sender_addr);
#if LLSEC802154_CONF_ENABLED
  LOG_INFO_(" LLSEC LV:%d", uipbuf_get_attr(UIPBUF_ATTR_LLSEC_LEVEL));
#endif
  LOG_INFO_("\n");
  rx_count++;
}
/*---------------------------------------------------------------------------*/
```

Định nghĩa hàm callback khi nhận được gói tin UDP:

udp_rx_callback: Hàm này được gọi khi ứng dụng nhận được gói tin UDP. Nó in ra thông tin về gói tin nhận được và tăng giá trị của biến rx_count.

Định nghĩa quá trình chính:

udp_client_process: Quá trình chính của ứng dụng khách UDP. Trong quá trình này, ứng dụng sẽ gửi yêu cầu UDP đến một địa chỉ IP và cổng UDP cụ thể.

AUTOSTART_PROCESSES: Xác định quá trình tự động bắt đầu khi hệ thống khởi động.

Định nghĩa các biến và thiết lập ban đầu:

periodic_timer: Một đối tượng etimer được sử dụng để đặt thời gian giữa các lần gửi yêu cầu.

str: Một mảng ký tự để lưu dữ liệu yêu cầu gửi đi.

dest_ipaddr: Địa chỉ IP đích.

```
PROCESS_THREAD(udp_client_process, ev, data)
{
  static struct etimer periodic_timer;
  static char str[32];
  uip_ipaddr_t dest_ipaddr;
  static uint32_t tx_count;
  static uint32_t missed_tx_count;

  PROCESS_BEGIN();

  /* Initialize UDP connection */
  simple_udp_register(&udp_conn, UDP_CLIENT_PORT, NULL,
                      UDP_SERVER_PORT, udp_rx_callback);

  etimer_set(&periodic_timer, random_rand() % SEND_INTERVAL);
  while(1) {
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));

    if(NETSTACK_ROUTING.node_is_reachable() &&
        NETSTACK_ROUTING.get_root_ipaddr(&dest_ipaddr)) {

      /* Print statistics every 10th TX */
      if(tx_count % 10 == 0) {
        LOG_INFO("Tx/Rx/MissedTx: %" PRIu32 "/%" PRIu32 "/%" PRIu32 "\n",
                 tx_count, rx_count, missed_tx_count);
      }

      /* Send to DAG root */
      LOG_INFO("Sending request %"PRIu32" to ", tx_count);
      LOG_INFO_6ADDR(&dest_ipaddr);
      LOG_INFO_("\n");
      snprintf(str, sizeof(str), "hello %" PRIu32 "", tx_count);
      simple_udp_sendto(&udp_conn, str, strlen(str), &dest_ipaddr);
      tx_count++;
    } else {
      tx_count++;
    } else {
      LOG_INFO("Not reachable yet\n");
      if(tx_count > 0) {
        missed_tx_count++;
      }
    }

    /* Add some jitter */
    etimer_set(&periodic_timer, SEND_INTERVAL
      - CLOCK_SECOND + (random_rand() % (2 * CLOCK_SECOND)));
  }

  PROCESS_END();
}
```

Bắt đầu quá trình chính:

PROCESS_BEGIN(): Khởi tạo quá trình.

Đăng ký kết nối UDP:

simple_udp_register(): Đăng ký kết nối UDP với cổng UDP của ứng dụng khách và hàm **callback udp_rx_callback()** khi nhận được gói tin.

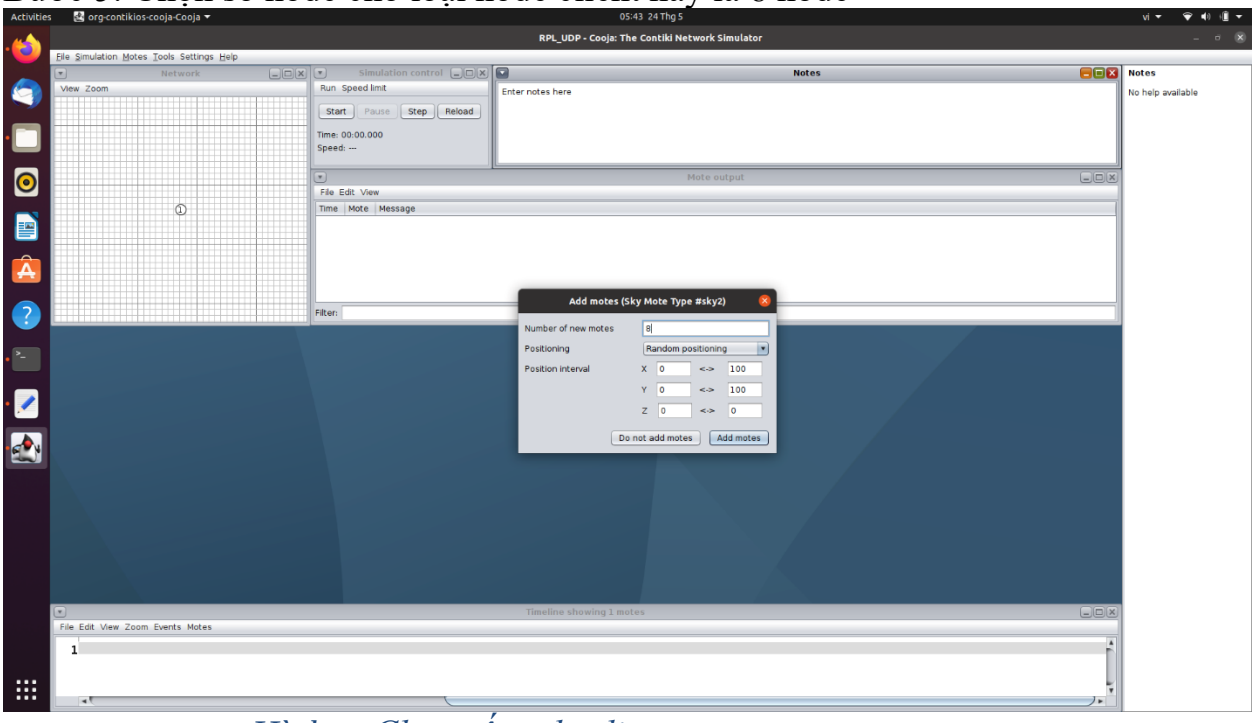
Đặt thời gian cho periodic_timer và chờ đến khi hết thời gian:

etimer_set(): Đặt thời gian cho đối tượng periodic_timer với một giá trị ngẫu nhiên trong khoảng **SEND_INTERVAL.**

PROCESS_WAIT_EVENT_UNTIL(): Chờ đến khi periodic_timer hết thời gian.

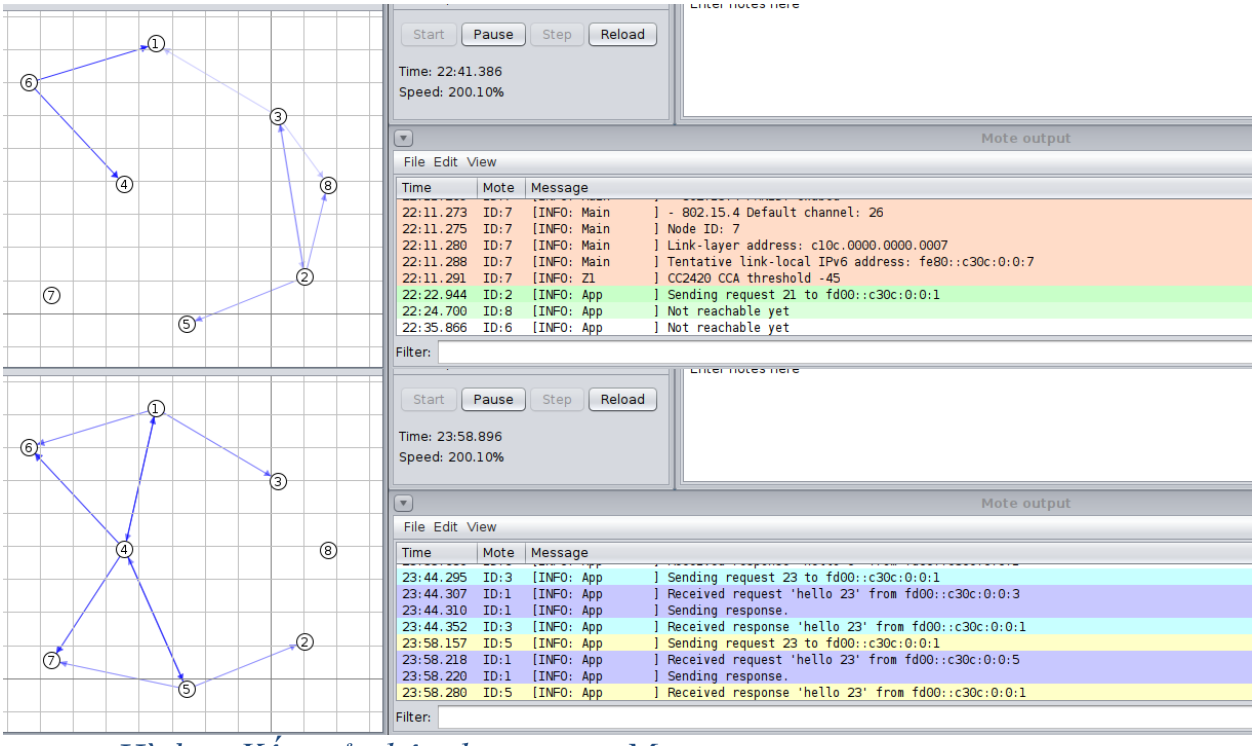
Kiểm tra xem nút hiện tại có thể kết nối và lấy địa chỉ IP của nút gốc:
NETSTACK_ROUTING.node_is_reachable(): Kiểm tra xem nút hiện tại có thể kết nối.
NETSTACK_ROUTING.get_root_ipaddr(): Lấy địa chỉ IP của nút gốc.
Gửi yêu cầu UDP đến nút gốc:
In ra thông tin thống kê mỗi khi gửi 10 yêu cầu.
Gửi yêu cầu UDP bằng cách sử dụng simple_udp_sendto() với dữ liệu là chuỗi "hello" và số lần yêu cầu tx_count.
Tăng giá trị của tx_count.
Xử lý khi không kết nối được:
In ra thông báo "Not reachable yet".
Nếu đã gửi ít nhất một yêu cầu, tăng giá trị của missed_tx_count.
Tạo sự biến đổi trong khoảng thời gian gửi yêu cầu:
Tạo sự biến đổi ngẫu nhiên trong khoảng SEND_INTERVAL bằng cách sử dụng etimer_set() và random_rand().
Kết thúc quá trình:
PROCESS_END(): Kết thúc quá trình chính của ứng dụng.

Bước 5: Chọn số node cho loại node client này là 8 node



Hình 27:Chọn số node client

Bước 6: Nhấn nút Start để bắt đầu quá trình mô phỏng và đánh giá kết quả nhận được trong cửa sổ Mote Output



Hình 28:Kết quả nhận được trong Mote output

Chương 3. Giao thức RPL và ContikiRPL

Trong bài báo, để quản lý tài nguyên cho mạng công suất và tổn thất thấp, thuật toán RPL đã được xây dựng và phát triển để tối ưu hóa các tài nguyên của các node trong mạng cảm biến.

3.1. Tổng quan RPL

RPL-Routing Protocol for Low Power and Lossy Network là giao thức định tuyến cho mạng tổn hao năng lượng thấp nói chung và IoT nói riêng. Dự thảo đầu tiên về RPL được IETF đưa ra vào 8/2009. Hiện nay, giao thức RPL đang trong quá trình hoàn thiện, với mục tiêu phát triển thành một chuẩn định tuyến trong tương lai. Quá trình phát triển RPL cũng nhận được sự quan tâm, đóng góp của nhiều tổ chức, cá nhân đến từ những tổ chức nghiên cứu khoa học, các trường đại học, viện nghiên cứu trên toàn thế giới.

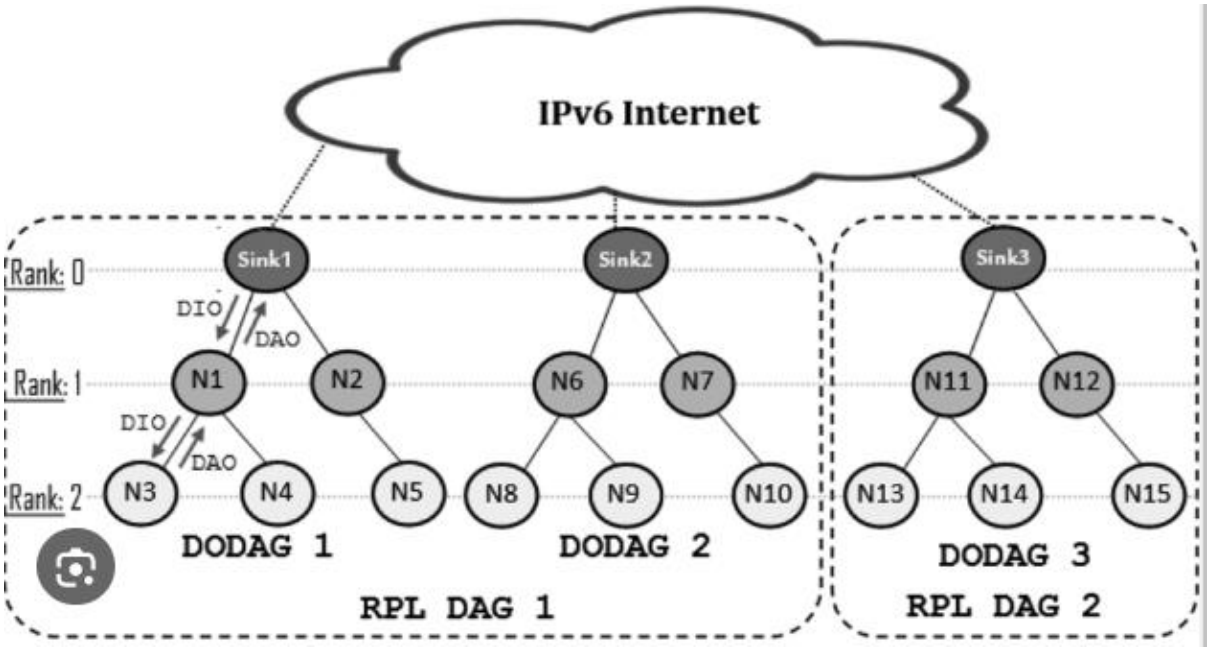
RPL được phát triển trên nền IPv6 nhằm đạt được những yêu cầu về định tuyến sau:

- Có khả năng mở rộng (số lượng, quy mô)
- Định tuyến dựa trên những thông số bị giới hạn như mức năng lượng, dung lượng bộ nhớ
- Hỗ trợ tính di động
- Hỗ trợ khả năng tự cấu hình và cấu hình từ bên ngoài
- Hỗ trợ truyền tin multicast và anycast
- Hỗ trợ các luồng thông tin định hướng đến một node trong mạng
- Có cấu trúc mạng động
- Hỗ trợ định tuyến theo nhiều mectric khác nhau
- Có khả năng hội tụ về thời gian. Giao thức định tuyến hội tụ về thời gian khi nó đáp ứng được những mức thời gian trễ cụ thể tương ứng với những điều kiện xác định
- Có khả năng quản lý. Khi một node tham gia vào mạng, node đó phải tự động cấu hình và tham gia vào mạng mà không cần sự can thiệp của con người
- Hỗ trợ truyền gói theo mức độ ưu tiên và có độ tin cậy cao
- Hỗ trợ các phương thức bảo mật

3.1.1. Khái niệm, thuật ngữ sử dụng trong RPL

- RPL xây dựng và sử dụng các DAG (Directed Acyclic Graph) trong mạng để thực hiện quá trình định tuyến. Trong đó, DAG là một cấu trúc mạng mà mọi liên kết giữa các node trong DAG đều có hướng nhất định, hướng về 1 DAG root và đảm bảo không tạo ra các vòng lặp trong DAG
- DAG root: một node trong DAG, có chức năng tập trung và xử lý dữ liệu từ các node khác trong DAG gửi đến. Mọi tuyến liên kết trong DAG đều hướng về và kết thúc tạo DAG root.
- DODAG (Destination-Oriented DAG) là một DAG chỉ có một node đích, vì dụ như chỉ có một DAG root (trong trường hợp này gọi là DODAG root)

- DODAG root: một DODAG root là DAG root của một DODAG, DODAG root có thể là router biên của một DODAG
- Virtual DODAG root: Một DODAG root ảo là kết quả của hai hoặc nhiều bộ định tuyến RPL, điều phối để đồng bộ hóa DODAG state và diễn ra như thể chúng là một DODAG root duy nhất
- DODAGID (DODAG Identifiter): mã nhận dạng của mỗi DODAG root trong mạng. Tất cả các node trong mạng đều lưu DODAGID của DODAG mà nó là thành viên

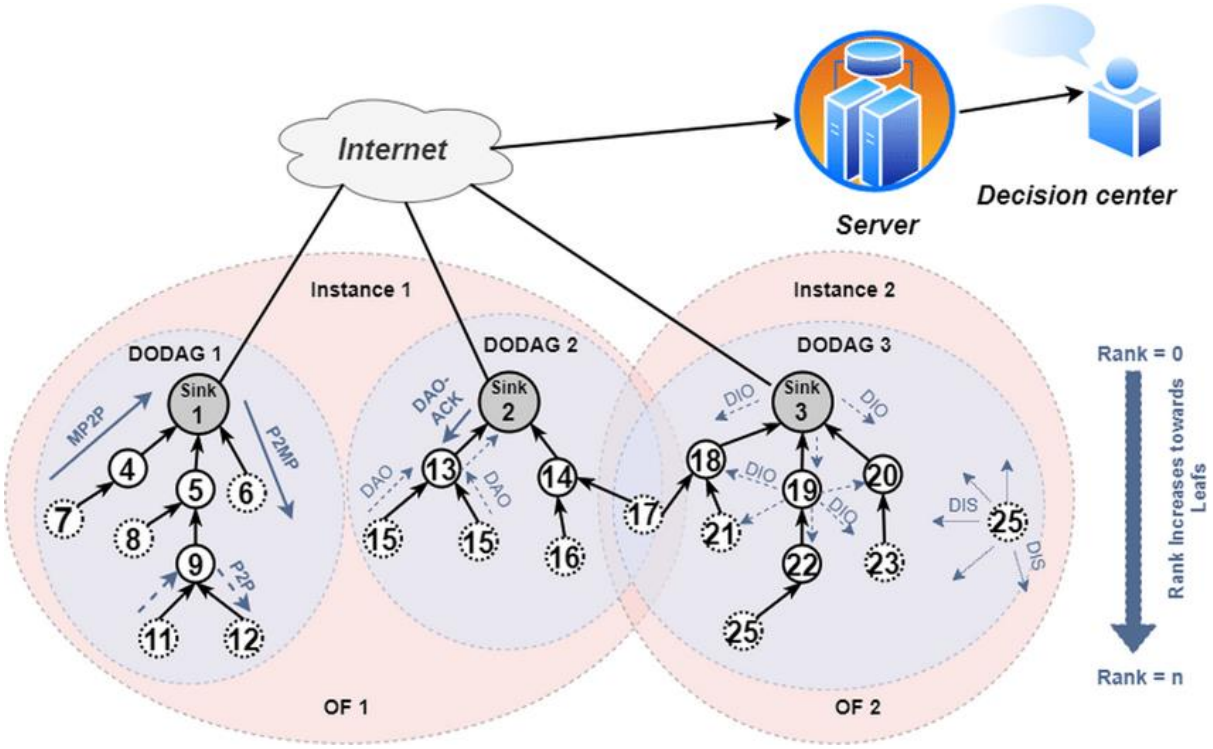


Hình 29:RPL DODAG

Hình 2.1. Mô hình RPL DODAG

- DODAG rank: là thông số cho biết vị trí tương đối của node so với DODAG root. Những node càng xa DODAG root thì có rank càng cao. Rank của node có thể được tính thông qua khoảng cách hình học giữa node và DODAG root, hoặc có thể tính qua các hàm chức năng khác. Trong RPL, DODAG root luôn có rank bằng 1. Rank được dung để đánh giá mối quan hệ parent-sibling-children giữa các node trong cùng một DODAG, từ đó tránh các vòng lặp có thể xảy ra khi truyền gói đến DODAG root
- DODAG parent: trong cùng một DODAG, node A được gọi là parent của node B khi A có khả năng kết nối trực tiếp đến B và A có rank thấp hơn B. Khi đó A có thể đóng vai trò là next-hop của B trong quá trình truyền gói về DODAG root và B là một node children của A.
- DODAG sibling: node A là một sibling của node B trong một DODAG nếu chúng có cùng rank trong DODAG đó.
- Object Function (OF): là hàm chức năng cung cấp các phương thức cho phép một node lựa chọn được DODAG phù hợp, tính toán rank và lựa chọn các parent trong DODAG. Các OF được thiết kế theo những quy tắc cụ thể tùy theo mục đích và phương thức định tuyến

- Object Code Point (OCP): là mã nhận dạng tương ứng với một OF cụ thể. OCP được sử dụng trong quá trình kiểm tra các bản tin trong mạng và quá trình tìm kiếm những DODAG phù hợp.
- RPL Instance : là một tập hợp gồm một hoặc nhiều DODAG có cơ chế định tuyến giống nhau. Mỗi Instance chỉ sử dụng một OF duy nhất để xây dựng cấu trúc mạng
- RPL Instance ID: mã nhận dạng của các Instance, tương ứng với các OF cụ thể
- DODAG Sequence Number: là một bộ đếm tuần tự được sử dụng trong quá trình sửa chữa và làm mới, sequence number được tăng lên một đơn vị và quảng bá tới các node khác trong mạng



Hình 30: DODAG Sequence Number

- DODAG Interaction: mỗi DODAG ID và DODAG sequence number cho phép xác định một DODAG Interaction. Khái niệm này cho phép mỗi node trong mạng phân biệt các DODAG mà node đã tham gia với một DODAG mà node chưa từng là thành viên. Đồng thời khái niệm này cũng cho phép cơ chế tránh các vòng lặp hoạt động hiệu quả hơn
- RPL sử dụng ba loại bản tin điều khiển gồm DODAG Information Solicitation (DIS), DODAG Information Object (DIO), Destination Advertisement Object (DAO) để quảng bá các thông tin định tuyến trong mạng.

3.1.2. RPL Identifiers

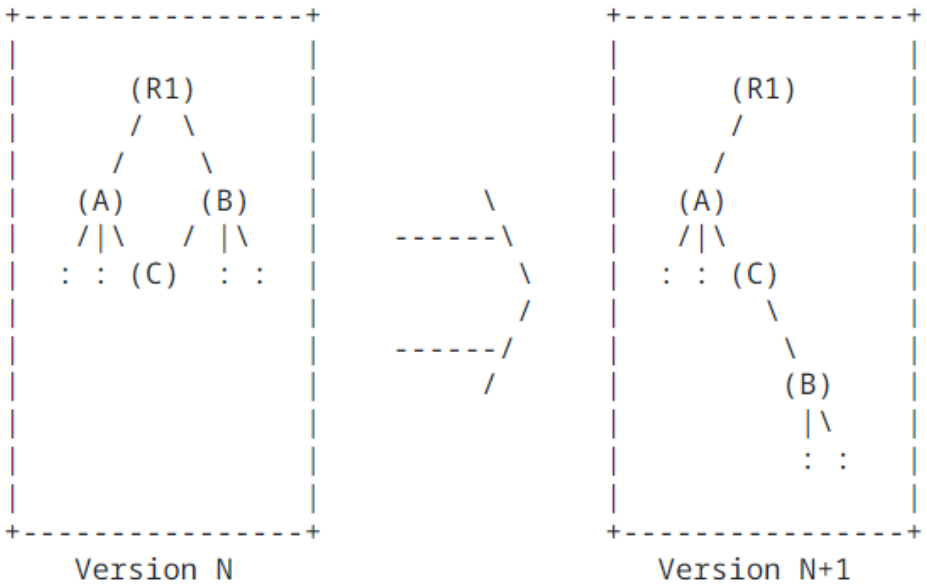
RPL sử dụng 4 giá trị để định danh và duy trì cấu trúc:

RPLInstanceID: một RPLInstanceID xác định một tập hoặc một DODAG. một mạng có thể có nhiều RPLInstanceID, mỗi trong số đó xác định một bộ DODAG độc lập, có thể được tối ưu hóa cho các hàm mục tiêu (OF) và/hoặc ứng dụng khác nhau. Tập hợp các DODAG được xác định bởi RPLInstanceId được gọi là RPL Instance. Tất cả các DODAG trong cùng 1 RPL Instance sử dụng chung một hàm mục tiêu (OF).

DODAGID: Phạm vi của một DODAGID là một RPL Instance. Sự kết hợp giữa RPLInstanceID và DODAGID xác định duy nhất một DODAG đơn trong mạng. Một RPL Instance có thể có nhiều DODAG, và mỗi DODAG có một DODAGID duy nhất

DODAGVersionNumber: Phạm vi của DODAGVersionNumber là trong một DODAG. DODAG đôi khi xây dựng lại từ DODAG root, bằng cách tăng DODAGVersionNumber. Sự kết hợp giữa DODAGVersionNumber, DODAGID, RPLInstanceID xác định một DODAG version duy nhất

Rank: Phạm vi của rank là trong một DODAG version. Xếp hạng thiết lập một phần thứ tự đối với Phiên bản DODAG, xác định các vị trí nút riêng lẻ đối với gốc DODAG.



Hình 31: DODAGVersionNumber

3.1.3. Các loại bản tin trong định tuyến RPL

a. Bản tin DIO

DIO là bản tin được tạo ra tại các DODAG root, mang những thông tin định tuyến của DODAG như instance, rank, metric,... DIO được sử dụng để quảng bá các thông tin định tuyến của một DODAG xác định trong mạng, phục vụ quá trình xây dựng DODAG và định tuyến upward. Quá trình nhận và xử lý bản tin DIO

cho phép một node nhận diện và tham gia vào DODAG phù hợp. Từ đó lựa chọn các parent, xác định các thông số cấu hình và tiếp tục quảng bá thông tin DODAG đến các node khác trong mạng
Cấu trúc bản tin DIO gồm 2 phần chính: các trường điều khiển và các sub-option:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| RPLInstanceID |Version Number |             Rank              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|G|0| MOP | Prf |     DTSN      |     Flags     |   Reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                            DODAGID                            +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Option(s)...
+-+-+-+-+-+-+-+-+
```

