**REMI ENGLISH**

**MỤC LỤC**

[PART 1: LOGIN/REGISTER VIEW 1](#_Toc438892663)

[1.1 Overview 1](#_Toc438892664)

[1.2 Login 1](#_Toc438892665)

[1.3 Register 1](#_Toc438892665)

[PART 2: MASTER TAB 3](#_Toc438892666)

[2.1 Overview 3](#_Toc438892667)

[2.2 Level (CRUD) 4](#_Toc438892668)

[2.3 Lesson (CRUD) 6](#_Toc438892669)

[2.4 Register (CRUD) 1](#_Toc438892665)

[PART 3: USER TAB 12](#_Toc438892672)

[3.1 Overview 12](#_Toc438892673)

[3.2 Role/Course 12](#_Toc438892674)

[3.3 Create/Update/Delete 13](#_Toc438892675)

[PART 4: COURSE TAB 17](#_Toc438892679)

[4.1 Overview 17](#_Toc438892680)

[4.2 Create/Update/Delete 17](#_Toc438892680)

[PART 5: STATISTICAL/ STATISTICAL MANAGER TAB 33](#_Toc438892699)

[5.1 Overview 33](#_Toc438892700)

[PART 6: TEST COURSE TAB 49](#_Toc438892710)

[6.1 Overview 49](#_Toc438892711)

[6.2 Course 56](#_Toc438892714)

# PART 1: LOGIN/RERISTER VIEW

## 1.1 Overview

## Phác thảo sơ lược luận văn

**Chương 2:** Tổng quan về mạng cảm biến không dây. Cung cấp một số kiến thức tổng quát về cấu trúc, đặc điểm cũng như những ứng dụng của mạng cảm biến không dây trong đời sống hiện nay.

**Chương 3:** Giới thiệu về mô hình cơ sở dữ liệu cảm biến. Phân tích đặc điểm của các loại kiến trúc mạng dữ liệu cảm biến hiện nay. So sánh chi phí năng lượng của quá trình xử lí dữ liệu trong mỗi thiết bị cảm biến với các quá trình giao tiếp trong mạng. Đưa ra cái nhìn tổng quá về hướng ứng dụng, công nghệ, cũng như những thách thức của mô hình cơ sở dữ liệu cảm biến.

**Chương 4:** Phân tích chi tiết về ANTELOPE, một hệ thống quản lí cơ sở dữ liệu cho các thiết bị cảm biến có nguồn lực hạn chế. Đi sâu về cấu trúc, các thành phần chính của ANTELOPE cũng như nhiệm vụ của chúng. Hiểu rõ quá trình thực hiện, đặc điểm về hiệu suất năng lượng và thời của việc đánh chỉ số cho các phần tử trong cơ sở dữ liệu.

**Chương 5:** Miêu tả về đặc điểm cấu trúc Hệ điều hành mã nguồn mở Contiki, được thiết kế cho các vi điều khiển có bộ nhớ nhỏ, ứng dụng trong rất nhiều lĩnh vực thực tế. Giới thiệu về phần mềm mô phỏng Cooja, một công cụ rất hữu ích, được tích hợp trong hệ điều hành Contiki dùng để mô phỏng cá hệ thống mạng, đặc biệt là các hệ thống mạng cảm biến. Cho phép người sử dụng thay đổi các thông số như vị trí, phạm vi kết nối tỉ lệ truyền gói thành công.

**Chương 6:** Đưa ra các mô hình khảo sát, đánh giá công suất tiêu tốn, thời gian xử lí của các nút cảm biến trong những thuật toán đánh chỉ số khác nhau bằng công cụ mô phỏng Cooja. Nhằm so sánh hiệu suất về năng lượng cũng như thời gian xử lí của các thuật toán chỉ số được dùng trong ANTELOPE

**Chương 7:** Tổng kết các công việc đã thực hiện trong luận văn, kết luận cũng như đưa ra những hướng phát triển và ứng dụng của đề tài.

# CHƯƠNG 2: MẠNG CẢM BIẾN KHÔNG DÂY

## 2.1 Giới thiệu

Mạng cảm biến không dây (Wireless Sensor Network) là một mạng bao gồm nhiều nút cảm biến nhỏ có giá thành thấp, và tiêu thụ năng lượng ít, giao tiếp thông qua các kết nối không dây có nhiệm vụ cảm nhận, đo đạc, tính toán nhằm mục đích thu thập, tập trung dữ liệu phân tán với quy mô lớn trong bất kỳ điều kiện và ở bất kỳ vùng địa lý nào để đưa ra các quyết định toàn cục về môi trường tự nhiên. Những nút cảm biến không dây có thể được triển khai cho các mục đích chuyên dụng như điều khiển giám sát và an ninh; kiểm tra môi trường; tạo ra không gian sống thông minh; khảo sát đánh giá chính xác trong nông nghiệp; trong lĩnh vực y tế;… Lợi thế chủ yếu của chúng là khả năng triển khai hầu như trong bất kì loại hình địa lý nào kể cả các môi trường nguy hiểm không thể sử dụng mạng cảm biến có dây truyền thống. Các thiết bị cảm biến không dây liên kết thành một mạng đã tạo ra nhiều khả năng mới cho con người. Các đầu đo với bộ vi xử lý và các thiết bị vô tuyến rất nhỏ gọn tạo nên một thiết bị cảm biến không dây có kích thước rất nhỏ, tiết kiệm về không gian. Chúng có thể hoạt động trong môi trường dày đặc với khả năng xử lý tốc độ cao.

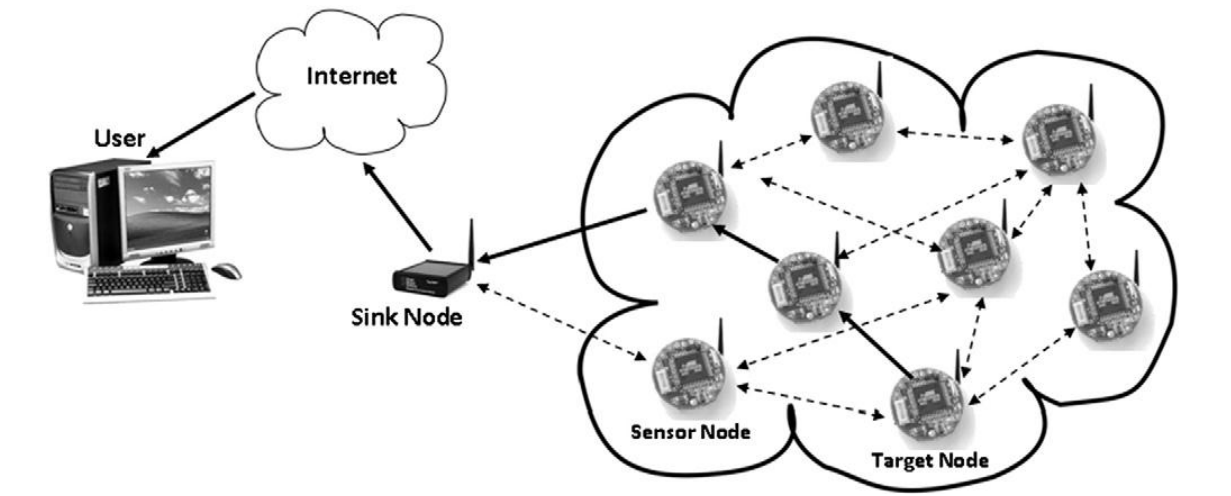
1 Hình 2.1: Ứng dụng của mạng cảm biến không dây



Ngày nay, các mạng cảm biến không dây được ứng dụng trong nhiều lĩnh vực như nghiên cứu vi sinh vật biển, giám sát việc chuyên chở các chất gây ô nhiễm, kiểm tra giám sát hệ sinh thái và môi trường sinh vật phức tạp, điều khiển giám sát trong công nghiệp và trong lĩnh vực quân sự, an ninh quốc phòng hay các ứng dụng trong đời sống hàng ngày.