Hình 32:Bản tin DIO

Cờ G (ground): cho phép xác định khả năng kết nối đến một grounded DODAG ảu một node gửi DIO. Nếu cơ G = 0, node được gửi DIO là thành viên của một floating DODAG. Ngược lại, nếu cờ này khác 0 cho biết node gửi DIO là thành viên của một grounded DODAG
prf (DODAG preference): sử dụng 3 bit kiểu số nguyên dương cho biết độ ưu tiên giữa các DODAG root trong cùng một instance. Miền giá trị của prf từ 0x00 đến 0x07. Nếu prf bằng 0x07, DODAG root sẽ có độ ưu tiên cao nhất, giá trị mặc định của prf bằng 0
rank: rank của node gửi DIO
RPLInstanceID: trường 8 bit set bởi DODAG root cho biết RPLInstance mà DODAG là một phần
Version Number: Số nguyên không dấu 8 bit được đặt bởi gốc DODAG thành **DODAGVersionNumber:** số nguyên dương 8 bit đặt bởi DODAG root để chỉ phiên bản của DODAG
DTSN (Destination advertisement Trigger Sequence Number): được thiết lập tại node phát ra bản tin DIO. sử dụng trong quá trình duy trì các thông số định tuyến theo hướng downward
DODAGID: là một địa chỉ IpV6 có độ dài 128 bit và được thiết lập tại DODAG root

Flags: trường không sử dụng 8 bit dành riêng cho các cờ. Trường này phải được khởi tạo bằng 0 bởi sender và phải bị bỏ qua khi nhận

Trigger Timer: Vấn đề quảng bá các bản tin DIO là vấn đề quan trọng nhất trong quá trình triển khai và xây dựng DODAG. Do đó, mỗi node trong mạng luôn duy trì một bộ định thời quản lý tốc độ quảng bá các bản tin DIO đến các node khác trong mạng

Mỗi bộ định thời duy trì các thông số:

- I (current Interval): Khoảng thời gian hiện hành được sử dụng việc xác định khoảng thời gian định thời cho DIO
- T (Timer): Khoảng thời gian định thời có sẵn giá trị ngẫu nhiên trong khoảng [I/2, I].
- C (Counter): Bộ đếm được sử dụng trong cơ chế quản lý các bản tin DIO dư thừa, như các bản tin DIO được quảng bá từ các node có cùng rank hoặc từ những node có rank cao hơn. Cơ chế này làm giảm số lượng bản tin phải xử lý và giảm dư thừa thông tin.
- I-min: Khoảng thời gian định thời nhỏ nhất được tính bằng ms. Giá trị này được quy định bởi các DODAG root và bằng lũy thừa cơ số 2 của thông số DIO Interval Min trong bản tin DIO gửi từ DODAG root.
- I-doublings: số lần giá trị I được phép tăng gấp đôi trước khi được thiết lập lại giá trị ban đầu

Khi một node tham gia vào một DODAG, node sẽ thực hiện việc kích hoạt bộ định thời DIO để quản lý tốc độ quảng bá các bản tin DIO tời những node xung quanh.

Các thông số khởi tạo được thiết lập chung cho toàn DODAG như sau:

- I-min và I-doubling bằng các giá trị tạo bởi root
- C được thiết lập bằng 0
- I được thiết lập bằng I-min
- T được chọn ngẫu nhiên trong khoảng [I/2, I].

Sau mỗi lần gửi DIO, bộ định thời DIO tự động nhân đôi khoảng thời gian I và lựa chọn một thời gian định thời T mới thuộc khoảng [I/2, I].

Khi bộ định thời DIO được khởi động lại, các giá trị định thời được đưa về giá trị khởi tạo mặc định trong DODAG.

Bộ định thời được reset trong các trường hợp sau:

- Khi node tìm thấy một thay đổi có ảnh hưởng đến vị trí của node trong DODAG
- Khi node thay đổi rank trong DODAG
- Khi node nhận được một bản tin DIO từ một parent có sự thay đổi về rank, DODAG
- Khi node nhận được một gói dữ liệu từ một parent với các xung đột trong quá trình truyền gói
- Khi node nhận được một bản tin multicast DIS
- Khi node di chuyển từ DODAG này sang DODAG khác

Nhận và xử lý bản tin DIO:

DIO đầu tiên được tạo ra tại DODAG root, sau đó được quảng bá đến các node khác trong mạng. Các node thành viên trong DODAG sử dụng các bản tin DIO nhận được để cập nhật thông tin DODAG, lựa chọn parent và quảng bá vị trí của nó trong DODAG

Các phương thức xử lý DIO phải tuân theo các nguyên tắc sau:

- Một node chỉ tạo và gửi bản tin DIO sau khi tham gia DODAG xác định
- Những node không phải DODAG root, chỉ được phép thay đổi giá trị các cờ điều khiển S (Destination Advertisements Stored), DTSN và DODAG rank trong bản tin DIO nhận được từ DODAG root
- DIO là hợp lệ nếu có RPLInstenceID được đáp ứng

Khi một node nhận được một bản tin DIO từ trong mạng, node phải thực hiện kiểm tra tính hợp lệ của DIO đó. Nếu bản tin DIO được đáp ứng, node mới tiếp tục thực hiện những hành động tiếp theo.

b. Bản tin DAO

DAO(Destination Advertisement Object) là bản tin được sử dụng để quảng bá thông tin của các đích, được gửi từ những node có rank cao hơn đến node có rank thấp hơn dọc theo DODAG. Trong Storing mode, bản tin DAO từ node child sẽ được unicast đến các node parents đã lựa chọn. Trong Non-storing mode, bản tin DAO được unicast đến DODAG Root.

Cấu trúc bản tin DAO

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| RPLInstanceID |K|D|   Flags   |   Reserved    | DAOSequence   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                            DODAGID*                           +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Option(s)...
+-+-+-+-+-+-+-+-+
```

Hình 33:Bản tin DAO

- RPL InstanceID: Trường 8 bit cho biết topology của instance liên kết với DODAG
- K: Cờ "K" cho biết node nhận bản tin sẽ gửi lại DAO-ACK
- D: Cờ "D" cho biết trường DODAGID có mặt. Nó thường được cài đặt khi sử dụng RPLInstanceID cục bộ
- Flags: Trường 6 bit chưa sử dụng dành riêng cho flag. Trường này phải khởi tạo bằng 0 bên gửi và bỏ qua ở bên nhận.
- Reserved: Trường 8 bit, khởi tạo bằng 0 bên gửi và bỏ qua ở bên nhận.

- DAO Sequence: Tăng mỗi khi một bản tin DAO duy nhất từ node và phản hồi lại trong bản tin DAO-ACK
- DODAGID: Số nguyên không dấu 128 bit được đặt bởi một gốc DODAG xác định duy nhất một DODAG. Trường này chỉ xuất hiện khi cờ 'D' được đặt. Thông thường, trường này chỉ xuất hiện khi RPLInstanceID cục bộ được sử dụng để xác định DODAGID được liên kết với RPLInstanceID. Khi sử dụng RPLInstanceID toàn cầu, trường này không cần phải có.

c. Bản tin DIS

DIS(DODAG Information Solicitation) là bản tin sử dụng để yêu cầu bản tin DIO từ 1 nút trong mạng RPL.

Các node tự do trong mạng có thể sử dụng bản tin DIS để quảng bá sự xuất hiện của node, thăm dò sự xuất hiện của các node hàng xóm.

Bản tin DIS đƣợc gửi multicast khi node ở trạng thái tự do và đƣợc gửi unicast đến một parent trong DODAG khi muốn nhận lại một bản tin unicast DIO nhằm cập nhật các thông tin DODAG của parent đó.

Cấu trúc bản tin DIS:

0 – 7	8 – 15	16 – 31
Type = 155	Code = 0x01	Check sum
Instance ID	RPL DIS body	

Hình 34:Bản tin DIS

Trường Instance ID cho biết instance mà node gửi DIS tham gia định tuyến. Khi một node trong mạng nhận được một bản tin DIS, nó chỉ gửi lại bản tin DIO phản hồi khi có khả năng định tuyến cho Instance ID đó. Nhờ đó, khi triển khai mạng có nhiều instance, có thể giảm đáng kể số lƣợng bản tin điều khiển trong mạng, tiết kiệm năng lƣợng và thời gian xử lý của các node.

d. Bản tin DAO-ACK

Thông báo DAO-ACK được gửi dưới dạng gói unicast bởi người nhận DAO (DAO parent hoặc DODAG root) để phản hồi thông báo DAO unicast.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| RPLInstanceID |D|  Reserved   |  DAOSequence  |    Status     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                            DODAGID*                           +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Option(s)...
+-+-+-+-+-+-+-+-+
```

Hình 35:Bản tin DAO-ACK

“ * ” biểu thị rằng DODAGID không phải lúc nào cũng có mặt
RPLInstanceID: Trường 8 bit cho biết topology của instance chứa với DODAG.
D: Cờ “D” cho biết trường DODAGID có mặt. Nó thường được cài đặt khi sử dụng RPLInstanceID cục bộ
Reverved: Trường 7 bit, dành riêng cho các cờ
DAOSequence: Được tăng lên ở mỗi thông báo DAO từ một nút và được người nhận lặp lại trong DAO-ACK.
Status: Biểu thị sự hoàn thành. Trạng thái 0 được định nghĩa là chấp nhận không đủ tiêu chuẩn trong đặc điểm kỹ thuật này. Các giá trị trạng thái còn lại được dành làm mã từ chối.
DODAGID: Số nguyên không dấu 128 bit được đặt bởi một gốc DODAG xác định duy nhất một DODAG. Trường này chỉ xuất hiện khi cờ 'D' được đặt. Thông thường, trường này chỉ xuất hiện khi RPLInstanceID cục bộ được sử dụng để xác định DODAGID được liên kết với RPLInstanceID. Khi sử dụng RPLInstanceID toàn cầu, trường này không cần phải có.
Các bit chưa được gán của DAO-ACK được bảo lưu. Chúng phải được đặt bằng 0 trong quá trình truyền và phải được phớt lờ khi tiếp nhận

3.1.4. Quá trình khởi tạo mạng

Khi triển khai một mạng RPL, mỗi RPL Instance được thiết lập với một hoặc một số DODAG root. Các thông số định tuyến được thiết lập phù hợp với mục đích triển khai. Những DODAG root tự động thiết lập rank bằng 1 (Root Rank), sau đó chúng định thời quảng bá các bản tin DIO đến những node xung quanh để xây dựng DODAG của bản than

Trong pha khởi tạo, những node khác trong mạng có thể lựa chọn một trong hai chế độ: hoặc chúng giữ trạng thái silent và không gửi bất kỳ bản tin DIO nào

cho đến khi chúng tham gia vào một DODAG xác định; hoặc ngay lập tức tự thiết lập DODAG root của một floating DODAG, sau đó gửi multicast các bản tin DIO đến các node khác trong mạng.

Trong quá trình này, mỗi node cũng có thể gửi multicast DIS đến các node xung quanh hoặc chờ những bản tin DIO được gửi đến. Khi triển khai mạng, cần thiết kế để các node trong mạng có khả năng đáp ứng được những cơ chế trên. Điều đó giúp quá trình khởi tạo mạng nhanh chóng và linh động hơn

3.1.5. Quá trình định tuyến upward

Trong RPL, quá trình định tuyến upward có vai trò then chốt, quyết định tính chất, hiệu năng hoạt động của mạng. Quá trình này dựa trên việc xử lý các bản tin DIO, xây dựng, xác định và duy trì các DODAG, từ đó mỗi node trong DODAG có thể xác định tuyến đường tối ưu để gửi dữ liệu về DODAG root một cách nhanh chóng và hiệu quả.

Các nút quyết định tham gia DODAG PHẢI cung cấp ít nhất một DODAG parent làm tuyến mặc định cho Instance liên quan. Tuyến đường mặc định này cho phép một gói được chuyển tiếp upward cho đến khi cuối cùng nó chạm vào một tổ tiên chung từ mà nó sẽ được định tuyến Downward đến đích. Nếu đích không có trong DODAG, thì gốc DODAG có thể chuyển tiếp gói bằng cách sử dụng kết nối ra bên ngoài DODAG; nếu như nó không thể chuyển tiếp gói ra bên ngoài, thì DODAG root phải bỏ nó.

a. **DODAG parent và preferred parent**

Trong RPL sử dụng ba khái niệm thể hiện mối quan hệ logic giữa các node trong mạng, bao gồm: neighbor, DODAG parent và preferred parent. Trong đó preferred parent của một node là một trong số những DODAG parent có các thông số định tuyến tốt nhất. Phƣơng thức lựa chọn preferred parent được quy định bởi các hàm OF. Preferred parent sẽ được chọn là next-hop trong quá trình truyền gói.

b. **Quản lý nenighbor, khám phá DODAG và lựa chọn DODAG parent**

Trong RPL, mỗi node trong mạng tìm kiếm và quản lý các node neighbor trong phạm vi kết nối thông qua cơ chế "neighbor discovery in IPv6" . Theo đó, mỗi node trong mạng sẽ quản lý đƣợc các thông số trạng thái của những node xung quanh như khả năng kết nối, địa chỉ, tình trạng,..

Khi một node được khởi tạo với vai trò DODAG root, node sẽ thực hiện việc tạo những bản tin DIO với rank bằng một và quảng bá chúng đến các node xung quanh. Những node gần root nhất khi nhận đƣợc DIO từ DODAG root sẽ tham gia vào DODAG, cập nhật thông tin trong DIO và xác định rank của bản thân trong DODAG. Sau đó node định thời chuyển tiếp các bản tin DIO đến những node xung quanh, quảng bá vị trí và DODAG mà nó tham gia. Quá trình này đuợc các thành viên trong DODAG lặp lại liên tục tại những thời điểm định thời.