## 2.2 Cấu trúc của mạng cảm biến không dây

Một mạng cảm biến không dây bao gồm số lượng lớn các nút được triển khai dầy đặc bên trong hoặc ở rất gần đối tượng cần thăm dò, thu thập thông tin dữ liệu. Vị trí các cảm biến không cần định trước vì vậy nó cho phép triển khai ngẫu nhiên trong các vùng không thể tiếp cận hoặc các khu vực nguy hiểm. Khả năng tự tổ chức mạng và cộng tác làm việc của các cảm biến không dây là những đặc trưng rất cơ bản của mạng này. Với số lượng lớn các cảm biến không dây được triển khai gần nhau thì truyền thông đa liên kết được lựa chọn để công suất tiêu thụ là nhỏ nhất (so với truyền thông đơn liên kết) và mang lại hiệu quả truyền tín hiệu tốt hơn so với truyền khoảng cách xa.



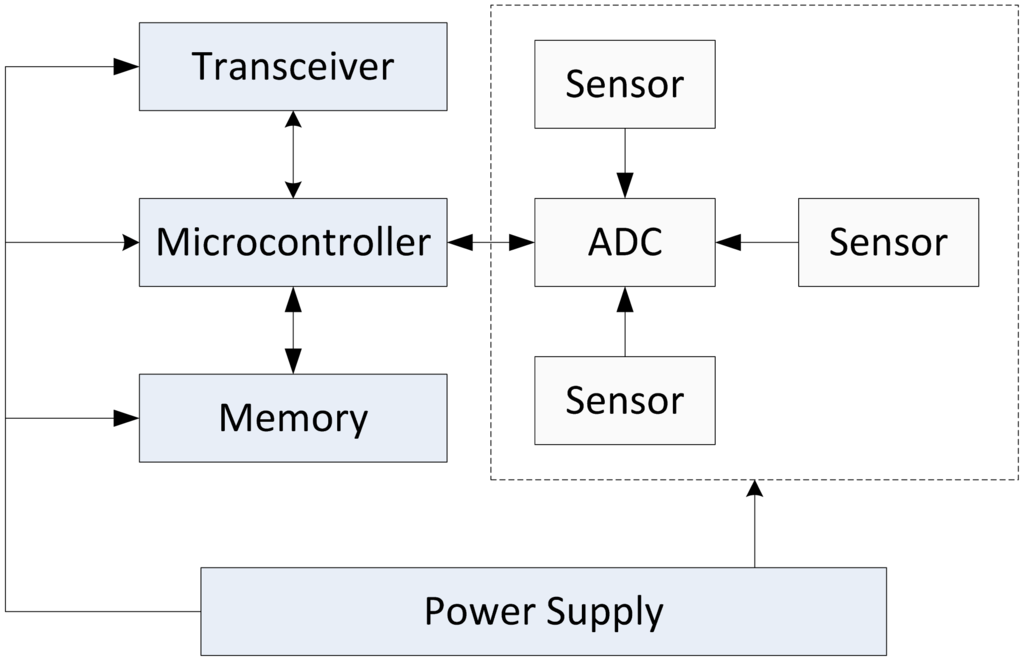
2 Hình 2.2: Mô hình mạng cảm biến không dây

Cấu trúc cơ bản của mạng cảm biến không dây được thể hiện trên hình 2.2. Các nút cảm biến được triển khai trong một trường cảm biến (sensor field). Mỗi nút cảm biến được phát tán trong mạng có khả năng thu thập thông số dữ liệu, định tuyến số liệu về bộ thu nhận (sink), nơi cho phép kết nối trường cảm biến trực tiếp hay gián tiếp với trạm điều khiển (task manager) thông qua Internet hay vệ tinh.

Cấu trúc mang cảm biến được chia thành ba phần chính:

* *Trường cảm biến (sensor field)*: được tạo thành từ một tập hợp các nút cảm biến liên kết với nhau. Một số đặc tính của trường cảm biến như: loại nút, phân bố địa lí của nút, mật độ của trường cảm biến… Và nó dựa vào sự cần thiết trên mỗi ứng dụng của mạng.
* *Bộ phận thu nhận (sink)*: là một nút đặc biệt của mạng, vai trò của nó chính là thu thập dữ liệu từ trường cảm biến và nó được thiết kế với nguồn năng lượng không hạn chế để nhận thông tin từ các cảm biến ở bất kì thời gian nào.
* *Trạm điều khiển (task manager)*: nhận dữ liệu được thu thập từ nút sink. Nó chịu trách nhiệm quản lí các dữ liệu đã được tập hợp bằng cách phân tích và thực thi để trích ra những dữ liệu có ích. Có thể thông qua một mô hình mạng khác để kết nối giữa trạm điều khiển và thành phần khác của mạng cảm biến, thông thường là mạng Internet.

Mỗi nút cảm biến bao gồm năm thành phần cơ bản là: *bộ cảm biến (sensor)*, *bộ xử lý (microcontroller)*, *bộ nhớ (memory)*, *bộ thu phát không dây (transceiver)* và *nguồn điện (power supply)*. Tuỳ theo ứng dụng cụ thể, nút cảm biến còn có thể có các thành phần bổ sung như hệ thống tìm vị trí, bộ sinh năng lượng và thiết bị di động. Các thành phần chính trong một nút cảm biến được thể hiện trên hình 2.3.



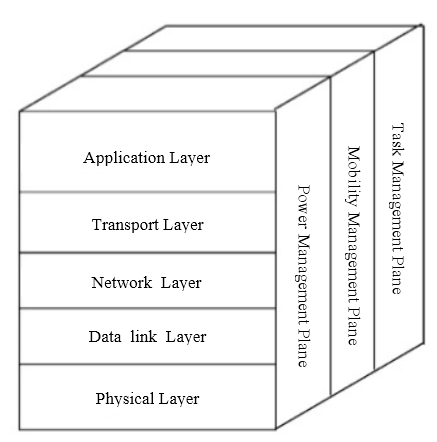
3Hình 2.3: Thành phần chính của nút cảm biến

* *Bộ cảm biến (sensor)*: thường gồm hai đơn vị thành phần là đầu đo cảm biến và bộ chuyển đổi tương tự/số (ADC). Các tín hiệu tương tự được thu nhận từ đầu đo, sau đó được chuyển sang tín hiệu số bằng bộ chuyển đổi ADC, rồi mới được đưa tới bộ xử lý.
* *Bộ xử lý (microcontroller)*: thường kết hợp với một bộ nhớ nhỏ, dung để phân tích thông tin cảm biến và quản lý các thủ tục cộng tác với các nút khác để phối hợp thực hiện nhiệm vụ.
* *Bộ thu phát (transceiver)*: đảm bảo thông tin giữa nút cảm biến và mạng bằng kết nối không dây, có thể là vô tuyến, hồng ngoại hoặc bằng tín hiệu quang. Một thành phần quan trọng của nút cảm biến là bộ nguồn.
* *Bộ nguồn (power supply)*: có thể là pin hoặc ắcquy, cung cấp năng lượng cho nút cảm biến và không thay thế được nên nguồn năng lượng của nút thường là giới hạn. Bộ nguồn có thể được hỗ trợ bởi các thiết bị sinh điện, ví dụ như các tấm pin mặt trời nhỏ.
* *Bộ nhớ (memory)*: Lưu cấu hình của nút cảm biến, và cơ sở dữ liệu cảm biến.

Hầu hết các công nghệ định tuyến trong mạng cảm biến và các nhiệm vụ cảm biến yêu cầu phải có sự nhận biết về vị trí với độ chính xác cao. Do đó, các nút cảm biến thường phải có hệ thống tìm vị trí. Các thiết bị di động đôi khi cũng cần thiết để di chuyển các nút cảm biến theo yêu cầu để đảm bảo các nhiệm vụ được phân công.

## 2.3 Kiến trúc giao thức mạng

Kiến trúc giao thức được sử dụng trong bộ thu nhận (sink) và tất cả các nút cảm biến được thể hiện trên hình 2.4. Kiến trúc giao thức này phối hợp các tính toán về định tuyến và năng lượng, kết hợp số liệu với các giao thức mạng, truyền tin với hiệu quả về năng lượng thông qua môi trường không dây và tăng cường sự hợp tác giữa các nút cảm biến. Kiến trúc giao thức bao gồm lớp ứng dụng (Application Layer), lớp giao vận (Transport Layer), lớp mạng (Network Layer), lớp liên kết số liệu (Datalink Layer), lớp vật lý (Physical Layer), mặt phẳng quản lý năng lượng (Power Management Plane), mặt phẳng quản lý di động (Mobility Management Plane) và mặt phẳng quản lý nhiệm vụ (Task Management Plane).



4Hình 2.4. Kiến trúc giao thức mạng cảm biến không dây

* *Lớp ứng dụng (Application Layer)* :Tùy vào từng nhiệm vụ của mạng cảm biến mà các phần mềm ứng dụng khác nhau được xây dựng và sử dụng trong lớp ứng dụng. Trong lớp ứng dụng có một số giao thức quan trọng như giao thức quản lí mạng cảm biến (SMP – Sensor Management Protocol), giao thức quảng bá dữ liệu và chỉ định nhiệm vụ cho từng cảm biến (TADAP – Task Assignment and Data Advertisement), giao thức phân phối dữ liệu và truy vấn cảm biến (SQDDP – Sensor Query and Data Dissemination).
* *Lớp truyền tải (Transport Layer):* giúp duy trì luồng số liệu nếu ứng dụng mạng cảm biến yêu cầu. Lớp truyền tải đặc biệt cần khi mạng cảm biến kết nối với mạng bên ngoài, hay kết nối với người dùng qua internet. Giao thức lớp vận chuyển giữa sink với người dùng (nút quản lý nhiệm vụ) thì có thể là giao thức gói người dùng (UDP – User Datagram Protocol) hay giao thức điều khiển truyền tải (TCP – Transmission Control Protocol) thông qua internet hoặc vệ tinh. Còn giao tiếp giữa sink và các nút cảm biến cần các giao thức kiểu như UDP vì các nút cảm biến bị hạn chế về bộ nhớ. Hơn nữa các giao thức này còn phải tính đến sự tiêu thụ công suất, tính mở rộng và định tuyến tập trung dữ liệu .
* *Lớp mạng (Network Layer):* quan tâm đến việc định tuyến dữ liệu được cung cấp bởi lớp truyền tải. Việc định tuyến trong mạng cảm biến phải đối mặt với rất nhiều thách thức như mật độ các nút dày đặc, hạn chế về năng lượng…Do vậy thiết kế lớp mạng trong mạng cảm biến phải theo các nguyên tắc sau:
  + Hiệu quả về năng lượng luôn được xem là vấn đề quan trọng hàng đầu.
  + Các mạng cảm biến gần như là tập trung dữ liệu
  + Tích hợp dữ liệu và giao thức mạng.
  + Phải có cơ chế địa chỉ theo thuộc tính và biết về vị trí

Có rất nhiều giao thức định tuyến được thiết kế cho mạng cảm biến không dây. Nhìn tổng quan, chúng được chia thành ba loại dựa vào cấu trúc mạng, đó là định tuyến ngang hàng, định tuyến phân cấp, định tuyến dựa theo vị trí. Xét theo hoạt động thì chúng được chia thành định tuyến dựa trên đa đường (multipath-based), định tuyến theo truy vấn (query-based), định tuyến thỏa thuận (negotiation-based), định tuyến theo chất lượng dịch vụ (QoS – Quanlity of Service), định tuyến kết hợp (coherent-based).

* *Lớp kết nối dữ liệu (Data Link Layer):* Lớp kết nối dữ liệu chịu trách nhiệm cho việc ghép các luồng dữ liệu, dò khung dữ liệu, điều khiển lỗi và truy nhập môi trường. Vì môi trường có tạp âm và các nút cảm biến có thể di động, giao thức điều khiển truy nhập môi trường (MAC – Media Access Control) phải xét đến vấn đề công suất và phải có khả năng tối thiểu hoá việc va chạm với thông tin quảng bá của các nút lân cận.
* *Lớp vật lý (Physical Layer)*: chịu trách nhiệm lựa chọn tần số, phát tần số sóng mang, điều chế, lập mã và tách sóng.
* *Mặt phẳng quản lý công suất (Power Management Plane):* điều khiển việc sử dụng công suất của nút cảm biến. Ví dụ, nút cảm biến có thể tắt khối thu của nó sau khi thu được một bản tin từ một nút lân cận. Điều này giúp tránh tạo ra các bản tin giống nhau. Khi mức công suất của nút cảm biến thấp, nút cảm biến phát quảng bá tới các nút lân cận để thông báo nó có mức công suất thấp và không thể tham gia vào các bản tin chọn đường. Công suất còn lại sẽ được dành riêng cho nhiệm vụ cảm biến.
* *Mặt phẳng quản lý di động (Mobility Management Plane):* phát hiện và ghi lại sự di chuyển của các nút cảm biến để duy trì tuyến tới người sử dụng và các nút cảm biến. Nhờ xác định được các nút cảm biến lân cận, các nút cảm biến có thể cân bằng giữa công suất của nó và nhiệm vụ thực hiện.
* *Mặt phẳng quản lý nhiệm vụ (Task Management Plane):* có thể lên kế hoạch các nhiệm vụ cảm biến trong một vùng xác định. Không phải tất cả các nút cảm biến trong vùng đó điều phải thực hiện nhiệm vụ cảm biến tại cùng một thời điểm. Kết quả là một số nút cảm biến thực hiện nhiệm vụ nhiều hơn các nút khác tuỳ theo mức công suất của nó.

Những phần quản lý này là cần thết để các nút cảm biến có thể làm việc cùng nhau theo một cách thức sử dụng hiệu quả công suất, chọn đường số liệu trong mạng cảm biến di động và phân chia tài nguyên giữa các nút cảm biến.

## 2.4 Các yếu tố ảnh hưởng tới mạng cảm biến không dây

***Thời gian sống bên ngoài:***Các nút WSN với nguồn năng lượng pin giới hạn. Ví dụ: một loại pin kiềm cung cấp 50Wh năng lượng, nó có thể truyền cho mỗi nút mạng ở chế độ tích cực gần một tháng hoạt động. Sự tiêu tốn và tính khả thi của giám sát và thay thế pin cho một mạng rộng, thì thời gian sống dài hơn được thiết kế. Trong thực tế, pin rất cần thiết trong rất nhiều ứng dụng để bảo đảm mạng WSN có thể tự động sử dụng không cần thay thế trong vài năm. Sự cải thiện của phần cứng trong thiết kế pin và kĩ thuật thu năng lượng sẽ giúp ta một phần trong việc tiết kiệm pin.