Khi một node ở trạng thái tự do (chƣa tham gia DODAG), node sẽ định thời gửi các bản tin DIS tới các neighbor để quảng bá instance mà nó có khả năng tham gia. Đồng thời yêu cầu những node đáp ứng đƣợc phản hồi lại những thông

tin về DODAG mà chúng tham gia. Khi một neighbor nhận đƣợc bản tin DIS, nếu đã tham gia một DODAG phù hợp với Instance trong DIS, nó sẽ phản hồi bằng một bản tin DIO tới node gửi DIS

3.1.6. Quá trình định tuyến downward

RPL xây dựng và duy trì định tuyến downward with bản tin DAO (destination advertisement object). Định tuyến downward hỗ trợ cơ chế giao tiếp P2MP (point to multiple points), từ DODAG root xuống các node lá. Định tuyến downward cũng hỗ trợ cơ chế giao tiếp P2P (Point to point): bản tin P2P có thể hướng lên DODAG root (hoặc bất kỳ một cha chung nào) thông qua định tuyến upward, tiếp đó từ DODAG root (cha chung) tới node đích thông qua định tuyến downward

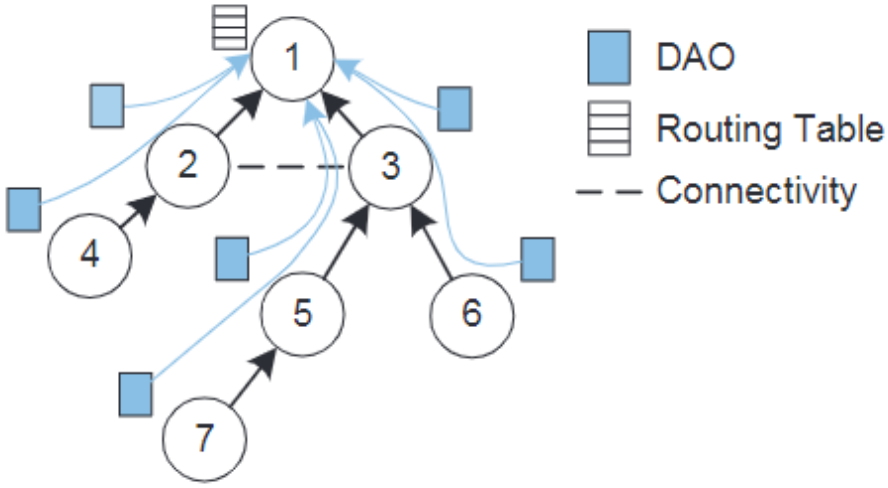
Có 2 chế độ phiên bản RPL có thể được chọn để duy trì tuyến đường đi xuống (downward). Đầu tiên là chế độ "storing", node lưu trữ các bảng định tuyến hướng xuống cho các sub-DODAG của chúng. Mỗi bước nhảy trên một tuyến đường đi xuống trong mạng lưu trữ sẽ kiểm tra bảng định tuyến để quyết định chặng tiếp theo. Ở chế độ thứ hai: "non-storing", node không lưu trữ bảng định tuyến downward. Các gói tin được định tuyến với các tuyền nguồn được tạo bởi DODAG root.

a. Destination advertisement parents

Để thiết lập bản định tuyến downward, các node trong mạng RPL gửi upward bản tin DAO. Các điểm đến next-hope của bản tin DAO được gọi là DAO parent. Tập hợp các node DAO parents được gọi là "DAO parents set"

- Một node có thể gửi bản tin DAO sử dụng gửi multicast tới tất cả các địa chỉ . Đó là cách tối ưu hóa để cung cấp định tuyến một bước. Bit K phải được xóa khi truyền DAO multicast
- Tập cha DAO của một node phải là tập con của tập cha DODAG của nó. Điều này đảm bảo rằng thông tin địa chỉ đích và đường dẫn của một ndoe con được truyền đi thông qua node cha và các node cha khác trên đường dẫn ngược về tới root
- Trong chế độ hoạt động "storing", một node không được gửi các bản tin DAO unicast tới các node không phải là DAO parents.
- Trong chế độ hoạt động "storing", địa chỉ đích và nguồn IPv6 của bản tin DAO phải là địa chỉ liên kết cục bộ, để đảm bảo rằng bản tin DAO chỉ được lưu trữ và xử lý bởi các node trong liên kết mạng cục bộ, không bị lan truyền ra các liên kết mạng khác hoặc tác động đến quá trình định tuyến trong các liên kết mạng khác
- Trong chế độ hoạt động "non-storing", 1 node không được giải quyết các bản tin DAO unicast tới các node không phải là node gốc
- Trong chế độ hoạt động "non-storing", địa chỉ đích và nguồn IPv6 của bản tin DAO phải là địa chỉ cục bộ duy nhất hoặc địa chỉ toàn cục

b. Non-storing mode



Hình 36:Chế độ không lưu trữ Non-storing

Trong non-storing mode, RPL định tuyến tin nhắn xuống dưới sử dụng định tuyến nguồn IP. Quy tắc sau chỉ được áp dụng cho các node ở chế độ non-storing (Storing mode có 1 chuỗi quy tắc riêng biệt).

- Trường con DODAO parent address của tùy chọn Transit Information phải chứa một hoặc nhiều địa chỉ. Tất cả các địa chỉ này phải là địa chỉ của DAO parent của sender.
- DAOs được gửi trực tiếp đến root dọc theo 1 tuyến mặc định được cài đặt như 1 phần của lựa chọn parent
- Khi một node xóa một node khỏi tập DAO parent của nó, nó phải sinh ra một bản tin DAO mới với sự cập nhật tùy chọn Transit Information

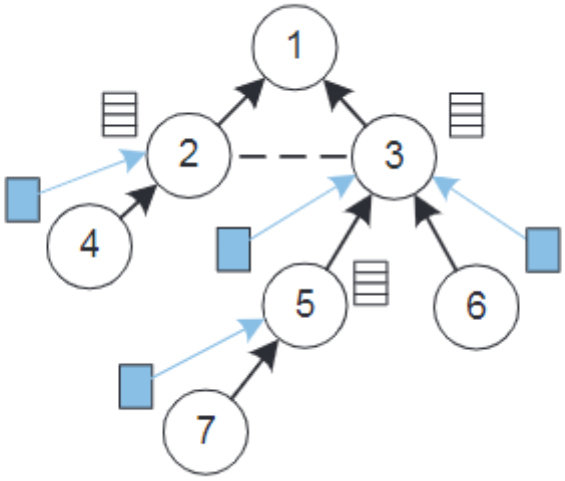
Trong non-storing mode, một node sử dụng bản tin DAOS để báo cáo lại các DAO parents của nó cho DODAG root. DODAG root có thể ghép nối một tuyến đường đi xuống tới một node bằng cách sử dụng các tập DAO parent từ mỗi node trong tuyến đường. Thông tin Path sequence có thể được sử dụng để phát hiện thông tin DAO cũ. Mục đích của việc tính toán tuyến đường perhop này là để giảm thiểu lưu lượng khi DAO parent thay đổi. Nếu các node đã báo cáo các tuyến nguồn hoàn chỉnh thì khi DAO gốc thay đổi, toàn bộ DODAG con sẽ phải gửi các DAO mới tới DODAG root. Do đó, ở non-storing mode, một node có thể gửi một DAO duy nhất, mặc dù nó có thể chọn gửi nhiều hơn một thông báo DAO cho mỗi trong số nhiều DAO parents.

Node gói bản tin DAOs bằng cachs gửi một bản tin DAO đơn với nhiều tùy chọn mục tiêu RPL. Mỗi tùy chọn mục tiêu RPL có các tùy chọn Transit Information riêng ngay sau đó.

c. Storing mode

Trong storing mode, RPL định tuyến bản tin xuống dưới thông qua địa chỉ đích IPv6. Tương tự như chế độ non-storing, chế độ storing cũng yêu cầu tạo lại các thông báo DAO. Cấu hình của bộ timer kích hoạt các thông báo. Tuy nhiên, một

DAO không còn được truyền đến root, thay vào đó, nó được gửi unicast tời tất cả các node cha mẹ duy trì bảng định tuyến downward bổ sung.



Hình 37:Chế độ lưu trữ Storing

Khi một node gửi thông báo DAO, nó phải giữ trống trường Parent address trong tùy chọn thông tin chuyển tiếp (transit information) vì trách nhiệm của một node không phải là quảng cáo tập hợp cha mẹ của nó, mà là thông báo các tiến tố có thể truy cập thông qua nó. Nếu thiết bị là bộ định tuyến, thiết bị phải sử dụng tùy chọn Target để quảng cáo tiền tố. Trong trường hợp nó có nhiều tiến tố trong quảng cáo, nó phải mở rộng DAO bằng nhiều tùy chọn Target. Nếu một gói dữ liệu được gửi xuống từ DODAG root, nó chỉ được gửi tời tất cả các one-hop neighbors. Sau đó, tra cứu thông qua gói được định tuyến xuống dưới cho đến khi đến đích hoặc hop-limit trong tiêu đề IPv6 đạt 0. Các mạng như vậy cũng có thể được tổ chức theo thứ bậc và có thể thực hiện tổng hợp định tuyến. Ngoài ra, một DODAG phụ có thể chạy được một phiên bản RPL khác, điều này có thể thực hiên để kết hợp storing mode và non-storing mode. Bằng cách này,node với ràng buộc về bộ nhớ có thể nhóm lại với nhau và chỉ hoạt động ở chế độ non-storing mode.

Khi một node tạo một DAO mới, nó nên truyền đơn hướng tới từng DAO parent của nó, không được phát đơn hướng tới các node không phải là parent. Node sẽ sủ dụng trạng thái được lưu trữ của các DAO mà nó đã nhận được để tạo ra một tập hợp các tùy chọn mục tiêu RPL và các tùy chọn thông tin chuyển tiếp liên quan của chúng

Khi node xóa một node khỏi tập DAO parents của nó, nó nên gửi bản tin No-Path DAO đến DAO parent bị xóa bỏ đó, để làm mất hiệu lực tuyến đường hiện có

Nếu các bản tin Downward address được quảng cáo gặp phải lỗi chuyển tiếp, phát hiện không thể truy cập hàng xóm (NUD) hoặc lỗi tương tự, một node có thể đánh dấu địa chỉ là không thể truy cập và tạo DAO không đường dẫn thích hợp

DAO quảng cáo tới địa chỉ đích và tiền tố mà một node có định tuyến

Bởi vì thông tin được gói trong bảng định tuyến của mỗi node trong storing mode, DAOs giao tiếp trực tiếp với DAO parents, là các node giữ các thông tin này

3.2. Các hàm mục tiêu cơ bản trong RPL

Hàm mục tiêu trong RPL tối ưu hóa đường định tuyến trong mạng dựa trên các metric của hàm như hop count, latency hay expected transmission count (ETX)

3.2.1. The Minimum Rank with Hysteresis Objective Function – MRHOF

3.2.1.1. Giới thiệu

Minimum Rank with Hysteresis Objective Function (MRHOF), một Objective Function cho RPL, sử dụng hysteresis trong quá trình lựa chọn đường đi có giá trị chỉ số nhỏ nhất. Chỉ số mà MRHOF sử dụng được xác định bởi các chỉ số trong DIO Metric Container. Ví dụ, việc sử dụng MRHOF với chỉ số độ trễ (latency) cho phép RPL tìm đường đi có độ trễ tối thiểu và ổn định từ các nút đến gốc trong đồ thị DAG (Directed Acyclic Graph) [RFC6550]. Việc sử dụng MRHOF với chỉ số Expected Transmission Count (ETX) [RFC6551] cho phép RPL tìm đường đi có ETX tối thiểu và ổn định từ các nút đến gốc trong đồ thị DAG. Trong trường hợp không có chỉ số trong DIO Metric Container hoặc không có DIO Metric Container, MRHOF mặc định sử dụng ETX để tính toán Rank.

3.2.1.2. Các thuật ngữ

- Selected metric (Chỉ số được chọn): Chỉ số được lựa chọn để lựa chọn đường đi bởi người điều hành mạng. MRHOF hỗ trợ việc sử dụng một chỉ số duy nhất để lựa chọn đường đi. Quyết định sử dụng một chỉ số (khác với ETX) như là chỉ số được chọn được chỉ ra bằng sự hiện diện của chỉ số đã chọn trong DIO Metric Container. Sự lựa chọn chỉ số ETX được chỉ ra bằng sự vắng mặt của Metric Container, trong trường hợp này, ETX được quảng cáo như Rank.

- Path cost (Chi phí đường đi): Chi phí đường đi đo lường một thuộc tính của một đường đi từ đầu đến cuối. Chi phí đường đi được tính bằng cách mỗi nút tổng hợp chỉ số liên kết được chọn vào chi phí đường đi được quảng cáo bởi cha. Chi phí đường đi có thể được sử dụng bởi RPL để so sánh các đường đi khác nhau.

- Worst parent (Cha tồi nhất): Nút trong tập cha có chi phí đường đi lớn nhất.

3.2.1.3. Tính toán chi phí đường đi

Các nút gốc (Grounded hoặc Floating) đặt biến cur_min_path_cost thành giá trị chỉ số tính toán tương ứng với một Rank là MinHopRankIncrease.

Nếu một nút không phải gốc không có chỉ số để tính toán chi phí đường đi qua bất kỳ hàng xóm ứng cử viên nào, nó PHẢI tham gia vào một trong những hàng xóm ứng cử viên như một RPL Leaf.

Nếu không, các nút tính toán chi phí đường đi cho mỗi hàng xóm ứng cử viên có thể đạt được thông qua một giao diện. Chi phí đường đi của một hàng xóm đại diện cho chi phí của đường đi, dựa trên chỉ số đã chọn, từ một nút đến gốc của Destination-Oriented DAG (DODAG) thông qua hàng xóm đó. Một nút không phải gốc tính toán chi phí đường đi của hàng xóm bằng cách thêm hai thành phần:

- Nếu chỉ số đã chọn là một chỉ số liên kết, thì chỉ số đã chọn cho liên kết đến hàng xóm ứng cử viên. Nếu chỉ số đã chọn là một chỉ số nút, thì chỉ số đã chọn cho nút đó.

- Giá trị của chỉ số đã chọn trong Metric Container trong DIO được gửi bởi hàng xóm đó. Trong trường hợp Metric Container là trống, ETX là chỉ số đã chọn - sử dụng Rank được quảng cáo bởi hàng xóm đó như là thành phần thứ hai.

Một nút NÊN tính toán chi phí đường đi cho đường đi qua mỗi hàng xóm ứng cử viên có thể đạt được thông qua một giao diện. Nếu một nút không thể tính toán được chi phí đường đi cho đường đi qua một hàng xóm ứng cử viên, nút PHẢI KHÔNG chọn hàng xóm ứng cử viên đó làm cha ưu tiên của nó. Tuy nhiên, nếu nút không thể tính toán được chi phí đường đi qua bất kỳ hàng xóm nào, nó có thể tham gia vào hàng xóm ứng cử viên như một RPL Leaf, như đã mô tả ở trên.

Nếu chỉ số đã chọn là một chỉ số liên kết và chỉ số của liên kết đến một hàng xóm không khả dụng, chi phí đường đi cho đường đi qua hàng xóm đó NÊN được đặt thành MAX_PATH_COST. Giá trị chi phí này sẽ ngăn cản đường đi này được xem xét cho việc lựa chọn đường đi.

Nếu chỉ số đã chọn là một chỉ số nút và chỉ số không khả dụng, chi phí đường đi qua tất cả các hàng xóm NÊN được đặt thành MAX_PATH_COST.

Chi phí đường đi tương ứng với một hàng xóm NÊN được tính toán lại mỗi khi đáp ứng một trong các điều kiện sau:

- Chỉ số đã chọn của liên kết đến hàng xóm ứng cử viên được cập nhật.

- Chỉ số đã chọn là một chỉ số nút và chỉ số được cập nhật.

- Một nút nhận được một quảng cáo chỉ số mới từ hàng xóm ứng cử viên.

Phép tính này NÊN được thực hiện định kỳ. Trong khi việc trì hoãn phép tính này lên tới một khoảng thời gian Trickle tối thiểu [RFC6550] không gây hại, nhưng sự trì hoãn lâu hơn trong việc cập nhật chi phí đường đi sau khi chỉ số được cập nhật hoặc nhận quảng cáo chỉ số mới có thể dẫn đến thông tin lỗi thời.

3.2.1.3 Lựa chọn node cha phù hợp

Sau khi tính toán chi phí đường đi cho tất cả các hàng xóm ứng cử viên có thể đạt được thông qua một giao diện cho vòng lặp DODAG hiện tại [RFC6550], một nút chọn cha ưu tiên. Quá trình này được gọi là "lựa chọn cha". Để cho phép hysteresis, lựa chọn cha duy trì một biến cur_min_path_cost, đó là chi phí đường đi của cha ưu tiên hiện tại.

Khi parent selection chạy

Một ứng dụng MRHOF NÊN thực hiện parent selection mỗi khi:

1. Chi phí đường đi cho một hàng xóm ứng cử viên hiện có, bao gồm cha ưu tiên, thay đổi. Điều kiện này có thể được kiểm tra ngay sau khi tính toán chi phí đường đi.
2. Một hàng xóm ứng cử viên mới được chèn vào bảng danh sách các hàng xóm.

Nếu không có đủ 2 điều kiện trên, cần phải hoãn việc lựa chọn cha cho đến một thời gian sau (ví dụ, lên đến khoảng thời gian tối thiểu của Trickle [RFC6550]), lưu ý rằng làm như vậy có thể làm trì hoãn việc sử dụng các đường đi tốt hơn có sẵn trong mạng.

Thuật toán Lựa chọn Cha

Nếu chỉ số đã chọn cho một liên kết lớn hơn MAX_LINK_METRIC, nút NÊN loại trừ liên kết đó khỏi việc xem xét trong quá trình lựa chọn cha.

Một nút PHẢI chọn hàng xóm ứng cử viên có chi phí đường đi thấp nhất làm cha ưu tiên của nó, trừ khi có chỉ dẫn như sau:

1. Một nút CÓ THỂ tự xác định mình là gốc Floating, và do đó không có cha ưu tiên, phụ thuộc vào cấu hình hệ thống.

2. Nếu cur_min_path_cost lớn hơn MAX_PATH_COST, nút CÓ THỂ tự xác định mình là gốc Floating.

3. Nếu chi phí đường đi nhỏ nhất cho các đường đi qua các hàng xóm ứng cử viên nhỏ hơn cur_min_path_cost ít hơn PARENT_SWITCH_THRESHOLD, nút CÓ THỂ tiếp tục sử dụng cha ưu tiên hiện tại. Đây là thành phần hysteresis của MRHOF.

4. Nếu ALLOW_FLOATING_ROOT là 0 và không có hàng xóm nào được khám phá, nút không có cha ưu tiên và PHẢI đặt cur_min_path_cost thành MAX_PATH_COST.

Nếu có nhiều hàng xóm chia sẻ chi phí đường đi nhỏ nhất, một nút CÓ THỂ sử dụng tiêu chí lựa chọn khác để chọn xem hàng xóm nào trong số này sẽ được coi là có chi phí thấp nhất.

Một nút CÓ THỂ bao gồm tối đa PARENT_SET_SIZE-1 hàng xóm ứng cử viên bổ sung trong tập cha của nó. Chi phí của đường đi qua các nút trong tập cha nhỏ hơn hoặc bằng chi phí của đường đi qua bất kỳ nút nào không thuộc tập cha. Nếu chi phí của đường đi qua cha ưu tiên và cha xấu nhất quá lớn, một nút CÓ THỂ giữ một tập cha nhỏ hơn so với PARENT_SET_SIZE.