***Sự đáp ứng:*** Giải pháp đơn giản nhất để kéo dài thời gian sống bên ngoài là điều khiển các nút trong một chu kì làm việc với chu kì chuyển mạch giữa hai chế độ: chế độ ngủ (mode sleep) và chế độ hoạt động (mode active). Trong khi quá trình đồng bộ ở chế độ ngủ là một thách thức của WSN, vấn đề lớn liên quan đến nữa là chu trình ngủ một cách tùy ý có thể làm giảm khả năng đáp ứng cũng như hiệu suất của các cảm biến. Trong một số ứng dụng, các sự kiện trong tự nhiên được tìm thấy và thông báo nhanh, thì sự trễ bởi lịch ngủ phải được giữ ở giới hạn chính xác, thậm chí trong sự tồn tại của nghẽn mạng.

***Tính chất mạnh (Robustness):*** Mục tiêu của WSN là cung cấp ở phạm vi rộng lớn, độ bao phủ chính xác. Mục tiêu này phổ biến ở số lượng lớn các thiết bị không đắt tiền. Tuy nhiên các thiết bị rẻ thường kém tin cậy và thường dễ xảy ra lỗi. Tốc độ lỗi cũng sẽ cao khi các thiết bị cảm ứng được triển khai trong các môi trường khắc khe và trong vùng của kẻ địch. Giao thức thiết kế do đó cũng phải xây dựng kỹ sảo để có thể đáp ứng tốt. Rất khó để chắc chắn rằng việc định dạng toàn cầu của hệ thống là không bị hỏng với các thiết bị lỗi.

***Tính mở rộng (Scalability ):*** WSN có khả năng hoạt động ở một vùng cực rộng (lớn hơn 10 ngàn, thậm chí là hàng triệu nút trong một giới hạn về độ dài).Có một vài hạn chế về thông lượng và dung lượng làm ảnh hưởng đến scalability của hoạt động mạng.

***Tính không đồng nhất (Heterogeneity):*** Sẽ tồn tại sự không đồng nhất trong dung năng của thiết bị trong quá trình cài đặt thực tế (cụ thể là máy móc, thông tin dữ liệu và cảm biến). Sự không đồng nhất sẽ có ảnh hưởng quan trọng đến thiết kế.

***Tự cấu hình:*** Do phạm vi và các ứng dụng trong tự nhiên, WSN là các hệ thống phân phốikhông cần chủ. Hoạt động tự động là vấn đề chính được đặt ra trong thiết kế. Ngay từ khi bắt đầu, các nút trong WSN có thể được cấu hình theo topo mạng của chúng; tự đồng bộ, tự kiểm tra, và quyết định các thông số hoạt động khác.

***Tự tối ưu và thích nghi:*** Trong WSN, thường có những tín hiệu không chắc chắn về điều kiện hoạt động trước khi triển khai. Dưới những điều kiện đó, việc xây dựng những máy móc để có thể tự học từ cảm biến và thu thập các phép đo mạng, sử dụng những cái học được đó để tiếp tục hoạt động cải tiến là điều rất quan trọng.

Ngoài ra, một điều trước tiên không biết chắc được là môi trường mà WSN hoạt động có thể thay đổi mạnh mẽ qua thời gian. Các giao thức WSN sẽ làm cho thiết bị có thể thích nghi với môi trường năng động trong khi nó đang sử dụng.

***Thiết kế có hệ thống:*** WSN có thể là một ứng dụng cao cho từng chức năng riêng, nên cần có sự cân bằng giữa hai yếu tố:

* Mỗi ứng dụng cần có những đặc điểm khai thác ứng dụng riêng để đưa ra những hoạt động phát triển cao.
* Các phương pháp thiết kế phải phổ biến cho các hoạt động

***Cách biệt và bảo mật:*** Phạm vi hoạt động lớn, phổ biến rộng, nhạy của thông tin thu được bởi vì WSN làm tăng yêu cầu chính cuối cùng là: bảo đảm sự cách biệt và bảo mật.

## 2.5 Một số ứng dụng của mạng cảm biến không dây

**Mạng cảm biến không dây trong y tế và giám sát sức khoẻ** :Một số ứng dụng trong y tế của mạng cảm biến không dây là cung cấp khả năng giao tiếp cho người khuyết tật; kiểm tra tình trạng của bệnh nhân; chẩn đoán; quản lý dược phẩm trong bệnh viện; kiểm tra sự di chuyển và các cơ chế sinh học bên trong của côn trùng và các loài sinh vật nhỏ khác; kiểm tra từ xa các số liệu về sinh lý con người; giám sát, kiểm tra các bác sĩ và bệnh nhân bên trong bệnh viện.

**Giám sát và điều khiển công nghiệp**: Đặc thù của giám sát và điều khiển công nghiệp là môi trường nhiễu lớn, không đòi hỏi lượng lớn dữ liệu thông tin được truyền tải nhưng yêu cầu rất cao về độ tin cậy và đáp ứng thời gian thực. Mạng cảm biến không dây được ứng dụng trong linh vực này chủ yêu phục vụ việc thu thập thông tin, giám sát trạng thái hoạt động của hệ thống, như trạng thái các van, trạng thái thiết bị, nhiệt độ và áp suất của nguyên liệu được lưu trữ, … Ngoài ra, trong một số ứng dụng điều khiển trên diện rộng thì mạng cảm biến không dây cũng thể hiện nhiều tính năng vượt trội. Đó là hệ thống điều khiển không dây ánh sáng quảng cáo. Rất nhiều chi phí trong quá trình cài đặt các bóng đèn trong một toà nhà lớn (các chuyển mạch có dây, các bóng đèn được bật/tắt cùng nhau, điều khiển bóng đèn, …). Một hệ thống không dây có tính mềm dẻo có thể tận dụng một bộ điều khiển từ xa có thể được lập trình để điều khiển một số lượng các bóng đèn trong một theo nhiều cách khác nhau gần như vô hạn, trong khi vẫn cung cấp mức độ an ninh được yêu cầu bởi một bộ phận lắp đặt quảng cáo. Hay việc sử dụng các mạng cảm biến không dây trong các ứng dụng an toàn công nghiệp.

**Tự động hoá gia đình và điện dân dụng**: Gia đình là không gian ứng dụng rất lớn cho các mạng cảm biến không dây. SmartHome là thuật ngữ để chỉ một ngôi nhà thông minh với sự ứng dụng toàn diện của các thiết bị cảm biến không dây. Mục đích lớn của các mạng cảm biến không dây trong gia đình được mong chờ là mức tiêu thụ điện thấp là điều kiện thiết yếu của các mạng cảm biến không dây. Ứng dụng khác trong gia đình là việc hỗ trợ các dịch vụ gia đình trên ôtô. Với các mạng cảm biến không dây, ổ khoá không dây, các cảm biến cửa ra vào và cửa sổ, và các bộ điều khiển bóng đèn không dây, chủ nhà có một thiết bị với một nút bấm. Khi bấm nút, thiết bị khoá tất cả các cửa ra vào và cửa sổ trong nhà, tắt hầu hết các bóng đèn trong nhà (trừ một vài bóng đèn ngủ), bật các bóng đèn an toàn ngoài nhà.

**Mạng cảm biến không dây với môi trường và ngành nông nghiệp:** Một số các ứng dụng về môi trường của mạng cảm biến không dây bao gồm theo dõi sự di chuyển của các loài chim, loài thú nhỏ, côn trùng; kiểm tra các điều kiện môi trường ảnh hưởng tới mùa màng và vật nuôi; tình trạng nước tưới; các công cụ vĩ mô cho việc giám sát mặt đất ở phạm vi rộng và thám hiểm các hành tinh; phát hiện hóa học, sinh học; tính toán trong nông nghiệp; kiểm tra môi trường không khí, đất trồng, biển; phát hiện cháy rừng; nghiên cứu khí tượng và địa lý; phát hiện lũ lụt; vẽ bản đồ sinh học phức tạp của môi trường và nghiên cứu ô nhiễm môi trường. Các ứng dụng của các mạng cảm biến không dây cũng được sử dụng trên các trang trại chăn nuôi. Người chăn nuôi có thể sử dụng các mạng cảm biến trong quá trình quyết định vị trí của động vật trong trang trại và với các cảm biến được gắn theo mỗi động vật, xác định yêu cầu cho các phương pháp điều trị để phòng chống các động vật ký sinh. Người chăn nuôi lợn hoặc gà có các đàn trong các chuồng nuôi mát, thoáng khí. Mạng  cảm biến không dây có thể được sử dụng cho việc giám sát nhiệt độ khắp chuồng nuôi, đảm bảo an toàn cho đàn.

# CHƯƠNG 3: MÔ HÌNH CƠ SỞ DỮ LIỆU CẢM BIẾN

## 3.1 Giới thiệu

Trong những năm gần đây, giá thành của các bộ nhớ flash công suất thấp đã giảm rất nhiều trong khi dung lượng lưu trữ vẫn tăng một cách nhanh chóng. Đối với những mục đích thực tế, bộ nhớ flash các nút cảm biến hiện nay có thể được coi là vô hạn vì các nút sẽ hết pin trước khi sử dụng hết không gian lưu trữ của chúng. Với lý do trên, nó sẽ giúp chúng ta mở ra một mô hình mạng cảm biến mới, nơi các nút lưu trữ dữ liệu của chúng trong một bộ nhớ flash và cung cấp phương tiện tiết kiệm năng lượng để truy vấn dữ liệu. Đối với nhiều ứng dụng mới nổi, như thu thập ánh sáng ban ngày và tính toán năng lượng, thì việc có thể truy vấn dữ liệu trực tiếp từ các cơ sở dữ liệu được lưu trữ trong mỗi nút sẽ giúp đạt được hiệu quả tốt hơn.

Với một mô hình mạng cảm biến, trong đó mỗi nút sẽ chạy một hệ thống quản lý cơ sở dữ liệu (DBMS), cung cấp một giao diện truy vấn hiệu quả tới bộ nhớ lưu trữ flash của các nút, mỗi nút cảm biến nắm giữ một cơ sở dữ liệu có thể được tự động truy vấn và dữ liệu sẽ được đưa vào tại thời gian chạy. Các cơ sở dữ liệu có thể được sử dụng để lưu trữ dữ liệu cảm biến, mỗi cảm biến có thể lưu lại toàn bộ lịch sử của việc đọc cảm biến, để lưu trữ thông tin thời gian chạy, chẳng hạn như các bảng định tuyến cho các mạng lớn có thể được tự động truy vấn bởi hệ thống chính nó; hoặc để duy trì một lịch sử của thống kê hiệu suất, mà sau này có thể được truy vấn cho mục đích gỡ lỗi mạng hoặc điều chỉnh hiệu suất. Với tiềm năng cơ sở dữ liệu rất lớn trên mỗi nút, cần thiết phải có các cơ chế truy vấn hiệu quả có thể hoạt động trên các tập dữ liệu lớn. Các dữ liệu cảm biến phải được lưu trữ một cách tiết kiệm năng lượng trên bộ nhớ flash. Ngoài ra, sự hạn chế khắt khe của nguồn tài nguyên trên nền tảng mỗi nút đòi hỏi phải có các cơ chế đơn giản có chất lượng và hiệu quả năng lượng cao.

## 3.2 Kiến trúc mạng dữ liệu cảm biến

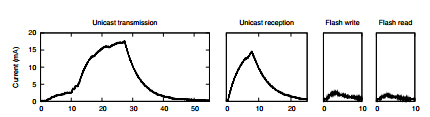
Các mạng cảm biến được triển khai cho đến nay có thể được tạm chia thành hai loại chính: mạng thu thập dữ liệu (data collection networks), mạng ghi chép dữ liệu (data logging networks).

*Mạng thu thập dữ liệu*: là kiến trúc mạng cảm biến nguyên mẫu. Trong một mạng thu thập dữ liệu, tất cả các nút truyền các giá trị cảm biến của chúng cho một hoặc nhiều ổ dữ liệu, sử dụng giao thức thu thập dữ liệu best-effort như giao thức Contiki Collection. Mạng lưới thu thập dữ liệu có thể hỗ trợ tập hợp dữ liệu, tuy nhiên điều này rất khó để sử dụng trong thực tế. TinyDB là một trường hợp đặc biệt của kiến trúc thu thập dữ liệu. Trong TinyDB, các dòng dữ liệu từ các mạng cảm biến được thu thập thông qua một giao diện truy xuất, có chức năng kiểm soát cách dữ liệu được thu thập từ mạng. Các truy xuất được sử dụng dựa vào SQL, như là một ngôn ngữ dùng để thông qua cổng vào của các nút, gửi chỉ thị tới các nút trong mạng về cách gửi các giá trị cảm biến mà nó đọc được.

*Mạng ghi chép dữ liệu*: các cảm biến ghi lại tất cả các dữ liệu đã được cảm nhận vào trong một bộ lưu trữ thứ cấp, nơi mà nó sẽ được lấy ra. Mạng ghi chép dữ liệu được sử dụng khi cần thiết phải lấy các bộ dữ liệu hoàn chỉnh.

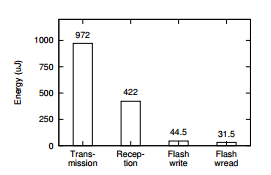
## 3.3 Chi phí năng lượng

Việc giao tiếp trong mạng cảm biến tiêu tốn rất nhiều năng lượng. Việc thu phát vô tuyến có mức tiêu thụ năng lượng tương đối cao. Để giảm năng lượng tiêu thụ của sóng vô tuyến, một cơ chế về chu trình hoạt động của sóng vô tuyến cần được áp dụng. Cùng với chu trình hoạt động, các bộ truyền nhận được ngắt càng nhiều sẽ càng có lợi, trong khi vẫn đủ khả năng cho các nút tham gia giao tiếp trong mạng đa hop. Ngược lại, chip nhớ flash có thể ở trong chế độ ngủ ở mức năng lượng thấp rất lâu miễn là không có quá trình xuất nhập diễn ra, và do đó nó không cần có một chu trình hoạt động.



5Hình 3.1. Biểu đồ dòng điện và thời gian đo được khi truyền, nhận 100 byte trong điều kiện sóng vô tuyến lí tưởng và quá trình ghi và đọc 100 byte từ bộ nhớ flash.

Việc tiêu thụ năng lượng của các phép đo trong hình 3.1 được thể hiện trong hình 3.2. Chúng ta thấy rằng chi phí năng lượng của các hoạt động giao tiếp là lớn hơn rất nhiều so với các hoạt động lưu trữ : mức tiêu thụ năng lượng của quá trình truyền 100 byte cao gấp 20 lần so với việt viết 100 byte vào bộ nhớ flash. Bởi vì truyền tin qua mạng đa hop đòi hỏi sự đa truyền và đa nhận, và các gói tin bị mất phải được truyền lại, do đó tổng chi phí của một truyền trên mạng có thể được tăng thêm rất nhiều.