Sau khi chọn cha ưu tiên, nút đặt biến cur_min_path_cost của mình thành chi phí đường đi tương ứng với cha ưu tiên. Giá trị của cur_min_path_cost được mang trong Metric Container tương ứng với chỉ số đã chọn khi gửi các thông điệp DIO.

3.2.1.4. Tính toán rank

Các gốc DAG đặt Rank của chúng thành MinHopRankIncrease.

Sau khi một nút không phải gốc chọn tập cha của mình, nó có thể sử dụng bảng sau để chuyển đổi chi phí đường đi của một cha (được viết là Cost trong bảng) thành giá trị Rank:

Node/Link metric	Rank
Hop count	Cost
Latency	Cost/65536
ETX	Cost

Nếu MRHOF được sử dụng với các chỉ số khác, thì Rank là không xác định. Nếu Rank là không xác định, nút phải tham gia một trong các hàng xóm là một nút RPL Leaf theo [RFC6550].

MRHOF sử dụng giá trị Rank này để tính toán Rank liên quan đến đường đi thông qua mỗi thành viên của tập cha. Rank liên quan đến đường đi thông qua một thành viên của tập cha là giá trị lớn nhất của hai giá trị. Giá trị đầu tiên là giá trị Rank tương ứng được tính toán bằng bảng trên, giá trị thứ hai là Rank được quảng bá của các nút cộng với MinHopRankIncrease.

Một nút đặt Rank của mình là giá trị lớn nhất của ba giá trị:

1. Rank được tính toán cho đường đi thông qua cha ưu tiên.

2. Rank của thành viên trong tập cha có Rank quảng bá cao nhất, làm tròn lên đến Rank nguyên cao hơn tiếp theo, tức là MinHopRankIncrease * (1 + floor(Rank/MinHopRankIncrease)).

3. Rank được tính toán lớn nhất giữa các đường đi thông qua tập cha, trừ đi MaxRankIncrease.

3.2.1.5. Quảng bá chi phí đường đi

Sau khi đã chọn được node cha ưu tiên, nút sẽ thiết lập biến cur_min_path_cost của mình thành chi phí đường dẫn tương ứng với node cha ưu tiên. Sau đó, nút tính toán chỉ số số học mà nó sẽ quảng bá trong metric container của mình. Giá trị này là chi phí đường dẫn của thành viên trong tập node cha có chi phí cao nhất. Do đó, trong khi cur_min_path_cost là chi phí thông qua node cha ưu tiên, một nút quảng bá đường dẫn chi phí cao nhất từ nút đến gốc thông qua một thành viên trong tập node cha. Giá trị của đường dẫn chi phí cao nhất này được chứa trong metric container tương ứng với số đo được chọn khi gửi các thông điệp DIO.

Nếu ETX là số đo được chọn, nút PHẢI KHÔNG quảng bá nó trong metric container. Thay vào đó, nút PHẢI quảng bá một xấp xỉ của ETX trong giá trị Rank được quảng bá của nó. Nếu một nút nhận được một DIO với một metric container chứa số đo ETX, MRHOF PHẢI bỏ qua giá trị số đo ETX trong các phép tính Rank của nó.

Gốc DODAG quảng bá một giá trị metric tính toán đến một giá trị Rank của MinHopRankIncrease.

3.2.1.6. Các biến và tham số trong MRHOF

MRHOF sử dụng các biến sau:

-) cur_min_path_cost: Chi phí của đường dẫn từ một nút thông qua node cha ưu tiên của nó đến gốc được tính toán trong lần chọn node cha cuối cùng.

MRHOF sử dụng các tham số sau:

-) MAX_LINK_METRIC: Giá trị tối đa cho metric liên kết được chọn cho mỗi liên kết trên đường dẫn.

-) MAX_PATH_COST: Giá trị tối đa cho metric đường dẫn của một đường dẫn được chọn.

-) PARENT_SWITCH_THRESHOLD: Sự khác biệt giữa chi phí đường dẫn thông qua node cha ưu tiên và đường dẫn có chi phí tối thiểu để kích hoạt việc chọn một node cha ưu tiên mới.

-) PARENT_SET_SIZE: Số lượng node cha ứng cử viên, bao gồm node cha ưu tiên, trong tập node cha.

-) ALLOW_FLOATING_ROOT: Nếu được đặt thành 1, cho phép một nút trở thành một gốc lơ lửng.

Các giá trị tham số được gán tùy thuộc vào số đo được chọn. Các giá trị tốt nhất cho các tham số này được xác định bởi yêu cầu của việc triển khai RPL cụ thể. Ví dụ, nếu chúng ta sử dụng ETX làm số đo được chọn và UDP làm giao thức truyền tải, chúng ta nên sử dụng một MAX_LINK_METRIC nhỏ (ví dụ: ETX là 1.1) để các lần gửi lại của lớp liên kết đủ để cung cấp khả năng tin cậy từ đầu đến cuối.

Dựa trên những kinh nghiệm làm việc với RPL của tổ chức IETF, các giá trị sau được KHUYẾN NGHỊ khi ETX là số đo được chọn:

MAX_LINK_METRIC: 512. Không cho phép các liên kết với số lượng truyền gửi dự kiến lớn hơn 4 trên đường dẫn đã chọn.

MAX_PATH_COST: 32768. Không cho phép các đường dẫn với số lượng truyền gửi dự kiến lớn hơn 256.

PARENT_SWITCH_THRESHOLD: 192. Chuyển sang một đường dẫn mới chỉ khi dự kiến yêu cầu ít nhất 1,5 truyền gửi ít hơn đường dẫn hiện tại.

PARENT_SET_SIZE: 3. Nếu node cha ưu tiên không khả dụng, vẫn còn hai node cha ứng cử viên mà không kích hoạt một vòng mới của việc khám phá đường dẫn.

ALLOW_FLOATING_ROOT: 0. Không cho phép một nút trở thành gốc lơ lửng.

3.2.1.7. MRHOF trong ContikiOS

Định các thông số cho trong mạng:
-) MAX_LINK_METRIC
-) PARENT_SWITCH_THRESHOLD (đại diện cho tính trễ)
-) MAX_PATH_COST

```
#define MAX_LINK_METRIC     1024 /* Eq ETX of 8 */

/*
 * Hysteresis of MRHOF: the rank must differ more than
 * PARENT_SWITCH_THRESHOLD_DIV in order to switch preferred
 * parent. Default in RFC6719: 192, eq ETX of 1.5.  We use a more
 * aggressive setting: 96, eq ETX of 0.75.
 */
#define PARENT_SWITCH_THRESHOLD 96 /* Eq ETX of 0.75 */
#else /* !RPL_MRHOF_SQUARED_ETX */
#define MAX_LINK_METRIC     2048 /* Eq ETX of 4 */
#define PARENT_SWITCH_THRESHOLD 160 /* Eq ETX of 1.25 (results in a churn comparable
                                       to the threshold of 96 in the non-squared case) */
#endif /* !RPL_MRHOF_SQUARED_ETX */

/* Reject parents that have a higher path cost than the following. */
#define MAX_PATH_COST      32768   /* Eq path ETX of 256 */
```

Hình 38: Định các thông số trong MRHOF trên ContikiOS

-) Chỉ số liên kết của cha: parent_link_metric được lấy từ hàm rpl_get_parent_link_stats(p)
-) Được quảng báo từ node cha tới node con

```c
static uint16_t
parent_link_metric(rpl_parent_t *p)
{
  const struct link_stats *stats = rpl_get_parent_link_stats(p);
  if(stats != NULL) {
#if RPL_MRHOF_SQUARED_ETX
    uint32_t squared_etx = ((uint32_t)stats->etx * stats->etx) / LINK_STATS_ETX_DIVISOR;
    return (uint16_t)MIN(squared_etx, 0xffff);
#else /* RPL_MRHOF_SQUARED_ETX */
    return stats->etx;
#endif /* RPL_MRHOF_SQUARED_ETX */
  }
  return 0xffff;
}
```

Hình 39: Chỉ số liên kết node cha MRHOF trong ContikiOS

-) Trả về giá trị path cost bằng cách tổng hợp chỉ số liên kết của node cha

```c
static uint16_t
parent_path_cost(rpl_parent_t *p)
{
  uint16_t base;

  if(p == NULL || p->dag == NULL || p->dag->instance == NULL) {
    return 0xffff;
  }

#if RPL_WITH_MC
  /* Handle the different MC types */
  switch(p->dag->instance->mc.type) {
  case RPL_DAG_MC_ETX:
    base = p->mc.obj.etx;
    break;
  case RPL_DAG_MC_ENERGY:
    base = p->mc.obj.energy.energy_est << 8;
    break;
  default:
    base = p->rank;
    break;
  }
#else /* RPL_WITH_MC */
  base = p->rank;
#endif /* RPL_WITH_MC */

  /* path cost upper bound: 0xffff */
  return MIN((uint32_t)base + parent_link_metric(p), 0xffff);
}
```

Hình 40: Chi phí đường truyền của node cha

-) Tính rank thông qua node cha

```c
static rpl_rank_t
rank_via_parent(rpl_parent_t *p)
{
  uint16_t min_hoprankinc;
  uint16_t path_cost;

  if(p == NULL || p->dag == NULL || p->dag->instance == NULL) {
    return RPL_INFINITE_RANK;
  }

  min_hoprankinc = p->dag->instance->min_hoprankinc;
  path_cost = parent_path_cost(p);

  /* Rank lower-bound: parent rank + min_hoprankinc */
  return MAX(MIN((uint32_t)p->rank + min_hoprankinc, 0xffff), path_cost);
}
```

Hình 41: Tính rank thông qua node cha

-) Lựa chọn ra node cha được chấp nhận và có thể sử dụng

```c
static int
parent_is_acceptable(rpl_parent_t *p)
{
  uint16_t link_metric = parent_link_metric(p);
  uint16_t path_cost = parent_path_cost(p);
  /* Exclude links with too high link metrics or path cost. (RFC6719, 3.2.2) */
  return link_metric <= MAX_LINK_METRIC && path_cost <= MAX_PATH_COST;
}
/*---------------------------------------------------------------------------*/
static int
parent_has_usable_link(rpl_parent_t *p)
{
  uint16_t link_metric = parent_link_metric(p);
  /* Exclude links with too high link metrics  */
  return link_metric <= MAX_LINK_METRIC;
}
```

Hình 42: Lựa chọn node cha có thể sử dụng

-) Lựa chọn ra cha tốt nhất

```
static rpl_parent_t *
best_parent(rpl_parent_t *p1, rpl_parent_t *p2)
{
  rpl_dag_t *dag;
  uint16_t p1_cost;
  uint16_t p2_cost;
  int p1_is_acceptable;
  int p2_is_acceptable;

  p1_is_acceptable = p1 != NULL && parent_is_acceptable(p1);
  p2_is_acceptable = p2 != NULL && parent_is_acceptable(p2);

  if(!p1_is_acceptable) {
    return p2_is_acceptable ? p2 : NULL;
  }
  if(!p2_is_acceptable) {
    return p1_is_acceptable ? p1 : NULL;
  }

  dag = p1->dag; /* Both parents are in the same DAG. */
  p1_cost = parent_path_cost(p1);
  p2_cost = parent_path_cost(p2);

  /* Maintain the stability of the preferred parent in case of similar ranks. */
  if(p1 == dag->preferred_parent || p2 == dag->preferred_parent) {
    if(p1_cost < p2_cost + PARENT_SWITCH_THRESHOLD &&
       p1_cost > p2_cost - PARENT_SWITCH_THRESHOLD) {
      return dag->preferred_parent;
    }
  }

  return p1_cost < p2_cost ? p1 : p2;
}
```

Hình 43: Node cha tốt nhất

3.2.2. Objective Function 0 - OF0

Trong RPL, hàm OF0 là một hàm mục tiêu cơ bản khác để tính toán rank của thiết bị. Trong phạm vi của Đồ án này chúng em không đi sâu vào tìm hiểu cơ chế hàm OF0 mà chỉ dừng lại ở mức độ giới thiệu.

OF0 (Objective Function Zero) là một trong các hàm mục tiêu được sử dụng trong giao thức định tuyến RPL (Routing Protocol for Low-Power and Lossy Networks). OF0 được thiết kế như một hàm mục tiêu mặc định cho phép tương tác giữa các cài đặt trong một loạt các trường hợp sử dụng khác nhau.

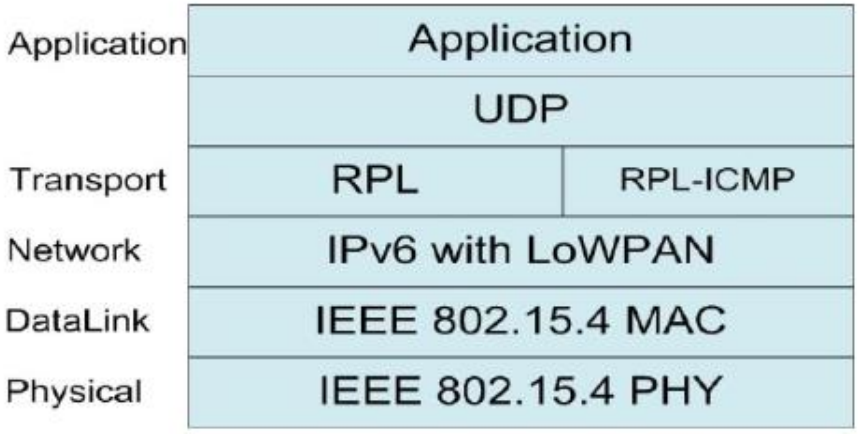
Mục tiêu chính của OF0 là cho phép các nút tham gia vào một phiên bản DODAG (Destination-Oriented Directed Acyclic Graph) cung cấp kết nối đủ tốt đến một tập hợp cụ thể các nút hoặc một hạ tầng định tuyến lớn hơn, mặc dù không có đảm bảo rằng đường đi sẽ được tối ưu theo một metric cụ thể. Việc xác thực kết nối này phụ thuộc vào triển khai và loại liên kết và nằm ngoài phạm vi của OF0.

OF0 hoạt động bằng cách tính toán Rank của mỗi nút trong mạng RPL. Rank của một nút được tính bằng cách thêm một rank_increase (tăng Rank) vào Rank của một preferred parent được chọn, rank_increase được tính dựa trên step_of_rank, một giá trị chuẩn hóa tương đối của các thuộc tính liên kết. OF0 cho phép điều chỉnh rank_increase bằng cách nhân step_of_rank với một rank_factor để phản ánh sự ưu tiên tương đối giữa các loại liên kết khác nhau mà được sử dụng trong cùng một phiên bản RPL.

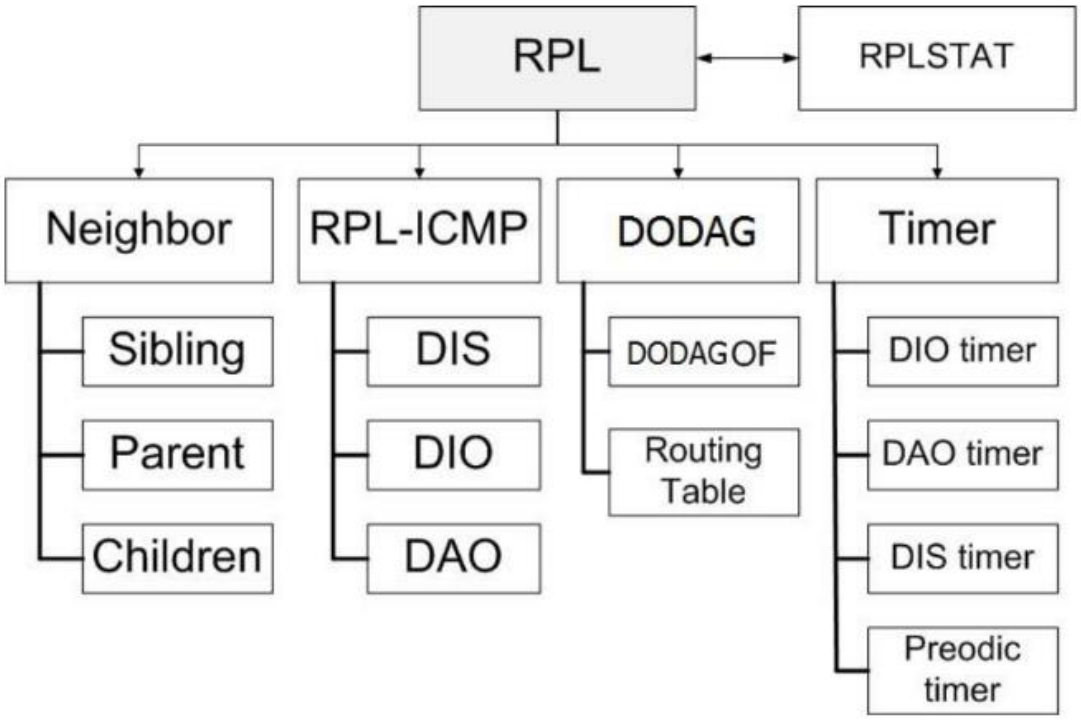
OF0 không chỉ định cách chuyển đổi thuộc tính liên kết thành rank_increase và để việc này do cài đặt thực hiện. OF0 chỉ chuẩn hóa giá trị của step_of_rank cho một liên kết bình thường và phạm vi chấp nhận được, thay vì xác định chi tiết của quá trình tính toán step_of_rank. OF0 cũng không quan tâm đến các metric containers.

3.3. Tổng quan về Contiki-RPL

Contiki RPL là một bộ phận trong stack uIP, hoạt động trên IPv6 và sử dụng giao thức truyền tin không tin cậy UDP/IP trên nền 6LowPan



Hình 44:Contiki-RPL



Hình 45:Cấu trúc module của Contiki RPL

Hình trên cho thấy cấu trúc mà module của Contiki RPL. Contiki RPL bao gồm những module chính

- Neighbor: quản lý thông tin về các node trong phạm vi kết nối, đồng thời quản lý mối quan hệ trong cùng DODAG của node với các neighbor trong danh sách. Các mối quan hệ cùng DODAG gồm: parent, sibling, children. Những thông tin được theo dõi của mõi neighbor gồm: DODAG tham gia, DODAG rank, địa chỉ Ipv6 của mỗi neighbor trong mạng
- RPL-ICMP: module quản lý bản tin điều khiển RPL-ICMP gồm 3 loại bản tin chính: DIS, DIO, DAO. Đồng thời quản lý các chức năng kiểm tra xử lý các bản tin ICMP đầu vào, tạo và gửi các bản tin ICMP đầu ra:
- uip_rpl_input(void): kiểm tra bản tin đầu vào, phân loại bản tin RPL-ICMP và gọi các hàm xử lý tương ứng
- dis_input(void): hàm được gọi khi bản tin nhận được là DIS, xử lý các thông tin trong DIS nhận được và gọi những hàm xử lý cần thiết
- dis_output(void): tạo một bản tin DIS, sau đó gửi multicast hoặc unicast tời những node khác trong mạng
- dio_output(void): tạo mới một bản tin DIO và gửi tới những node khác trong mạng
- dio_input(void): hàm xử lý và lưu những thông tin trong bản tin DIO nhận được
- dao_output(void): tạo bản tin DAO, gửi tới những node parent trong DODAG. Bản tin DAO chỉ được gửi sau khi node đã là thành viên của một DODAG cụ thể
- dao_input(void): hàm kiểm tra và lưu những thông tin trong bản tin DAO nhận được từ các node khác trong mạng.
- Timer: là module quản lý những bộ định thời trong Contiki RPL, bao gồm 4 bộ định thời chính được xây dựng dựa trên những bộ định thời sẵn có của Contiki: periodic timer, dis timer, dio timer, dao timer.
- DODAG: module chính, quản lý các yếu tố ảnh hưởng đến quá trình tham gia, xây dựng và duy trì DODAG. Module này cung cấp 3 phương thức quản lý DODAG, xử lý thông tin, tìm kiếm, xây dựng bảng định tuyến
- DODAG_OF: cung cấp phương thức xác định Object function của DODAG mà node tham gia vào, cung cấp các phương thức so sánh, xác định vị trí và các thông số của node trong DODAG : rank, OCP, hàm lựa chọn best-parent->next-hop.
- Routing table: cung cấp các phương thức xây dựng bản tin định tuyến, lựa chọn default route, lọc các router mất kết nối.
- RPLSTAT: module được xây dựng phục vụ mục đích thống kê, lưu các thông số định tuyến như số lượng gói tin, số lượng các bản tin thất lạc. Qua đó, người sử dụng có thể đánh giá được một số những tiêu chí như Packet Delevery Rate (PDR), Packet Delevery Cost (PDC),…

Chương 4. Áp dụng mô phỏng bài báo khoa học

Cooja là một công cụ phổ biến được dùng trong nghiên cứu về IoT, cùng với hệ điều hành Contiki. Một trong những hướng nghiên cứu chủ yếu hiện nay là tối ưu hóa thuật toán Routing trong mạng nhằm mục đích tiết kiệm năng lượng và tài nguyên cho toàn mạng, đồ án này giới thiệu một bài báo khoa học về đề tài này: Learning-Based Resource Management for Low-Power and Lossy IoT Networks - https://ieeexplore.ieee.org/document/9717281

Trước khi mô phỏng ý tưởng của bài báo, em trình bày cơ bản nội dung của bài báo.

4.1. Abstract - Tổng quan

Trong phần tổng quan, bài báo nêu lên sự quan trọng của IoT trong thời đại 4.0. Một ứng dụng phổ biến của IoT được nhắc đến là truyền thông trong lưới điện thông minh. Trong mạng truyền thông cho lưới điện thì mặt hạn chế đấy là khả năng tính toán và năng lượng. Bên cạnh đó, truyền tin không dây giữa các thiết bị đi cùng với việc tỉ lệ mất mát gói bản tin, thông lượng thấp và bất bền vững. Để cung cấp một cơ chế truyền thông ổn định, một stack mạng IoT được đề xuất cho những thiết bị này. Tuy nhiên, những tầng trong stack này đều có một giới hạn của nó.

Một ví dụ đó là, để tạo điều kiện thực thi mạng công suất thấp và có tính mất mát (LLN) thì tổ chức đặc nhiệm kỹ thuật thế giới (IETF) chuẩn hóa giao thức tầng network và gọi nó là giao thức định tuyến cho mạng công suất thấp và có tính mất mát (RPLs), tuy nhiên, RPL thường không hiệu quả trong những mạng có mật độ node dày và điều kiện tải trên các node thay đổi. Những mạng IoT có mật độ dày trong tương lai được kỳ vọng sẽ tự động tối ưu hóa độ tin cậy và sự hiệu quả của truyền thông bằng cách quan tâm đến những đặc tính đa dạng của môi trường và cá quyết định từ devices. Học máy (ML) cung cấp một framework tin cậy cho môi trường mạng mật độ dày. Trong bài báo này, tác giả sử dụng một phương pháp học máy là Multiarmed bandit (MAB) và áp dụng nó cùng với kỹ thuật tính toán năng lượng mong đợi (BEEX), cung cấp các node khả năng tối ưu hóa hiệu quả sự vận hành của chúng.

Hệ thống được mô phỏng và đánh giá thông qua Contiki 3.0 Cooja simulation. Phương pháp này cải tiến năng lượng tiêu thụ và tỉ lệ gói bản tin được truyền, sử dụng ít tài nguyên hơn các cơ chế khác

4.2. Introduction - Giới thiệu

Tron phần giới thiệu, tác giả nêu định nghĩa của IoT, ứng dụng của IoT trong thời đại công nghiệp 4.0, như y tế thông mình, công nghiệp thông minh, lưới điện

thông minh, nhà thông minh, nông nghiệp thông minh. Trong một môi trường hạn chế về mặt tài nguyên, thực thi truyền thông thông minh và ổn định cho mạng IoT là một trong những khía cạnh trong thiết kế mạng IoT.

Một trong những ứng dụng phổ biến của IoT có tính diện rộng đấy là lưới điện thông minh. Một lưới điện thông minh nhắc đến mạng truyền thông của các cảm biến nằm trên đường lưới điện để đo, phân tích và truyền thông tin từ những thành phần công suất của lưới điện. Hạ tầng lưới điện thông minh nhằm mục đích quản lý công suất của lưới điện. Một ví dụ của lưới điện thông minh là hạ tầng công tơ thông minh (AMI). AMI phát triển một mạng truyền thông hai chiều giữa công tơ thông minh và vận hành trung tâm điều khiển để tự đông đo lường lượng điện sử dụng, thu thập thông tin dòng điện, điện áp, nhận diện giả mạo và mất cắp, kết nối và mất kết nối. AMI là một thành phần chính của một hệ thống lưới điện, với giat cả hợp lý, cải thiện năng lượng tiêu thụ, tin cậy và có hệ thống giám sát và điều khiển công suât.

Với một lượng lớn dữ liệu chia sẻ trong mạng AMI. cung cấp một công nghệ truyền thông chuẩn mực là một trong những vấn đề chính trong hệ thống IoT của lưới điện thông minh.

Tài nguyên hạn chế làm tăng một lượng lớn thách thức trong mạng LLN (được giới thiệu ở phần trước). Để giải quyết những thách thức đấy, cả phương pháp về phần mềm và phần cứng được đề xuất cho những thiết bị trong mạng. Tuy nhiên, cơ chế phần cứng làm chi phí hệ thống tăng cao vì phải thêm những thành phần phần cứng khác vào mạch. Bằng cách sử dụng những phương pháp bằng phần mềm, có thể thiết kế những giao thức mạng hiệu quả. Hơn nữa, những hệ điều hành IoT nhỏ gọn như Contiki, TinyOS, RIOT được phát triển để quản lý hiệu quả tài nguyên hạn chê.

RPL (được giới thiệu ở trên) được thiết kế để cung cấp khả năng kết nối giao thức mạng IPv6 cho các thiết bị LLN. Nó tạo một kiến truc định tuyến hình cây, gọi nó là Đồ thị không tuần hoàn có hướng hướng đến đích (Destination-oriented directed acyclic graph - DODAG). DODAG được xây dựng dựa trên một hàm mục tiêu nhất định (OF). Tổ chức IETF (được giới thiệu ở trên) chuẩn hóa hàm mục tiêu với rank thấp nhất và trễ (MRHOF), hàm dựa trên việc xem thông số ETX (số lần truyền mong đợi) như một thông số mặc định được định nghĩa trong RFC 6719 (RFC - Request for comments). Tương tự thế. hàm ÒF0 lại xem số bước nhảy là thông số định tuyến. Hàm OF nhằm mục tiêu tối ưu hóa thông số mạng cụ thể, như năng lượng, độ trễ và thông lượng. Rank dựa trên ETX cung cấp một số lượng xấp xỉ lần truyền gói bản tin được truyền thành công đến nút sink dựa trên chất lượng đường truyền (link quality) đối với nút đích. Trong MRHOF, chất lượng đường truyền được đánh giá bằng cách quảng bá những gói bản tin dò trong những khoảng thời gian. Những gói bản tin dò được quảng bá lại bởi những node nhận. Quá trình đánh giá đường truyền liên tục thế này sẽ làm cạn kiệt tài nguyên của node.

Tuy nhiên, cơ chế RPL tiêu chuẩn này không thể giải quyết bài toán mất cân bằng tải và phân bố năng lượng, đặc biết với các node ở gần node sink.
Học máy cung cấp một framework đáng tin cậy, chúng ta có thể thấy được những hệ thống IoT cao cấp có thể quản lý hiệu quả tài nguyên của nó với sữ hỗ trợ của học máy. Ví dụ, một trong những phương thức cấp phát tài nguyên được giới thiệu gần đây là cơ chế học xác suất va chạm thông minh (iCPLA). tối ưu hóa sự vận hành của tầng mạng sử dụng kỹ thuật Q-learning. Mặc dù thế, thuật toán Q-learning là không đủ trong những hệ thống mà ở đó xác suất thưởng (reward) có thể thay đổi theo thời gian khi có sự thay đổi trong điều kiện của mạng. Kỹ thuật iCPLA cũng không thể thay đổi lượng phần thưởng trong quá trình node đang học. Hơn nữa, Q-learning lấy mảng hai chiều cho các trạng thái và các hành động nên vì thế nó có độ phức tạp O(n)(a), sẽ dẫn đến độ trễ cao. Trong một thiết lập mà phân bố phần thưởng không cố định, nó sẽ có tính thực tế hơn nếu gán lượng phần thưởng vào phần thưởng hiện tại hơn là phần thưởng trong quá khứ. Điều này có thể lấy được bằng cách sử dụng một sơ đồ kẻ cắp trung bình lượng phần thưởng gần đây. Để giải quyết vấn đề của kỹ thuật Q-learning, nó được quan sát thông qua hệ mô phỏng Contiki OS cooja, vấn đề cấp phát tài nguyên của thiết bị IoT được giải quyết bằng kỹ thuật Học tăng cường (RL) đơn giản nhất như kiểu là kẻ cắp đa vũ trang (Multiarmed Bandit -MAB).

Trong bài báo này, tác giả thử nghiệm phương thức cơ bản áp dụng ML cho hệ thống AMI. Hơn nữa, tác giả hình thành bài toán học như một nhiệm vụ MAB để chọn chiến thuật tôts nhất cho từng đợt đo.
MAB là một bài toán học máy, trong đó người chơi cố gắng để lấy nhiều phần thưởng nhất từ một lượng máy slot machines. Sự vấn hành của tầng mạng của một hệ thống mạng lớn được tối ưu hóa bằng cách áp dụng cơ chế MAB sử dụng năng lương như một thông số học. Cơ chế được giới thiệu được tích hợp trên từng node và kiểm tra hồi quy năng lượng và trạng thái liên kết của chúng. Nó học và điều chỉnh hàm mục tiêu của mạng RPL cho các node trong LLN và cập nhật bảng định tuyến bằng cách sử dụng ít tài nguyên điều khiển nó. Phương pháp được giới thiệu trong bài báo được gọi là BEEX - bandit learning based expected energy count.

Những đóng góp chính của bài báo này được tóm tắt như sau:

- Tác giả giới thiệu mô hình kẻ cắp và hình thành hệ thống cho mạng chuẩn IEE 802.15.4 dựa trên mạng RPL. Nêu lên ảnh hưởng của cơ chế thằm dò ở tầng MAC đến tài nguyên của mạng và vấn đề mất cân bằng tải trong mạng RPL của cơ chế xếp hạng.
- Để xây dựng bảng định tuyến, tác giả giới thiệu phương pháp tính toán năng lượng mong đợi dựa trên cơ chế xếp hạng (EEX). Thông tin của EEX được nhúng trong gói bản tin điều khiển gọi là một đối tượng thộng tin DODAG (DIO). Sự truyền tải của gói tin DIO được dựa trên chơ chế trickle-timer (được đề xuất trong RFC 6206)

- Tác giả phát triển một framework kẻ cắp học Q-values để xây dựng bảng định tuyến thông minh.
- Cơ chế đề xuất kiểm tra môi trường mạng AMI không dây bằng cách sử dụng sơ đồ đánh cắp lượng phần thưởng trung bình gần đây theo cấp số nhân (ERWA) trong đó xác suất phần thưởng có thể thay đổi theo thời gian dựa trên sự thay đổi trong mạng. Trạng thái mạng có thể thay đổi dựa theo sự giao động của tải trong mạng hay sự thay đổi trong độ lớn của mạng với một node lỗi hay một node mới thêm vào mạng.
- Tác giả cải tiến sự vận thành của cơ chế trickle-timer trên các node bằng cách nén gói bản tin DIO trong giai đoạn khai thác (exploitation trong thuật toán của MAB). Phương pháp được đề xuất duy trì sự ổn định của mạng, thiết lập một DODAG hoặc bằng khám phá môi trường sử dụng thông tin EEX (exploration trong MAB) được nhúng trong gói bản tin DIO hoặc bằng khai thác môi trường bằng việc ước lượng Q-value
- Đánh giá sự thể hiện qua những chỉ số cơ bản và mở rộng

4.3. Background và các nghiên cứu liên quan

A. RPL

OF0 tìm đường định tuyến ngắn nhất đến node sink, không quan tâm đến điều kiện đường truyền. Nó chọn node parent có rank thấp nhất (về mặt khoảng cách) từ node sink.

Ngược lại, OF dựa trên ETX sử dụng đường định tuyến yêu cầu số lượng gói bản tin đến node sink phải truyền lại là ít nhất. Giá trị rank tăng đơn điệu từ node gốc đến các node con. Node gốc được rank là 1, đến node bước nhảy tiếp theo rank 2 và tiếp như thế. Rank được tính như sau:

Rank(c) = h + Rank(p) + rank_increase

Rank(c) : rank node con

h: khoảng cách 1 hop

rank_increase: đại lượng biến thiên giữa rank của node parent và rank của node con. Trong giao thức tiêu chuẩn thì rank_increase là ETX cho MRHOF và số lượng bước nhảy là thông số cho OF0.

ETX dùng 16 bit trong trường đối tượng điều khiển tài nguyên, 16 bit giá trị của ETX được làm tròn tới số gần nhất, ví dụ, nếu ETX=2.3 thì trường đối tượng có giá trị là 2.3 * 128 = 294.

Môiz node trong mạng DODAG duy trì một danh sách của node parents nhằm mục đích duy trì với mạng khi xảy ra lỗi .

DIO: DODAG infomation object: đã giới thiệu ở trên, được truyền tải từ node root.

DAO: DODAG advertisement object: đối tượng quảng bá DODAG

DIS: DODAG information solicitation: Thông tin huyên hoặc DODAG (thông tin để gia nhập mạng)

Một DODAG được xây dựng như sau: Đầu tiên, node sink truyền tải bản tin DIO chứa thông tin về rank được tính toán từ hàm mục tiêu. Sau đó, node nhận được tính toán rank của nó và quảng bá tới các node bên cạnh, cứ thế tiếp diễn cho đến khi tất cả các node trong mạng nhận được gói tin DIO.
Node con gửi thông tin của node đích đến cho node parent thông qua gói tin DAO. Còn nếu node bất kỳ nào đấy không nhận được gói tin DIO có thể yêu cầu tham gia mạng bằng cách gửi gói DIS
Hình dưới miêu tả cơ chế hình thành một DODAG:

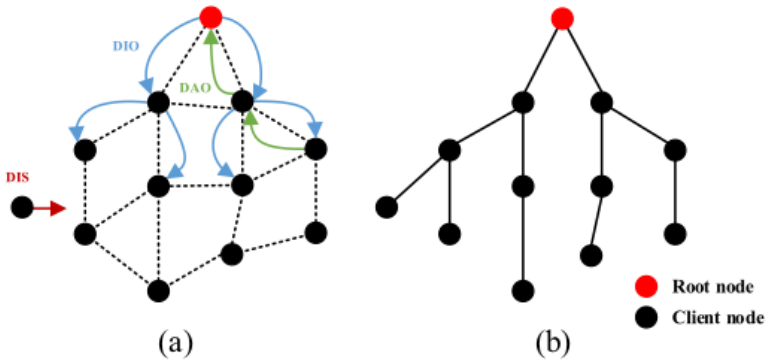


Fig. 1. RPL DAG with (a) control message sequence and (b) DODAG construction.

Hình 46:Cấu trúc RPL DAG với DODAG

Các node mạng IoT có hạn chế về năng lượng và tài nguyên, chính vì thế mà việc truyền tải các bản tin phải tối ưu hóa nhất có thể, không được truyền một cách thừa thãi. Cơ chế trickle-timer quản lý năng lượng bằng cách quản lý năng lượng tiêu thụ, quản lý tỉ lệ mà gói bản tin DIO được truyền. Nếu mạng đang ổn định, ít gói bản tin DIO sẽ được truyền đi, nó sẽ tăng tỷ lệ lên gấp đôi khi mạng không ổn định. Mạng sẽ không ổn định khi một node mới muốn tham vào mạng hay sẽ gãy liên kết giữa 2 node, khi đó cơ chế sẽ reset timer về giá trị nhỏ nhất để nhanh chóng cập nhật DODAG.

B. Các nghiên cứu liên quan
Các nghiên cứu liên quan bao gồm iCPLA, QU-RPL
iCPLA tối ưu hóa cross-layer sử dụng Q-learning. Phương pháp này sử dụng sự va chạm của tầng MAC để thể hiện quyết định của tầng mạng.
QU-RPL cải thiện sự tắc nghẽn và mất cân bằng tải, tuy nhiên hàm mục tiêu đề xuất lại không cân nhắc đến năng lượng tiêu thụ, điều đó có thể dẫn đến việc chọn node có năng lượng thấp.
Các nghiên cứu liên quan được đưa ra để có sự so sánh sau khi thực thi phương án đề xuất trong bài báo.

4.4. Đề xuất thực thi MAB-Learning cho việc bảo tồn năng lượng và sự mất mát gói bản tin của mạng IoT

Thuật toán đề xuất thuộc bài toán Học tăng cường trong học máy (Reinforcement Learning). Cơ chế của RL nhằm mục tiêu tìm phương án tốt nhất để đạt được một mục tiêu nhất định. Trong trường hợp mạng cảm biến, agent là từng sensor node học và đưa ra quyết định sau khi học một môi trường chưa biết.
Vấn đề đưa ra quyết định của Học tăng cường có thể là bài toán kẻ cắp hoặc bài toán quyết định dựa trên Markov (MDP). 1 MDP gồm có các trạng thái, trong từng trạng thái s, có một set các hành động a. Agent chọn hoạt động a ở trạng thái s và chuyển đến trạng thái s'. Một phần thưởng được cho trong quá trình chuyển trạng thái.
Nếu bài toán RL không phải là MDP, thì có thể giải quyết bằng kỹ thuật kẻ cắp. Ví dụ, trong trường hợp chỉ có 1 trạng thái hoặc các trạng thái đều tính, thì quá trình giảm xuống chỉ 1 set các hành động và phần thưởng dựa trên hành động được thực hiện và yếu tố giảm giá (discount factor). Trong tường hợp đấy, kỹ thuật kẻ cắp có thể sử dụng để giải quyêt bài toán RL. Nếu chỉ có một hành động, nó được gọi là kẻ cắp 1 tay (single-armed bandit), còn nếu agent phải chọn 1 hành động trong nhiều hành động.
Trong MAB, có một slot machine gồm K cánh tay gọi là các kẻ cắp. Mỗi cánh tay có xác suất thành công riêng của nó. Chọn từng cánh tay cho một phần thưởng tích cực (r+) nếu thành công và phần thưởng tiêu cực (r-) nếu thất bại. Mục tiêu là lấy từng tay theo một chuỗi thử nghiệm để tối đa phần thưởng tổng cộng cho mục tiêu dài hạn. Với từng hành động, có một giá trị Q(a) giúp agent cải thiện hành động trong tưởng lai. Giá trị Q(a) sẽ được cập nhật trong từng vòng lặp.
Agent hoặc khám phá (exploration) hoặc khai thác (exploitation) kinh nghiệm trước đó. Khai thác là quá trình đưa ra quyết định tốt nhất dựa trên dữ liệu có sẵn, trong đó khám phá chỉ đến quá trình lấy thêm dữ liệu. Trong quá trình học, phải có sự cần bằng giữa khai thác và khám phá. Khám phá quá nhiều có thể nhận một phần thưởng tiêu cực, và nếu khai thác quá nhiều có thể ngăn cản việc tối ưu phần thưởng một cách dài hạn. Sự đánh đổi khám phá-khai có thể lấy được bằng cách dùng kỹ thuật e-greedy. Giá trị của e giữa khoảng 0 và 1, ví dụ e=0.7 thuật toán sẽ áp dụng 70% cho khám phá và 30% dành cho quyết định.