6Hình 3.2 Quá trình tiêu thụ năng lượng.

## 3.4 Hướng ứng dụng

Đối với những ứng dụng đầu tiên trên mạng cảm biến, mạng cảm biến được xem như là mạng đồng nhất, do đó dữ liệu cảm biến nên được tổng hợp từ nhiều cảm biến. Ví dụ một mạng cảm biến được triển khai trong một trường nhất định nhằm mục đích truy xuất giá trị nhiệt độ trung bình của trường, và để đạt được độ chính xác thì nó nên tập hợp các giá trị nhiệt độ đo được từ tất cả các nút cảm biến trong trường đó. Tuy nhiên, thực tế cho thấy sự kết hợp như vậy rất hiếm khi được sử dụng. Thay vào đó, đặc tính và vị trí của mỗi thiết bị cảm biến đóng vai trò rất quan trọng. Trong nhiều trường hợp, mỗi thiết bị có một nhiệm vụ duy nhất, ví dụ như giám sát các thiết bị cá nhân tại nhà hoặc giám sát ánh sáng của từng cửa sổ. Đối với những ứng dụng như trên, dữ liệu được tổng hợp từ các cảm biến riêng lẽ. Vì vậy, những mạng cảm biến thu thập dữ liệu ở cấp độ mạng, chẳng hạn như TinyDB và Cougar, là thiếu các chức năng cần thiết.

Nhiều ứng dụng mới nổi tạo ra dữ liệu có tiềm năng nhưng nó lại vi phạm quyền riêng tư của nhiều người. Ví dụ, các dữ liệu từ cảm biến năng lượng điện trong các thiết bị, được sử dụng để tiết kiệm năng lượng, nhưng những dữ liệu đó có thể tiết lộ thông tin nhạy cảm về các hoạt động của chủ sở hữu. Do đó, điều quan trọng là chỉ có những dữ liệu thực sự cần thiết mới được lưu trữ trên các thiết bị cảm biến, và những dữ liệu này chỉ truyền cho người có thẩm quyền. Trong nhiều trường hợp, dữ liệu phải trải qua quá trình xử lý, tổng hợp, ví dụ như công suất tiêu thụ trung bình trong mỗi giờ hoặc công suất tiêu thụ lớn nhất trong ngày. Việc thu thập dữ liệu từ từng mẫu cảm biến trong mỗi thiết bị có thể sẽ vi phạm quyền riêng tư, như trong mô hình thu thập dữ liệu truyền thống.

## 3.5 Hướng phát triển công nghệ

*Năng lượng và nguồn điện cung cấp có hạn* : Trong hầu hết các ứng dụng của công nghệ không dây cảm biến, nguồn điện và năng lượng là những yếu tố bị hạn chế. Trong các hệ thống hoạt động bằng pin, năng lượng được lưu trữ trong pin cũng ảnh hưởng tới các hoạt động của mạng. Trong các hệ thống hỗ trợ bởi năng lượng sạch, các nguồn năng lượng thường chỉ sinh ra lượng điện năng rất thấp, vì vậy, ngay cả khi các nguồn cung cấp năng lượng rất dồi dào, điện năng tiêu thụ cũng sẽ bị hạn chế. Đối với những thiết bị được cung cấp nguồn điện năng vô tận, chẳng hạn như đồng hồ công suất điện cho lưới điện thông minh, điện năng tiêu thụ cũng là yếu tố bị hạn chế. Thứ nhất, điện năng tiêu thụ của một thiết bị tác động đến kích thước vật lý và chi phí máy biến áp điện của nó. Thứ hai, để gắn thêm hàng triệu máy đo năng lượng cho lưới điện thông minh nhằm tiết kiệm năng lượng, điện năng tiêu thụ của từng thiết bị phải đủ thấp để có thể tiết kiệm năng lượng.

*Băng thông có hạn* : Băng thông truyền ảnh hưởng đến năng lượng tiêu thụ của máy thu và máy phát vô tuyến điện. Một băng thông truyền cao hơn đòi hỏi phương pháp điều chế và giải điều chế cao hơn, cũng như khả năng đồng bộ cao hơn, tốc độ xung nhịp cao hơn, để đạt được những điều kiện trên thì năng lượng tiêu thụ của bộ thu – phát phải càng tăng.

*Lưu trữ không giới hạn :* Trong khi nguồn điện, năng lượng và băng thông tiếp tục trở thành các nhân tố bị hạn chế, thì xu hướng chính là tăng nhanh chóng kích thước lưu trữ trong khi chi phí giảm. Bởi vì lưu trữ là một thành phần thụ động, kích thước của nó không ảnh hưởng đến sự tiêu thụ điện năng của nó. Đối với các thiết bị cảm biến không dây, kích thước lưu trữ của thiết bị lưu trữ flash hiện nay là rất lớn, đủ đáp ứng cho tất cả các mục đích thực tế, do vậy nó có thể được coi là không có giới hạn: bộ lưu trữ sẽ dư thừa cho đến khi kết thúc thời gian phục vụ của nó . Đối với các hệ thống hoạt động bằng pin, khả năng lưu trữ dễ dàng vượt qua được tuổi thọ pin.

## 3.6 Thách thức

Các mô hình cơ sở dữ liệu đòi hỏi phải có một hệ thống quản lý cơ sở dữ liệu hiệu quả trong mỗi cảm biến. Những thống hiện tại đã được thiết kế cho những nền tảng yêu cầu hoàn toàn khác nhau về mức tiêu thụ năng lượng và tài nguyên hệ thống.

*Năng lượng-hiệu quả để truy vấn và lưu trữ :* Cảm biến là thiết bị có thời gian sống dài và có thể lưu trữ một lượng lớn dữ liệu. Việc truy vấn phải đảm bảo hiệu quả về mặt năng lượng và hoạt động nhanh chóng qua các tập dữ liệu lớn dưới sự hạn chế về tài nguyên của các nút. Hơn nữa, dữ liệu có thể được lưu trữ thường xuyên, đôi khi nhiều lần mỗi giây. Điều này đòi hỏi rằng mức tiêu thụ năng lượng của chip lưu trữ phải rất thấp.

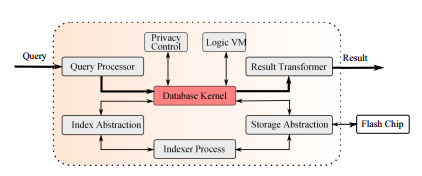
*Ngữ nghĩa lưu trữ vật lý :* Bộ nhớ flash làm phức tạp cho việc quản lý lưu trữ vật lý bằng cách cấm việc cập nhật tại chổ. Cấu trúc lưu trữ được thiết kế cho các ổ đĩa từ trong nhiều trường hợp không sử dụng được trong bộ nhớ flash. Không giống như các đĩa từ, bộ nhớ flash chỉ cho phép bit được lập trình từ 1 đến 0. Để thiết lập lại một bit về 1, một khu vực rộng lớn của các bit liên tiếp phải được xóa đi, thường liên quan đến nhiều kB. Bên cạnh hạn chế này, có những biến thể khác nhau của bộ nhớ flash có quan hệ ràng buộc thêm: NAND flash là trang định hướng, trong khi đó NOR flash là byte theo định hướng. NAND flash thường cho phép một dung lượng lưu trữ lớn hơn và tiết kiệm năng lượng xuất nhập, nhưng có nhiều hạn chế xuất nhập ngữ nghĩa.