Hai thuật toán sử dụng trong phương pháp:

Algorithm 1: BEEX Framework (Updating Reward and Q-Value)

1. Bước khởi tạo: Các nút (nodes) khởi tạo giá trị thưởng (reward) và giá trị Q (Q-value) toàn cục và sau đó lưu trữ giá trị thưởng tức thì (instant reward) và giá trị thưởng tích lũy (cumulative reward) cho tất cả các hành động.
2. Vòng lặp chính (while): Trong khi thiết bị đang hoạt động, thực hiện các bước sau:

3. Đặt giới hạn số lần thử lại tối đa = 3.
4. Đặt số lượng giai đoạn (back-off stages) tối đa = 5.
5. Đặt giá trị CWmin = 0, CWmax = 31. (CW là viết tắt của "Contention Window" - cửa sổ cạnh tranh, được sử dụng trong các giao thức truyền dẫn không dây để điều chỉnh quy trình xung đột và truyền dẫn dữ liệu.)
6. Đặt giá trị thưởng hiện tại (current reward) = 0, giá trị Qn(a) = 0, giá trị Qn+1(a) = 0. (Qn(a) và Qn+1(a) là giá trị Q tại hai thời điểm liên tiếp.)
7. Tính toán ETX bằng cách sử dụng hàm neighbor_link_callback(). (ETX là viết tắt của "Expected Transmission Count" - số lần truyền dự kiến, được sử dụng để ước tính khả năng truyền dữ liệu từ một nút đến một nút láng giềng trong mạng không dây.)
8. Tính toán EEX bằng cách sử dụng new_ETX trong (10). (EEX là viết tắt của "Expected Energy Consumption" - lượng năng lượng tiêu thụ dự kiến, được sử dụng để ước tính lượng năng lượng tiêu thụ khi truyền dữ liệu từ một nút đến một nút láng giềng trong mạng không dây.)
9. Nếu (EEX(current) = EEX(Previous)), thực hiện các bước sau:
10. Giá trị thưởng (reward) = r+.
11. Ngược lại, nếu (EEX(current) > EEX(Previous)), thực hiện các bước sau:
12. Giá trị thưởng (reward) = r-.
13. Cập nhật bảng giá trị thưởng (reward table) cho hành động a.
14. Cập nhật bảng giá trị Q (Q-values table) theo (13).
15. Kết thúc câu lệnh điều kiện.
16. Kết thúc vòng lặp chính (while).

Thuật toán đang thực hiện việc cập nhật giá trị thưởng và giá trị Q dựa trên sự thay đổi của ETX và EEX. Giá trị thưởng được cập nhật dựa trên sự so sánh giữa EEX hiện tại và EEX trước đó, trong khi bảng giá trị Q được cập nhật dựa trên quy tắc cụ thể.

Sau khi đã có được thông tin cập nhật của Q và các giá trị ETX, EEX, thuật toán 2 tiến hành khởi tạo mạng, thiết lập bộ định thời để truyền gói bản tin DIO:
Algorithm 2: BEEX Framework (Performing Action)
1. Khởi tạo các tham số của thuật toán.
2. Đặt I = Imin (một giá trị cố định), c = 0 (biến đếm).
3. Tạo một khoảng thời gian mới.
4. Đặt I = I × 2 (nhân I với 2).
5. Kiểm tra nếu Imax (một giá trị cố định) nhỏ hơn hoặc bằng I, thì đặt I = Imax.
6. Kiểm tra nếu "exploitation" (khai thác) được kích hoạt, tìm địa chỉ IP có minQn+1(a) và loại bỏ DIO (Data Input/Output).
7. Kiểm tra nếu "exploration" (khám phá) được kích hoạt. Nếu node là root node, đặt root rank = 1. Nếu parent là null, đặt rank = max path cost. Nếu parent khác null, tính rank theo công thức rank = h + Rank(pi) + rankincrease, với h là một giá trị cố định và rankincrease là một giá trị được tính dựa trên EEX.

8. Nếu Rank(pi) = 0, đặt rank = base rank (128).
9. Trả về giá trị MIN (Base rank + rankincrease).
10. Nhúng rankincrease vào DIO.
11. Đặt ttimer = ngẫu nhiên trong khoảng [I/2, I].
12. Kiểm tra nếu mạng có tính nhất quán. Nếu có, tăng biến đếm lên 1. Nếu không, đặt I = Imin. Nếu ttimer hết hạn, gửi DIO tới tất cả các node trong mạng.

4.5. Kết quả thực nghiệm và thảo luận

Bài báo tiến hành mô phỏng trên môi trường Contiki OS Cooja simulation với các hàm mục tiêu khác nhau để so sánh với hàm mục tiêu được giới thiệu trong bài báo, với một mạng gồm có 1 node root và có 20-100 node lá, với các thông số của mạng trong bảng sau:

TABLE VI
SIMULATION PARAMETERS

Parameter type	Value
DAG size	20–100
Run time	3600s
Buffer occupancy	4 packets
Packet size	127 bytes
PHY & MAC	802.15.4
Uip payload size	140 bytes
Mote	Z1
Maximum retry limits	3
Maximum backoff stages	5
I_{min}	10
I_{max}	8 doublings
RPL_min_hoprankinc (h)	128
Packets transmission	Variable
Script testlog analysis	Python 3.8

Hình 47:Bảng thông số đưa vào mô phỏng

Qua đó tác giả đánh giá, so sánh phương pháp đề xuất trong bài báo với các phương pháp khác, qua các biểu đồ sau:

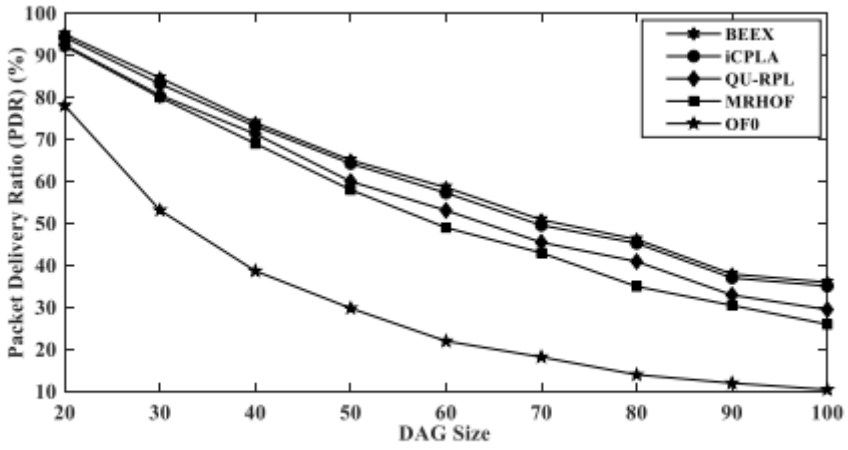


Fig. 4. PDR (%) in different DAG-sized networks.

Hình 48: Biểu đồ tỷ lệ gói tin truyền được trong các mạng DAG có kích cỡ khác nhau

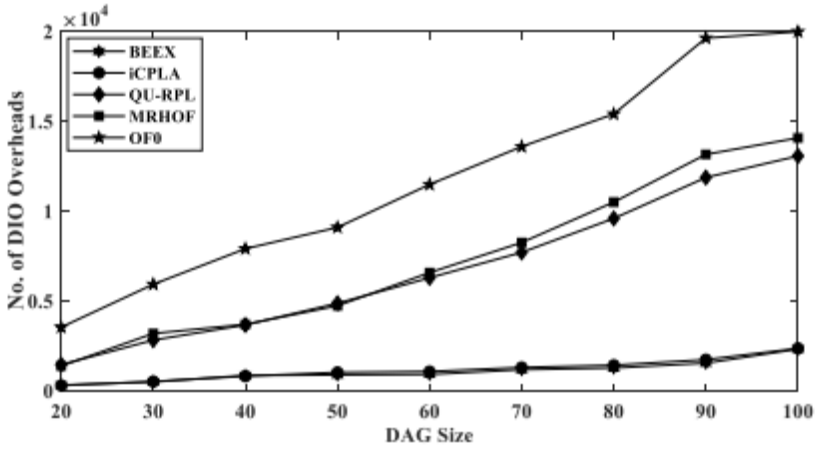


Fig. 5. Number of DIO control overheads in different DAG-sized networks.

Hình 49:Biểu đồ số lượng chi phí điều khiển dio trong các mạng có kích thước DAG khác nhau

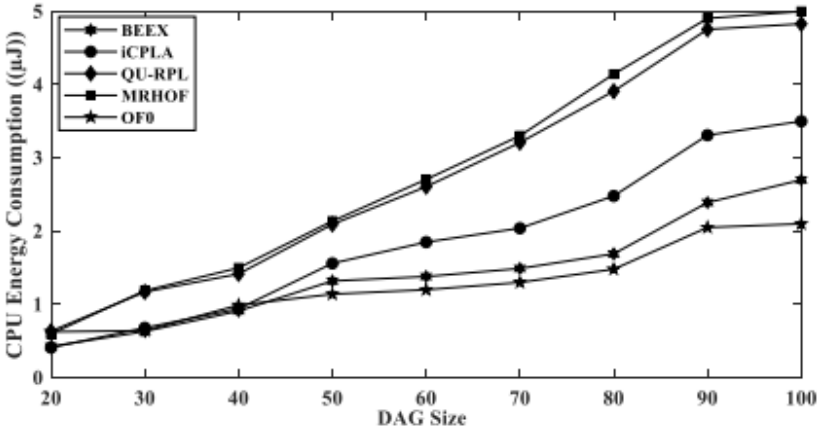


Fig. 6. Total CPU energy consumption in different DAG-sized networks.

Hình 50:Biểu đồ tổng năng lượng tiêu thụ của CPU trong các mạng DAG có kích thước khác nhau

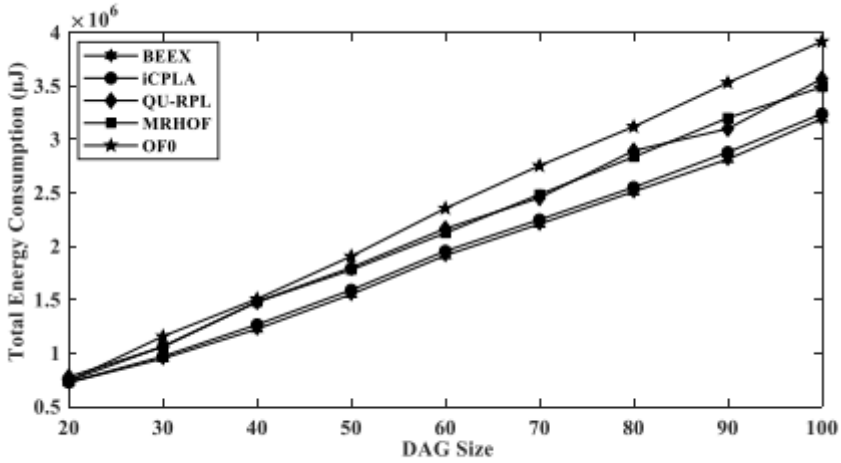


Fig. 7. Total energy consumption in different DAG-sized networks.

Hình 51:Biểu đồ tổng năng lượng tiêu thụ của các mạng DAG có kích thức khác nhau

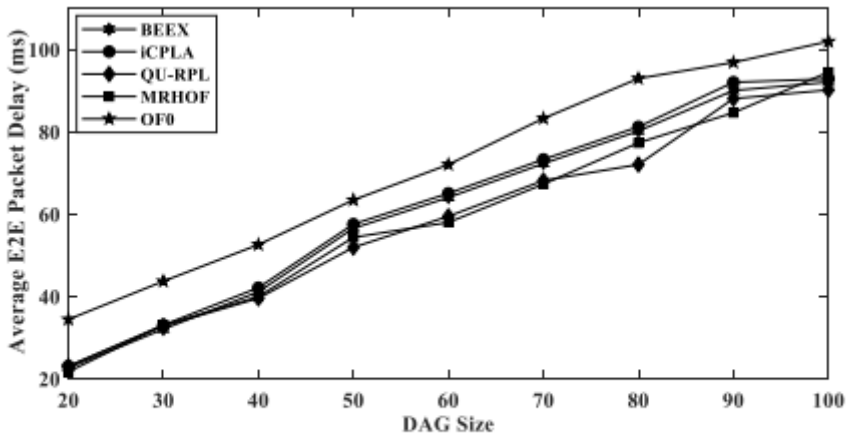


Fig. 8. Average end-to-end (E2E) packet delay (ms) in different DAG-sized networks.

Hình 52:Độ trễ của gói tin E2E trong các mạng DAG có kích thước khác nhau

Qua đó cho thấy:

-) Cơ chế đề xuất (proposed mechanism) và cho thấy rằng cơ chế này cải thiện hiệu suất mạng, đo bằng PDR (Packet Delivery Ratio - tỷ lệ gói tin được nhận thành công bởi nút thu).

-) PDR được so sánh giữa cơ chế đề xuất và các cơ chế khác như MRHOF, OF0, QU-RPL và iCPLA. MRHOF và QU-RPL có PDR tốt hơn OF0. Cơ chế đề xuất sử dụng BEEX (Bandit-based Energy Expended) thông minh hơn iCPLA.

-) Cơ chế đề xuất giảm thiểu mất mát dữ liệu nhờ việc lựa chọn đường truyền nhanh chóng dựa trên việc học nhanh.

-) Cơ chế đề xuất giảm thiểu chi phí điều khiển DIO (DODAG Information Object) hơn các cơ chế khác như QU-RPL, MRHOF và OF0.

-) Cơ chế đề xuất sử dụng ít năng lượng CPU hơn iCPLA do tính phức tạp thấp hơn.

-) Cơ chế đề xuất giảm tổng lượng tiêu thụ năng lượng của các nút so với các giao thức khác như OF0, MRHOF, QU-RPL và iCPLA.

-) Độ trễ giao hàng từ đầu đến cuối (E2E) của cơ chế đề xuất thấp hơn iCPLA, OF0 và gần tương đương với MRHOF và QU-RPL.

Tổng quan, cơ chế đề xuất (BEEX) được cho là cải thiện hiệu suất mạng IoT bằng cách giảm thiểu mất mát dữ liệu, giảm chi phí điều khiển và tiêu thụ năng lượng, đồng thời cung cấp độ trễ giao hàng tốt hơn.

4.6. Mô phỏng mô hình đánh giá năng lượng tiêu thụ trên các node trong bài báo

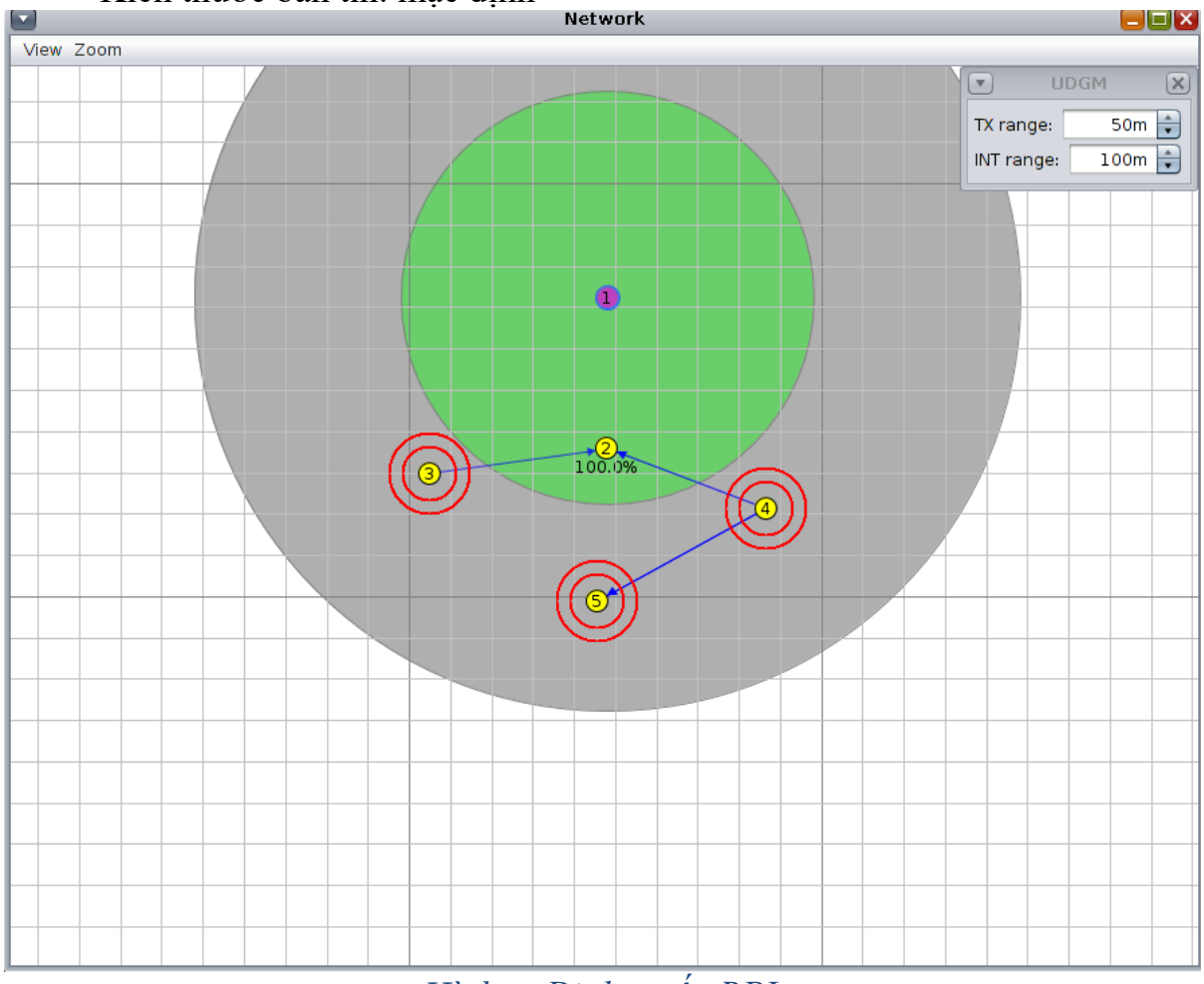
a. Xây dựng mạng 5 node

Mục tiêu mô phỏng:

- Sử dụng mô phỏng cooja, hiểu được các tính năng, tool hỗ trợ
- Thu thập được dữ liệu thông qua collect view

Tham số cài đặt:

- Thời gian mô phỏng: 6000s
- Diện tích mô phỏng: 40000m2
- Cự ly giữa các node: nhỏ hơn 50m so với node parent của nó
- Node mô phỏng Zolertia z1
- Số lượng node 4 node: 1 sink và 3 client
- Vị trí node: phân bố ngẫu nhiên nhưng phải đảm bảo khoảng cách giữa các node nhỏ hơn 50m
- Kích thước bản tin: mặc định
- Thời gian gửi bản tin data: mặc định
- Kích thước bản tin: mặc định



Hình 53:Định tuyến RPL

File Edit View

Time	Mote	Message
08:50.949	ID:1	30 55084 1566 0 4626 8 1 0 22 2344 0 4461 42072 77 480 257 128 512 10 1048 119 1 56 49 65535 65535 0 0 0 0
08:52.557	ID:1	30 55084 1567 0 8995 8 4 0 22 2530 0 3366 40742 162 371 8738 539 1353 5 524 35 1 228 221 65535 65535 0 0 0 0
08:54.030	ID:1	30 55084 1569 0 2827 8 2 0 22 2686 0 3553 43073 59 442 4626 256 913 6 1048 63 1 0 249 65535 65535 0 0 0 0
08:57.235	ID:1	30 55084 1572 0 6939 8 2 0 22 3123 0 4571 37006 430 577 5397 256 793 11 1048 126 1 63 112 65535 65535 0 0 0 0
08:58.243	ID:1	30 55084 1573 0 1285 8 5 0 22 3142 0 4568 61820 157 544 6682 697 1729 3 524 112 1 49 42 65535 65535 0 0 0 0
09:03.736	ID:1	30 55084 1579 0 771 9 3 0 22 3832 0 6134 56300 301 667 6939 402 1085 11 524 117 1 54 103 65535 65535 0 0 0 0
09:04.934	ID:1	Rime started with address 193.12.0.0.0.0.0.1
09:04.941	ID:1	MAC c1:0c:00:00:00:00:00:01 Contiki 3.0 started. Node id is set to 1.
09:04.946	ID:1	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
09:04.954	ID:1	Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0001
09:04.958	ID:1	Starting 'UDP server process' 'collect common process'
09:04.959	ID:1	I am sink!
09:04.960	ID:1	UDP server started
09:04.963	ID:1	created a new RPL dag
09:04.966	ID:1	Server IPv6 addresses: aaaa::c30c:0:0:1
09:04.967	ID:1	aaaa::1
09:04.969	ID:1	fe80::c30c:0:0:1
09:04.975	ID:1	Created a server connection with remote address :: local/remote port 5688/8775
09:05.171	ID:2	Rime started with address 193.12.0.0.0.0.0.2
09:05.178	ID:2	MAC c1:0c:00:00:00:00:00:02 Contiki 3.0 started. Node id is set to 2.
09:05.183	ID:2	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
09:05.191	ID:2	Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0002
09:05.195	ID:2	Starting 'UDP client process' 'collect common process'
09:05.197	ID:2	UDP client process started
09:05.201	ID:2	Client IPv6 addresses: aaaa::c30c:0:0:2
09:05.202	ID:2	fe80::c30c:0:0:2
09:05.207	ID:2	Created a connection with the server :: local/remote port 8775/5688
09:05.317	ID:3	Rime started with address 193.12.0.0.0.0.0.3
09:05.324	ID:3	MAC c1:0c:00:00:00:00:00:03 Contiki 3.0 started. Node id is set to 3.
09:05.329	ID:3	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
09:05.337	ID:3	Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0003
09:05.341	ID:3	Starting 'UDP client process' 'collect common process'
09:05.343	ID:3	UDP client process started
09:05.346	ID:3	Client IPv6 addresses: aaaa::c30c:0:0:3
09:05.348	ID:3	fe80::c30c:0:0:3
09:05.353	ID:3	Created a connection with the server :: local/remote port 8775/5688
09:05.430	ID:4	Rime started with address 193.12.0.0.0.0.0.4
09:05.438	ID:4	MAC c1:0c:00:00:00:00:00:04 Contiki 3.0 started. Node id is set to 4.
09:05.443	ID:4	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
09:05.450	ID:4	Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0004
09:05.454	ID:4	Starting 'UDP client process' 'collect common process'
09:05.457	ID:4	UDP client process started
09:05.460	ID:4	Client IPv6 addresses: aaaa::c30c:0:0:4
09:05.462	ID:4	fe80::c30c:0:0:4
09:05.467	ID:4	Created a connection with the server :: local/remote port 8775/5688

Hình 54:Mote output

b. Đánh giá năng lượng tiêu thụ bằng module energest trong ContikiOS

ContikiOS cho phép đánh giá năng lượng tiêu thụ của các node trong mạng bằng việc sử dụng module Energest, dưới đây là các bước lập trình sử dụng module Energest trong ContikiOS và được mô phỏng trên Cooja.