# CHƯƠNG 4: ANTELOPE (DATABASE MANAGEMENT SYSTEM)

Antelope là một hệ thống quản lý cơ sở dữ liệu (DBMS) cho các thiết bị cảm biến có nguồn lực hạn chế. Nó cung cấp một tập hợp hệ thống cơ sở dữ liệu quan hệ, bao gồm những thiết lập cho phép xây dựng một cách tự động cơ sở dữ liệu và truy vấn dữ liệu phức tạp. Để có thể thực hiện hiệu quả các truy vấn trên các cơ sỡ dữ liệu lớn, Antelope chứa một cơ chế xuất dữ liệu linh hoạt bao gồm ba thuật toán chỉ số khác nhau. Để có thể thông qua trên nhiều nền tảng khác nhau, và tránh sự phức tạp của việc đối phó với cân bằng hao mòn bộ nhớ flash và những cấu hình chip lưu trữ khác nhau, Antelope thúc đẩy việc lưu trữ trừu tượng được cung cấp bởi hệ thống tập tin Coffee.

## 4.1 Cấu trúc của Antelope

Antelope gồm tám thành phần, như thể hiện trong hình 4.1.

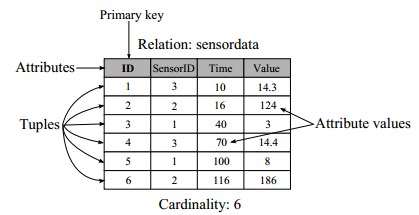


7Hình 4.1. Kiến trúc của Antelope

* *Bộ xử lý truy vấn (Query Processor)*: dùng để phân tích cú pháp truy vấn AQL.
* *Bộ điều khiển riêng**(Privacy Control):* đảm bảo rằng các truy vấn được cho phép.
* *LogicVM:* thực hiện các truy vấn.
* *Hạt nhân cơ sở dữ liệu* ***(****Database Kernel):* giữ logic của cơ sở dữ liệu và phối hợp thực hiện truy vấn.
* *Chỉ số trừu tượng (Index Abstraction):* giữ các chỉ số logic.
* *Bộ lập chỉ mục (Indexer Process):* xây dựng những chỉ số từ dữ liệu hiện có.
* *Bộ lưu trữ trừu tượng (Storage Abstraction):* trong đó có chứa tất cả các logic lưu trữ**.**
* *Chuyển đổi kết quả (Result Transformer):* trong đó trình bày các kết quả của một truy vấn theo cách mà làm cho nó dễ dàng để sử dụng bởi các chương trình.

## 4.2 Những thuật ngữ liên quan

Antelope sử dụng các thuật ngữ tiêu chuẩn từ các trường cơ sở dữ liệu quan hệ. Theo thuật ngữ này, *tuple* là một tập hợp các giá trị *attribute*, nơi mỗi giá trị thuộc về một *attribute*. Mỗi *attribute* là một miền chứa các giá trị có kiểu dữ liệu cụ thể. Một tập hợp các *tuple* có các *attribute* tương tự nhau được gọi là *relation*. Trong *relation*, mỗi *tuple* sẽ có một khóa duy nhất để định danh cho chính nó. Số lượng *tuple* trong *relation* được gọi là số các yếu tố trong *relation* đó. Chính vì vậy, một *relation* đôi khi có thể được gọi là một bảng, một *attribute* tương ứng với một cột, và một *tuple* tương ứng với một hàng. Một ví dụ về *relation* có chứa mẫu được thu thập từ các cảm biến khác nhau được mô tả trong hình 4.2.



8Hình 4.2. Mối quan hệ có chứa các mẫu lấy từ một bộ cảm biến tại các điểm khác nhau trong thời gian.

## 4.3 Ngôn ngữ truy vấn

Ngôn ngữ truy vấn của Antelope được gọi là AQL, được sử dụng để xây dựng và truy vấn cơ sở dữ liệu. Cách điển hình của việc sử dụng Antelope trong một thiết bị cảm biến, đầu tiên để xác định một cơ sở dữ liệu bao gồm một hay nhiều *relation*. Dữ liệu có thể được mô hình hóa bằng cách sử dụng các nguyên tắc thiết kế có cơ sở cho cơ sở dữ liệu quan hệ, như đơn giản hóa dữ liệu và lập kế hoạch cho những loại truy vấn mà hệ thống có thể xử lý một cách hiệu quả.

Bảng 4.1 đưa ra một cái nhìn tổng quan về các hoạt động sẵn có trong AQL. Người dùng xác định các cơ sở dữ liệu bằng cách sử dụng một tập các câu lệnh được cung cấp bởi ngôn ngữ truy vấn của Antelope. Cơ sở dữ liệu có thể được truy vấn bằng cách sử dụngmột tập các hoạt động quan hệ *(relational operations)*. Một số câu lệnh cócú pháp tương tự SQL *(Structured Query Language)*, tuy nhiên AQL và SQL cũng có nhiều sự khác nhau đáng kể do mục tiêu của từng ngôn ngữ. SQL nhằm vào một loạt các hệ thống cơ sở dữ liệu, bao gồm cả các hệ thống cao cấp với nhiều cấp bậc có nguồn tài nguyên lớn hơn rất nhiều so với các thiết bị cảm biến. Ngược lại, AQL được thiết kế cho các hệ thống với tài nguyên phần cứng khiêm tốn hơn. Do đó, những chức năng phức tạp của SQL như phần mở rộng thủ tục, kích hoạt, và các giao dịch được loại trừ. Hơn nữa, AQL không thực sự là một tập hợp con của SQL, chúng ta có thể nhận thấy chủ yếu trong việc phân tích các hoạt động JOIN và SELECT, một thực thi đơn giản của Antelope.

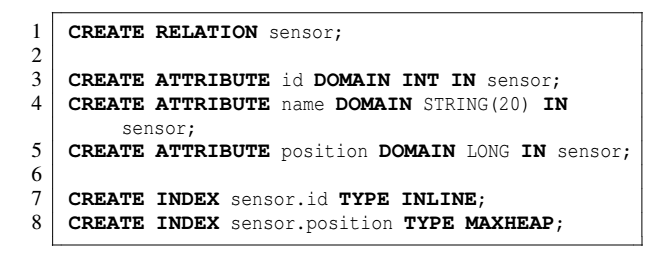
1 Bảng 4.1: Các hoạt động trong cơ sở dữ liệu Antelope

|  |  |
| --- | --- |
| Hoạt động | Mục đích |
| INSERT  REMOVE  SELECT  JOIN | Chèn một *tuple* vào trong một *relation*.  Xóa *tuple* phù hợp với một điều kiện.  Chọn các *tuple* và các *attribute* cho dự án.  Ghép hai *relation* dựa trên một điều kiện nhất định. |
| CREATE RELATION  REMOVE RELATION  CREATE ATTRIBUTE  CREATE INDEX  REMOVE INDEX | Tạo một *relation* rỗng.  Xóa *relation* và tất cả các chỉ số liên quan của nó  Tạo một *attribute* cho *relation*.  Tạo một chỉ số cho *attribute*.  Xóa một chỉ số *attribute*. |

Một *relation* được thiết lập bằng cách sử dụng một tập các câu lệnh trong ngôn ngữ truy vấn của Antelope**.** Các lệnh bao gồm tạo ra các *attribute* và các miền giá trị tương ứng, một tập giá trị (được gọi là *tuple*) tương ứng với các *attribute* sẽ được chèn vào *relation*, và cuối cùng là thiết lập loại chỉ số cho các *attribute*.