-) B1: Điều chỉnh make file

```
Makefile M ×
examples > rpl-udp > Makefile
1   CONTIKI_PROJECT = udp-client udp-server
2   all: $(CONTIKI_PROJECT)
3
4   CONTIKI=../..
5
6   include $(CONTIKI)/Makefile.dir-variables
7   MODULES += $(CONTIKI_NG_SERVICES_DIR)/simple-energest
8
9   include $(CONTIKI)/Makefile.include
10
```

Hình 55: Điều chỉnh Makefile

-) B2: Thêm Process năng lượng tiêu thụ vào process rpl-udp

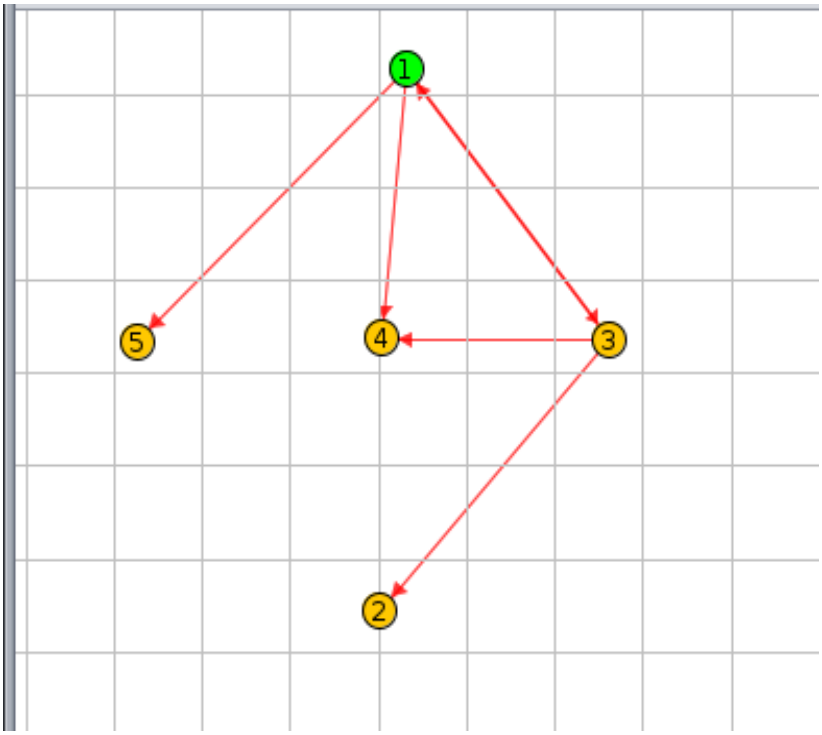
```
/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP client");
PROCESS(example_process, "Example process: simple-energest");
AUTOSTART_PROCESSES(&example_process
                    ,&udp_client_process);
/*---------------------------------------------------------------------------*/
static void
PROCESS_THREAD(example_process, ev, data)
{
  PROCESS_BEGIN();

  /* Do nothing, just let simple-energest write its summary
   * at a period of SIMPLE_ENERGEST_CONF_PERIOD */

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

Hình 56: Thêm Process năng lượng

```
simple_energest_step(void)
{
  static unsigned count = 0;
  uint64_t curr_tx, curr_rx, curr_time, curr_cpu, curr_lpm, curr_deep_lpm;
  uint64_t delta_time;

  energest_flush();

  curr_time = ENERGEST_GET_TOTAL_TIME();
  curr_cpu = energest_type_time(ENERGEST_TYPE_CPU);
  curr_lpm = energest_type_time(ENERGEST_TYPE_LPM);
  curr_deep_lpm = energest_type_time(ENERGEST_TYPE_DEEP_LPM);
  curr_tx = energest_type_time(ENERGEST_TYPE_TRANSMIT);
  curr_rx = energest_type_time(ENERGEST_TYPE_LISTEN);

  delta_time = MAX(curr_time - last_time, 1);

  LOG_INFO("--- Period summary #%u (%"PRIu64" seconds)\n",
           count++, delta_time / ENERGEST_SECOND);
  LOG_INFO("Total time  : %10"PRIu64"\n", delta_time);
  log_energest("CPU", curr_cpu - last_cpu, delta_time);
  log_energest("LPM", curr_lpm - last_lpm, delta_time);
  log_energest("Deep LPM", curr_deep_lpm - last_deep_lpm, delta_time);
  log_energest("Radio Tx", curr_tx - last_tx, delta_time);
  log_energest("Radio Rx", curr_rx - last_rx, delta_time);
  log_energest("Radio total", curr_tx - last_tx + curr_rx - last_rx,
               delta_time);

  last_time = curr_time;
  last_cpu = curr_cpu;
  last_lpm = curr_lpm;
  last_deep_lpm = curr_deep_lpm;
  last_tx = curr_tx;
  last_rx = curr_rx;
}
```

Hình 57: Một phần trong Process Energest

-) B3: Tiến hành mô phỏng trên Cooja



Hình 58: Topo mạng trong Cooja

```
06:01.618  ID:5   [INFO: Energest  ] --- Period summary #5 (59 seconds)
06:01.625  ID:5   [INFO: Energest  ] Total time  :    1966079
06:01.637  ID:5   [INFO: Energest  ] CPU         :       6143/   1966079 (3 permil)
06:01.651  ID:5   [INFO: Energest  ] LPM         :    1959936/   1966079 (996 permil)
06:01.661  ID:5   [INFO: Energest  ] Deep LPM    :          0/   1966079 (0 permil)
06:01.672  ID:5   [INFO: Energest  ] Radio Tx    :        179/   1966079 (0 permil)
06:01.687  ID:5   [INFO: Energest  ] Radio Rx    :    1965895/   1966079 (999 permil)
06:01.702  ID:5   [INFO: Energest  ] Radio total :    1966074/   1966079 (999 permil)
06:01.802  ID:3   [INFO: Energest  ] --- Period summary #5 (60 seconds)
06:01.809  ID:3   [INFO: Energest  ] Total time  :    1966080
06:01.821  ID:3   [INFO: Energest  ] CPU         :       5992/   1966080 (3 permil)
06:01.836  ID:3   [INFO: Energest  ] LPM         :    1960088/   1966080 (996 permil)
06:01.846  ID:3   [INFO: Energest  ] Deep LPM    :          0/   1966080 (0 permil)
06:01.857  ID:3   [INFO: Energest  ] Radio Tx    :        159/   1966080 (0 permil)
06:01.871  ID:3   [INFO: Energest  ] Radio Rx    :    1965920/   1966080 (999 permil)
06:01.886  ID:3   [INFO: Energest  ] Radio total :    1966079/   1966080 (999 permil)
06:09.674  ID:4   [INFO: App       ] Sending request 5 to fd00::c30c:0:0:1
06:09.702  ID:1   [INFO: App       ] Received request 'hello 5' from fd00::c30c:0:0:4
06:09.705  ID:1   [INFO: App       ] Sending response.
06:09.751  ID:4   [INFO: App       ] Received response 'hello 5' from fd00::c30c:0:0:1
06:19.569  ID:2   [INFO: App       ] Sending request 5 to fd00::c30c:0:0:1
06:19.631  ID:1   [INFO: App       ] Received request 'hello 5' from fd00::c30c:0:0:2
06:19.634  ID:1   [INFO: App       ] Sending response.
06:19.718  ID:2   [INFO: App       ] Received response 'hello 5' from fd00::c30c:0:0:1
06:43.100  ID:3   [INFO: App       ] Sending request 6 to fd00::c30c:0:0:1
06:43.151  ID:1   [INFO: App       ] Received request 'hello 6' from fd00::c30c:0:0:3
06:43.154  ID:1   [INFO: App       ] Sending response.
06:43.204  ID:3   [INFO: App       ] Received response 'hello 6' from fd00::c30c:0:0:1
06:59.806  ID:5   [INFO: App       ] Sending request 6 to fd00::c30c:0:0:1
06:59.834  ID:1   [INFO: App       ] Received request 'hello 6' from fd00::c30c:0:0:5
06:59.837  ID:1   [INFO: App       ] Sending response.
06:59.860  ID:5   [INFO: App       ] Received response 'hello 6' from fd00::c30c:0:0:1
```

Hình 59: Mote Output cho mạng

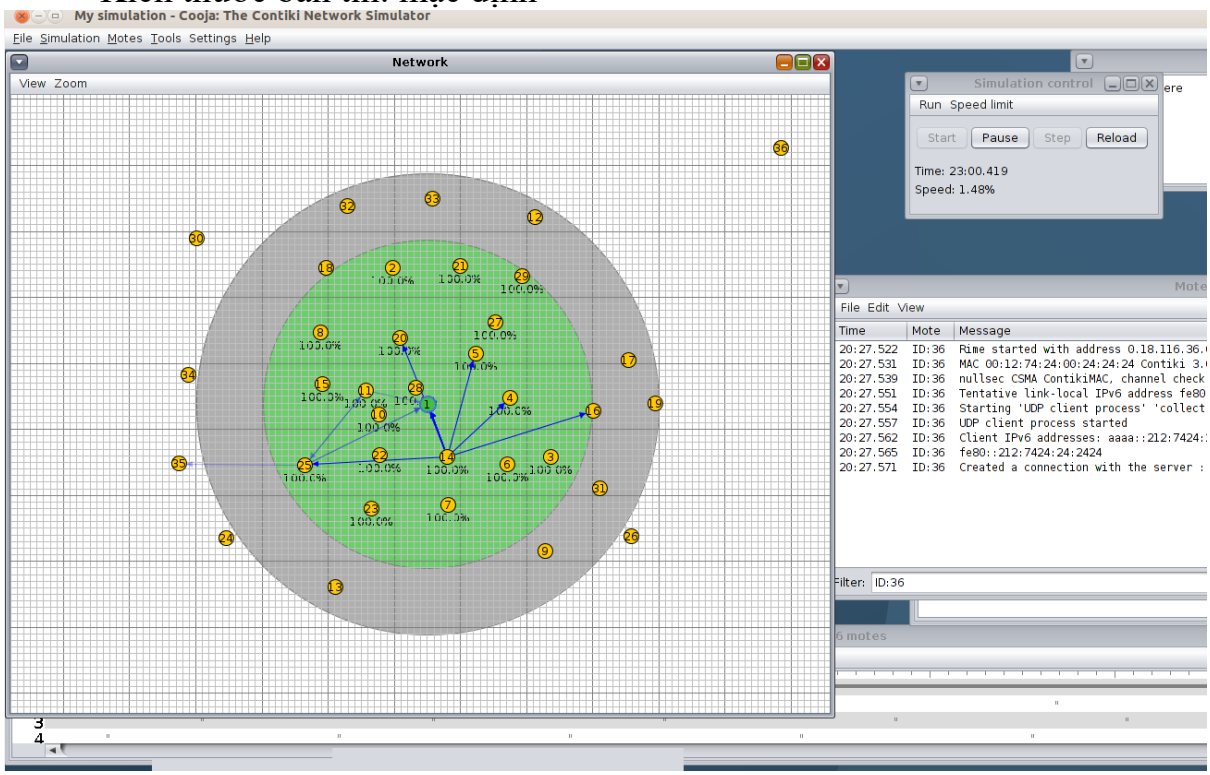
c. Xây dựng mạng với số node lớn hơn

Mục tiêu mô phỏng:

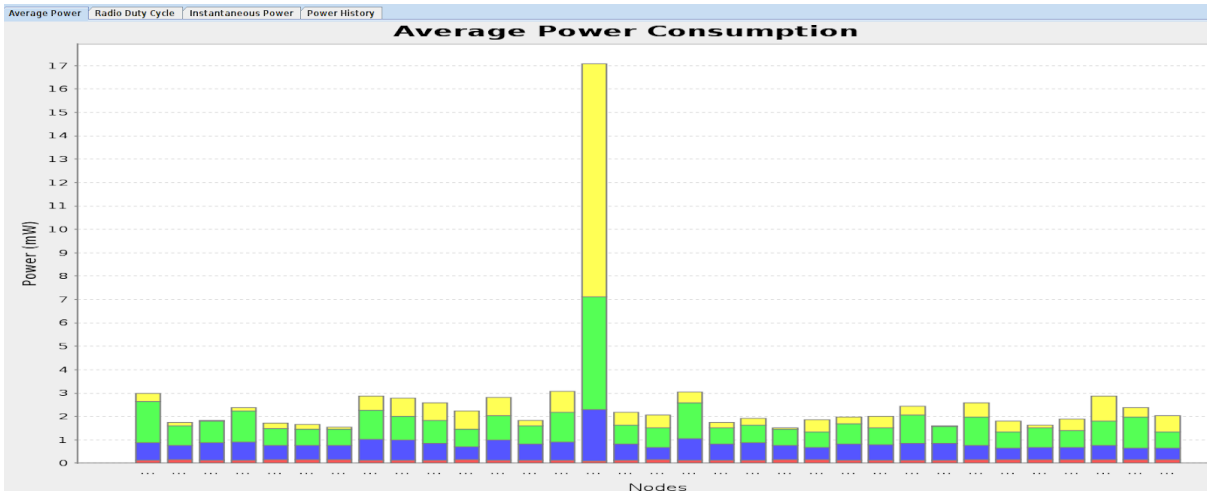
- Sử dụng mô phỏng cooja, hiểu được các tính năng, tool hỗ trợ
- Thu thập được dữ liệu thông qua collect view: thông số về năng lượng, tỷ lệ bản tin lỗi, đánh giá mức tiêu thụ điện năng,…

Tham số cài đặt:

- Thời gian mô phỏng: 6000s
- Diện tích mô phỏng: 100000m2
- Cự ly giữa các node: nhỏ hơn 50m so với node parent của nó
- Node mô phỏng Zolertia z1
- Số lượng node 35 node: 1 sink và 34 client
- Vị trí node: phân bố ngẫu nhiên nhưng phải đảm bảo khoảng cách giữa các node nhỏ hơn 50m
- Kích thước bản tin: mặc định
- Thời gian gửi bản tin data: mặc định
- Kích thước bản tin: mặc định



Hình 60: Topology mạng



Hình 61:Năng lượng tiêu thụ các node trong mạng

Sensor Data Collect with Contiki (connected to <stdin>)

File Tools

Nodes: <All>, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.10, 11.11, 12.12, 13.13, 14.14, 15.15, 16.16, 17.17, 18.18, 19.19, 20.20, 21.21, 22.22, 23.23, 24.24, 25.25, 26.26, 27.27, 28.28, 29.29, 30.30, 31.31, 32.32, 33.33, 34.34, 35.35

Node Control | Sensor Map | Network Graph | Sensors | Network | Power | Node Info | Serial Console

Node	Received	Dups	Lost	Hops	Rtmetric	ETX	Churn	Beacon Interval	Reboots	CPU Power	LPM Power	Listen Power
1.1	0	0	0	0.000	0.000	0.000	0		0	0.000	0.000	0.000
2.2	2	0	0	1.000	537.500	16....	0	3 min, 16 sec	0	0.729	0.141	1.774
3.3	2	0	0	1.000	512.000	16....	0	1 min, 38 sec	0	0.609	0.145	0.849
4.4	2	0	0	1.000	542.000	16....	0	34 min, 57 sec	0	0.730	0.141	0.945
5.5	1	0	0	1.000	512.000	16....	0	34 min, 57 sec	0	0.761	0.140	1.334
6.6	2	0	0	1.000	554.500	16....	0	2 min, 11 sec	0	0.630	0.144	0.716
7.7	2	0	0	1.000	512.000	16....	0	2 min, 11 sec	0	0.628	0.144	0.687
8.8	2	0	0	1.000	550.000	16....	0	2 min, 11 sec	0	0.612	0.145	0.700
9.9	0	0	0	0.000	0.000	0.000	0		0	0.000	0.000	0.000
10.10	1	0	0	1.000	534.000	16....	0	1 min, 05 sec	0	0.881	0.137	1.253
11.11	2	0	0	2.000	787.500	32....	0	1 min, 38 sec	0	0.844	0.138	1.015
12.12	1	0	0	2.000	819.000	32....	0	1 min, 05 sec	0	0.698	0.142	0.990
13.13	2	0	0	2.000	817.500	32....	1	1 min, 38 sec	0	0.568	0.146	0.738
14.14	1	0	0	1.000	537.000	16....	0	1 min, 05 sec	0	0.863	0.137	1.023
15.15	2	0	0	1.000	512.000	16....	0	2 min, 11 sec	0	0.684	0.143	0.781
16.16	2	0	0	2.000	768.000	32....	0	1 min, 21 sec	0	0.765	0.140	1.264
17.17	1	0	0	2.000	859.000	32....	0	0 min, 32 sec	0	2.186	0.097	4.829
18.18	2	0	0	2.000	799.500	32....	0	2 min, 11 sec	0	0.669	0.143	0.817
19.19	2	0	0	3.000	1093.0...	52....	0	3 min, 16 sec	0	0.520	0.148	0.856
20.20	1	0	0	1.000	512.000	16....	0	1 min, 05 sec	0	0.930	0.135	1.515
21.21	2	0	0	1.000	531.000	16....	0	2 min, 43 sec	0	0.689	0.143	0.695
22.22	2	0	0	1.000	512.000	16....	0	2 min, 43 sec	0	0.732	0.141	0.756
23.23	2	0	0	1.000	543.000	16....	0	1 min, 38 sec	0	0.631	0.144	0.691
24.24	2	0	0	2.000	788.000	33....	0	1 min, 38 sec	0	0.536	0.147	0.661
25.25	2	0	0	1.000	531.500	16....	0	2 min, 11 sec	0	0.691	0.143	0.843
26.26	2	0	0	2.000	839.000	32....	0	2 min, 43 sec	0	0.650	0.144	0.722
27.27	1	0	0	1.000	546.000	16....	0	1 min, 05 sec	0	0.717	0.142	1.199
28.28	2	0	0	1.000	512.000	16....	0	34 min, 57 sec	0	0.706	0.142	0.716
29.29	1	0	0	2.000	817.000	32....	0	1 min, 05 sec	0	0.625	0.145	1.220
30.30	2	0	0	3.000	1103.0...	49....	0	1 min, 38 sec	0	0.489	0.149	0.717
31.31	2	0	0	2.000	823.000	32....	0	34 min, 57 sec	0	0.518	0.148	0.851
32.32	2	0	0	2.000	797.000	33....	0	3 min, 16 sec	0	0.530	0.147	0.728
33.33	1	0	0	2.000	793.000	33....	0	0 min, 08 sec	0	0.624	0.145	1.049
34.34	2	0	0	2.000	842.500	32....	0	3 min, 16 sec	0	0.491	0.149	1.334
35.35	2	0	0	2.000	782.000	32....	0	3 min, 16 sec	0	0.507	0.148	0.698
Avg	**1.727**	**0.000**	**0.000**	**1.545**	**682.379**	**25....**	**0.030**	**5 min, 55 sec**	**0.000**	**0.710**	**0.142**	**1.060**

Hình 62:Bảng tổng hợp các thông số mạng (1)

Transmit Power	Power	On-time	Listen Duty Cycle	Transmit Duty Cycle	Avg Inter-packet Time	Min Inter-packet Time	Max Inter-packet Time
0.000	0.000		0.000	0.000			
0.338	2.982	0 min,...	2.957	0.637	0 min, 30 sec	1 min, 01 sec	1 min, 01 sec
0.148	1.751	0 min,...	1.415	0.279	0 min, 24 sec	0 min, 48 sec	0 min, 48 sec
0.008	1.824	0 min,...	1.575	0.014	0 min, 06 sec	0 min, 13 sec	0 min, 13 sec
0.160	2.396	0 min,...	2.224	0.302			
0.218	1.709	0 min,...	1.193	0.411	0 min, 15 sec	0 min, 31 sec	0 min, 31 sec
0.207	1.667	0 min,...	1.145	0.391	0 min, 24 sec	0 min, 48 sec	0 min, 48 sec
0.092	1.549	0 min,...	1.167	0.173	0 min, 16 sec	0 min, 33 sec	0 min, 33 sec
0.000	0.000		0.000	0.000			
0.609	2.880	0 min,...	2.089	1.148			
0.791	2.788	0 min,...	1.692	1.489	0 min, 21 sec	0 min, 42 sec	0 min, 42 sec
0.739	2.570	0 min,...	1.650	1.392			
0.779	2.231	0 min,...	1.230	1.467	0 min, 19 sec	0 min, 39 sec	0 min, 39 sec
0.786	2.809	0 min,...	1.705	1.480			
0.228	1.836	0 min,...	1.301	0.430	0 min, 19 sec	0 min, 39 sec	0 min, 39 sec
0.920	3.089	0 min,...	2.106	1.733	0 min, 37 sec	1 min, 15 sec	1 min, 15 sec
9.963	17.0...	0 min,...	8.049	18.763			
0.559	2.188	0 min,...	1.361	1.052	0 min, 16 sec	0 min, 32 sec	0 min, 32 sec
0.533	2.056	0 min,...	1.426	1.004	0 min, 23 sec	0 min, 47 sec	0 min, 47 sec
0.480	3.060	0 min,...	2.525	0.904			
0.223	1.750	0 min,...	1.159	0.420	0 min, 38 sec	1 min, 17 sec	1 min, 17 sec
0.282	1.911	0 min,...	1.259	0.531	0 min, 42 sec	1 min, 24 sec	1 min, 24 sec
0.047	1.514	0 min,...	1.152	0.089	0 min, 29 sec	0 min, 58 sec	0 min, 58 sec
0.518	1.863	0 min,...	1.102	0.976	0 min, 20 sec	0 min, 41 sec	0 min, 41 sec
0.309	1.986	0 min,...	1.405	0.581	0 min, 15 sec	0 min, 30 sec	0 min, 30 sec
0.501	2.016	0 min,...	1.203	0.943	0 min, 47 sec	1 min, 35 sec	1 min, 35 sec
0.394	2.451	0 min,...	1.998	0.742			
0.030	1.594	0 min,...	1.193	0.057	0 min, 12 sec	0 min, 24 sec	0 min, 24 sec
0.586	2.577	0 min,...	2.034	1.104			
0.446	1.801	0 min,...	1.195	0.840	0 min, 32 sec	1 min, 05 sec	1 min, 05 sec
0.122	1.639	0 min,...	1.418	0.231	0 min, 16 sec	0 min, 32 sec	0 min, 32 sec
0.474	1.879	0 min,...	1.214	0.893	0 min, 37 sec	1 min, 15 sec	1 min, 15 sec
1.046	2.863	0 min,...	1.748	1.969			
0.415	2.390	0 min,...	2.224	0.782	0 min, 22 sec	0 min, 45 sec	0 min, 45 sec
0.691	2.044	0 min,...	1.163	1.301	0 min, 17 sec	0 min, 35 sec	0 min, 35 sec
0.716	**2.628**	**0 min...**	**1.766**	**1.349**	**0 min, 17 sec**	**0 min, 35 sec**	**0 min, 35 sec**

Hình 63:Bảng tổng hợp các thông số mạng (2)

Đánh giá:

- Từ quan sát mô phỏng thấy được năng lượng sử dụng bởi các node có sự phân bố tương ứng với vị trí node trong mạng như phỏng đoán, phần mềm Cooja mô phỏng tốt. Năng lượng ở các node xa có khoảng cách xa với các parent hơn sẽ tiêu tốn nhiều năng lượng hơn và có xảy ra mất mát bản tin. Các node được phân bố đồng đều sẽ có mức tiêu thụ năng lượng tương đương nhau
- Các node chọn parent theo chất lượng đường truyền, node ở quá xa mạng sẽ không tham gia vào mạng được
- Chất lượng đường truyền được thể hiện qua thông số ETX, trong mô phỏng, với phân bố mạng đều thì ETX ổn định
- Một số kịch bản mô phỏng thử nghiệm đều có kết quả hình thành mạng trong thời gian ngắn từ 2-3 phút qua đó thấy được giao thức RPL có hiệu năng tốt trong việc hình thành topo mạng
- Các kịch bản đưa ra đều thỏa mãn yêu cầu của mục tiêu mô phỏng: tối thiểu hóa số chặng truyền gói tin từ các node tời DODAG root
- Trên contiki-3.0 hỗ trợ công cụ hiển thị các thông số mạng Collect view, đây là công cụ mạnh mẽ cho phép thu thập các thông số mạng
- Trên contiki-ng không còn hỗ trợ công cụ Collect view này nữa, việc thu thập hiển thị các thông số mạng mong muốn phải thông qua sử dụng API, người dùng tự triển khai để đạt được các thông số định tuyến mong muốn

- Do giới hạn về thời gian, chúng em chưa học được cách sử dụng các API này, do đó còn nhiều hạn chế trong việc trực quan hóa và so sánh các quá trình mô phỏng với lý thuyết.

Hạn chế:

Chưa thể hiện trực quan hóa quá trình khởi tạo mạng, định tuyến downward và upward và các gói bản tin trên cooja thông qua việc sử dụng API

Chưa phân tích hết được hiệu năng của mạng từ các dữ liệu đã thu thập

Nhật ký

(ghi ngắn gọn, mỗi tuần ghi không quá 1 trang A4 đánh máy)

Tuần 3:

Tổ chức

Địa điểm: C1-108

Thời gian: T3

Tham dự: Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung 1: Tìm hiểu tổng quan về đề tài
Kết luận của buổi họp: Tìm tài liệu về hệ điều hành Contiki
Tuần 4:
Tổ chức
Địa điểm: C1-108
Thời gian: T3
Tham dự: Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung 1: Tìm hiểu về công cụ mô phỏng hệ điều hành Contiki
Kết luận của buổi họp: Xác định các tính năng hỗ trợ của Cooja cho mô phỏng Contiki
Tuần 5:
Tổ chức
Địa điểm: C1-108
Thời gian: T3
Tham dự: Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung 1: Tìm hiểu bài báo
Kết luận của buổi họp: Xác định mục tiêu, kết quả thí nghiệm bài báo
Tuần 6:
Tổ chức
Địa điểm: Thư viện - 713
Thời gian: T4
Tham dự: Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung 1:Tìm hiểu bài báo
Kết luận của buổi họp: Thống nhất các kiến thức trong bài báo, tìm hiểu triển khai dần thí nghiệm
Tuần 7:
Tổ chức
Địa điểm: Thư viện - 713
Thời gian: T4
Tham dự: Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung 1:Tìm hiểu bài báo
Kết luận của buổi họp: Thống nhất các kiến thức trong bài báo, tìm hiểu triển khai dần thí nghiệm

Tuần 8:
Tổ chức
Địa điểm: Thư viện - 713

Thời gian: T4
Tham dự: Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung 1: Triển khai mô phỏng thí nghiệm trong bài báo
Kết luận của buổi họp: Sử dụng các công cụ hỗ trợ Cooja để thực hiện lại kết quả thí nghiệm 1 của bài báo

Tuần 10:
Tổ chức
Địa điểm: Thư viện - 713
Thời gian: T4
Tham dự: Vũ Việt Hoàng, Nguyễn Tiến Thành, Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung : Tìm hiểu các vấn đề trong quá trình khởi tạo mạng
Kết luận của buổi họp: Đưa ra được một số kết luận về cách khởi tạo mạng thông qua bản tin DIO và DIS
Tuần 11:
Tổ chức
Địa điểm: Thư viện - 713
Thời gian: T4
Tham dự: Vũ Việt Hoàng, Nguyễn Tiến Thành, Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung : Tìm hiểu lại các vấn đề trong quá trình khởi tạo mạng, đưa ra phương án triển để trực quan quá trình khởi tạo mạng qua mô phỏng cooja
Kết luận của buổi họp: Sử dụng một số tool có sẵn của cooja để quan sát, đồng thời phân tích thử bản tin được truyền giữa các node bằng phần mềm wireshark
Tuần 12:
Tổ chức
Địa điểm: Thư viện - 713
Thời gian: T4
Tham dự: Vũ Việt Hoàng, Nguyễn Tiến Thành, Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung : Định tuyến upward và các vấn đề liên quan
Kết luận của buổi họp: Đưa ra một số kết luận về quá trình upward routing sử dụng bản tin DAO, DAO-ACK, cách chọn prefer parent của một node trong quá trình định tuyến lên…
Tuần 13:
Tổ chức
Địa điểm: Thư viện - 713

Thời gian: T6
Tham dự: Vũ Việt Hoàng, Nguyễn Tiến Thành, Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung : Thảo luận lại các vấn đề về cơ chế các bản tin DIO, DAO, DIS trong các quá trình mạng: khởi tạo, định tuyến upward, downward, khi mạng ổn định hoặc bất ổn định…. cơ chế Trigger Timer
Kết luận của buổi họp: Đưa ra một số kết luận về quá trình upward routing sử dụng bản tin DAO, DAO-ACK, cách chọn prefer parent của một node trong quá trình định tuyến lên…
Tuần 14:
Tổ chức
Địa điểm: Thư viện - 713
Thời gian: T4
Tham dự: Vũ Việt Hoàng, Nguyễn Tiến Thành, Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung : Định tuyến downward, các loại giao tiếp P2P, MP2P, P2MP
Kết luận của buổi họp: Các chế mode trong downward rooting: storing mode, non-storing mode, cơ chế chung của các mode trong downward routing
Tuần 15:
Tổ chức
Địa điểm: Thư viện - 713
Thời gian: T6
Tham dự: Vũ Việt Hoàng, Nguyễn Tiến Thành, Nguyễn Đình Khánh, Vũ Hoàng Mạnh
Nội dung cuộc họp:
Nội dung : Báo cáo đồ án
Kết luận của buổi họp: phân chia công việc để hoàn thành báo cáo

Thử nghiệm và Đánh giá

Hoàn thiện sản phẩm

Kết luận
(Đối chiếu lại kế hoạch dự kiến ở mục Tổ chức thực hiện, Nếu có thay đổi thì cần ghi rõ những phần thay đổi so với Kế hoạch, nêu lý do)

Nguyễn Đình Khánh + Vũ Hoàng Mạnh

Nội dung	Kết quả cần đạt	Thời gian thực hiện thực tế (theo tuần)	Trạng thái (hoàn thành, chư hoàn thành, không thực hiện) và kết quả đã đạt được so với dự kiến
Tìm hiểu nội dung khái quát về dự án	Báo cáo tổng quan, đánh giá	T1-T3	ĐẠT
Lên phương án sơ bộ	Cài đặt các phần mềm cần thiết: Contiki, Cooja	T3-T4	ĐẠT
Làm quen với hệ điều hành Contiki, công cụ mô phỏng cooja Đọc hiểu khái quát	Phân tích khả năng mô phỏng của cooja, lợi ích sử dụng cooja cùng với	T4-T6	ĐẠT

bài báo cho dự án	contiki Đọc hiểu sơ bộ nội dng bài báo		
Phân tích yêu cầu và mục đích của bài báo khoa học Mô phỏng bài toán trong bài báo	Mô hình hóa thành công bài báo trong bài toán và mô phỏng mô hình đó	T6-T8	Để mô hình hóa được bài toán trong bài báo, cần đọc hiểu kiến thức về mạng RPL, các hàm mục tiêu thực hiện trong mạng RPL. Để mô phỏng được năng lượng tiêu thụ như trong bài báo, cần tìm hiểu module Energest, module tính toán năng lượng được contiki hỗ trợ.
Viết tài liệu hướng dẫn sử dụng Cooja và Contiki	Đưa ra bản word hướng dẫn cài đặt Cooja, Contiki và mô phỏng mô hình bài báo	T8-T9	ĐẠT
Quay video hướng dẫn dựa trên bản	Cho ra video được chia làm các phần	T9-T10	Video: 1. https://www.youtube.com/watch?v=Pb2P6y2UsT 2. https://www.youtube.com/watch?v=2xKL-rR7EgA 3. https://www.youtube.com/watch?v=JnTErHXQSa4&t=819s

word đã báo cáo	hướng dẫn nhỏ		

Nguyễn Đình Khánh

Nội dung	Mô tả (tính năng, thông số,…)	Kết quả cần đạt	Thời gian thực hiện thực tế (theo tuần)	Trạng thái (hoàn thành, chư hoàn thành, không thực hiện) và kết quả đã đạt được so với dự kiến
Tìm hiểu thêm về công cụ Cooja	Tìm hiểu các công cụ khác trong cooja simulation	Các công cụ trong mô phòng như Radio message, Breack point, Buffer View, RSSI base	Tuần 10 - 11	ĐẠT
Mô phỏng thêm sô lượng node trong mạng	Nâng số node lên 20-100 node như trong bài báo, đánh giá về năng lượng, CPU. Radio như trong bài báo. Viết lại file python	Nâng số node lên 20-100 node như trong bài báo, đánh giá về năng lượng, CPU. Radio như trong bài báo	Tuần 11-12	ĐẠT
Cách hàm mục tiêu MRHOF được thực thi trong Contiki	Mạng RPL dùng hàm mục tiêu để thiết lập DADOG, bảng định tuyến. Cần hiểu được cách triển khai code hàm mục tiêu thế nào	Hiểu được code thực thi hàm MRHOF	Tuần 12-13	Cần thêm thời gian để hiểu về thiết lập mạng RPL
Thực thi Contiki	Port contiki os vào vi điều khiển msp430	Thực hiện các chương	Tuần 13-14	ĐẠT

trên phần cứng	và thực hiện một số chương trình cơ bản	trình như lấy năng lượng từ CPU, Udp-server		
Thêm các node phần cứng	Thêm các node MSP430 để hiểu được giao thức RPL thực thi trên phần cứng như thế nào	Dùng chương trình rpl-udp và module energest trong contiki	Tuần 14-15	Chưa đi sâu vào sử dụng phần cứng, mà thay vào đó hiểu thêm về cách thiết lập mạng RPL và ContikiRPL
Áp dụng thuật toán học máy	Thuật toán RL được áp dụng trong contiki os ở đâu và thực hiện thế nào	Code lại chương trình cho foldẻ rpl-lite để áp dụng thuật toán học máy nếu được	Tuần 15-16	Chưa đi sâu vào thuật toán, tìm hiểu ContikiRPL
Làm báo cáo và trình bày kết thúc dự án	Tổng hợp kiên thức thông qua dự án	Làm Slide, báo cáo		

Vũ Hoàng Mạnh

Nội dung	Mô tả (tính năng, thông số,…)	Kết quả cần đạt	Thời gian thực hiện (theo tuần)	Ghi chú
Tìm hiểu thêm về công cụ Cooja	Tìm hiểu các công cụ khác trong cooja simulation	Các công cụ trong mô phòng như Radio message, Breack point, Buffer View, RSSI base	Tuần 10 - 11	ĐẠT
Mô phỏng thêm	Nâng số node lên 20-100 node như trong bài báo, đánh giá về	Nâng số node lên 20-100 node như	Tuần 11-12	ĐẠT

sô lượng node trong mạng	năng lượng, CPU. Radio như trong bài báo. Viết lại file python	trong bài báo, đánh giá về năng lượng, CPU. Radio như trong bài báo		
Cách hàm mục tiêu OF0 được thực thi trong Contiki	Mạng RPL dùng hàm mục tiêu để thiết lập DADOG, bảng định tuyến. Cần hiểu được cách triển khai code hàm mục tiêu thế nào	Hiểu được code thực thi hàm OF0, so sánh với MRHOF	Tuần 12-13	ĐẠT
Thực thi Contiki trên phần cứng	Port contiki os vào vi điều khiển msp430 và thực hiện một số chương trình cơ bản	Thực hiện các chương trình như lấy năng lượng từ CPU, Udp-server	Tuần 13-14	ĐẠT
Thêm các node phần cứng	Thêm các node MSP430 để hiểu được giao thức RPL thực thi trên phần cứng như thế nào	Dùng chương trình rpl-udp và module energest trong contiki	Tuần 14-15	Chưa đi sâu vào sử dụng phần cứng, mà thay vào đó hiểu thêm về cách thiết lập mạng RPL và ContikiRPL
Áp dụng thuật toán học máy	Thuật toán RL được áp dụng trong contiki os ở đâu và thực hiện thế nào	Code lại chương trình cho foldẻ rpl-lite để áp dụng thuật toán học máy nếu được	Tuần 15-16	Chưa đi sâu vào thuật toán, tìm hiểu ContikiRPL
Làm báo cáo và	Tổng hợp kiên thức thông qua dự án	Làm Slide, báo cáo		

trình bày kết thúc dự án				

TÀI LIỆU THAM KHẢO

1. https://www.contiki-ng.org/
2. https://ieeexplore.ieee.org/document/9717281
3. https://www.rfc-editor.org/rfc/rfc6552
4. https://www.rfc-editor.org/rfc/rfc6719
5. https://www.ietf.org/slides/slides-smartobjectsws-tutorial-understanding-routing-in-low-power-and-lossy-networks-00.pdf
6. Luận văn thạc sĩ: Định tuyến cho mạng internet of things - Nguyễn Hải Nam - Thư viện số Đại học Bách Khoa Hà Nội
7. https://datatracker.ietf.org/doc/id/draft-ietf-roll-rpl-13.html#:~:text=Routing%20Protocol%20for%20LLNs%20%28RPL%29%2C%20which%20provides%20a,the%20Support%20for%20point-to-point%20traffic%20is%20also%20available.