Đối với mỗi hoạt động TẠO (*CREATE*) thì sẽ có một hoạt động XÓA (*REMOVE***)** tương ứng. Tuy nhiên, để giảm bớt sự phức tạp trong việc xử lý ở lớp vật lý của dữ liệu, Antelope hạn chế hoạt động TẠO và XÓA các *attribute* trước khi tuple đầu tiên được chèn vào. Tuy nhiên, chỉ số được lưu trữ riêng biệt trên một cơ sở mỗi *attribute* vì vậy nó có thể được TẠO và XÓA tự động.

Một ví dụ chi tiết về việc tạo một cơ sở dữ liệu được biểu diễn trên hình 4.3. Ở dòng 1, một *relation* được tạo ra, thủ tục này kéo theo lớp lưu trữ vật lý, trong đó Antelope sẽ đặt một cấu trúc dữ liệu quan hệ, cấu trúc này sẽ được tải vào RAM bất cứ khi nào hệ thống truy cập vào *relation* lần đầu tiên sau khi khởi động. Ở dòng 3-5, chúng ta tạo ra ba thuộc tín trong *relation.* Miền giá trị được biểu diễn chi tiết như INT, LONG, STRING. Các *attribute* của miền STRING phải được kèm theo một điều kiện về chiều dài của chuỗi. Chúng ta có thể thấy ở dòng 4, khi *attribute* *name* có chiều dài lớn nhất là 20 ký tự. Cuối cùng, ở dòng 7-8, chúng ta tạo hai chỉ số để tăng khả năng truy vấn dữ liệu có giá trị *attribute* phù hợp với các điều kiện tìm kiếm nhất định.



**9** Hình 4.3: Một cơ sở dữ liệu trong Antelope được tạo ra bởi một tập các thao tác, đầu tiên tạo *relation,* tiếp theo tạo các *attribute* và cuối cùng là tạo các chỉ số

Khi một *relation* được thiết lập, thì nó đã sẵn sàng để chứa các *tuple* dữ liệu. Đầu tiên, khi chèn một *tuple*, Antelope sẽ kiểm tra xem các giá trị của *tuple* có thõa mãn với miền giá trị tương ứng của *attribute* trong *relation* không. Các biểu diễn trừu tượngtrong AQL sau đó được chuyển đổi thành biểu diễn vật lícủa nó, sự biểu diễn đó là một bản ghi bằng mảng byte nhỏ gọn của giá trị *attribute*. Nếu *attribute* được lập chỉ số, Antelope gọi hàm trừu tượng cho việc chèn chỉ số, các hàm sẽ chuyển tiếp hàm gọi đến các thành phần *attribute* thực sự đã được lựa chọn cho *attribute*. Trong bước cuối cùng, tuple sẽ được truyền vào bộ lưu trữ trừu tượng nơi thực hiện viêc ghi tuple vào bộ nhớ ổn định.

Các hoạt động của cơ sở dữ liệu như SELECT, JOIN, và REMOVE được thực hiện trong hai phần: phần chuẩn bị, trong đó quá trình xử lý được thiết lập và phần lặp lại, trong đó tập hợp các kết quả của các *tuple* được xử lí bởi cả Antelope và bởi người sử dụng.

Số lượng tối thiểu của việc xử lý bao gồm cả việc đọc *tuple* từ bộ lưu trữ vào một bộ đệm RAM. Tuy nhiên, thông thường có thể in các tuple trên cổng nối tiếp, gửi qua radio, hoặc tính toán thống kê trên các giá trị *attribute*. Theo mặc định, các kết quả truy vấn được gán cho một kết quả quan hệ ảo, nơi mà các tuple không được lưu trữ, thay vì chuyển từng cái một vào trong bộ đệm của RAM. Nếu cần dùng những truy vấn tiếp theo trên kết quả, người dùng có thể gán kết quả vào một quan hệ ổn định mới.

*Hoạt động lựa chọn (select) :* Hoạt độngSELECT cho phép người dùng lựa chọn một tập hợp con của các *tuple* trong một *relation*. Việc lựa chọn được quyết định bởi một điều kiện xác định. Ngoài ra, các *attribute* được thiết lập cho kết quả quan hệ có thể được sử dụng**.** Giá trị *attribute* có thể được tổng hợp dựa trên toàn bộ các *tuple* phù hợp với điều kiện lựa chọn. Các hàm tập hợp gồm COUNT, MAX, MEAN, MIN, và SUM, mỗi hàm sẽ được cập nhật với mỗi *tuple* được xử lý sau khi Antelope đã xác nhận rằng *tuple* đáp ứng các điều kiện của hoạt động SELECT. Trong trường hợp này, kết quả quan hệ sẽ bao gồm các *tuple* đơn lẽ chứa các dữ liệu xác định. Một ví dụ của truy vấn SELECT bao gồm các quan hệ lựa chọn và tập hợp được trình bày sau đây:

**SELECT** MEAN(humidity), MAX(humidity) **FROM** samples **WHERE**

year = 2015 **AND** month >= 6 **AND** month <= 8;

Trong truy vấn này, chúng ta lấy giá trị trung bình và giá trị lớn nhất của độ ẩm trong mùa hè năm 2015, Antelope xác định truy vấn thông qua tất cả chỉ số có sẵn, và chuyển kết quả tới tới người dùng theo yêu cầu. Ngược lại, các mạng cảm biến thường phải truyền định kì toàn bộ giá trị tới nút thu (sink)**,** có thể qua nhiều hop. Đây là một trong những điều phản ánh chức năng của Antelope.

*Hoạt động tham gia (join):* Khi một truy xuất bao gồm hai *relation*, hoạt động JOIN thực sự hữu ích. Xét ví dụ dưới đây, người sử dụng muốn đếm số vùng của *contacts* có giá trị thời gian là April 2015 và có địa chỉ Ipv6 là aaaa::1.

contacts <- **JOIN** device, rendezvous **ON** device\_id **PROJECT**

address, year, mon;

**SELECT** COUNT(\*) **FROM** contacts **WHERE** year = 2011 **AND** mon

= 4 **AND** address = ’aaaa::1’;

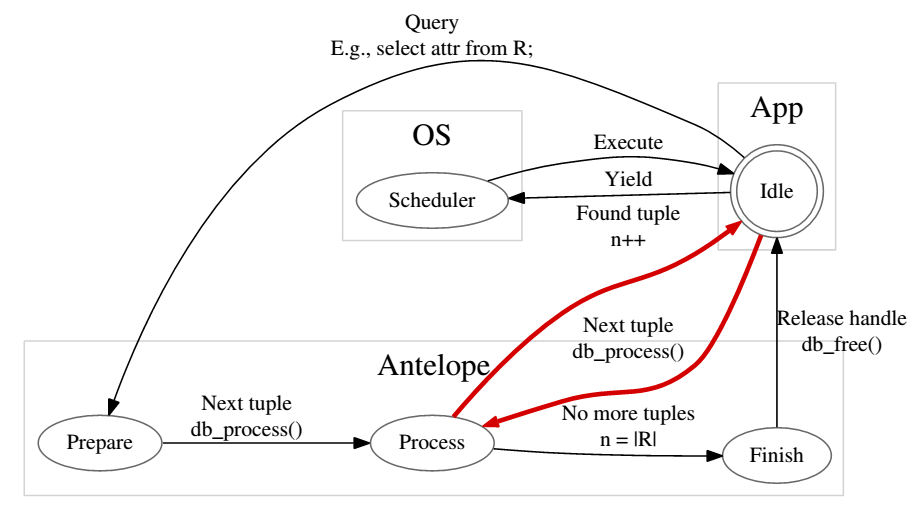
JOIN thực sự là một hoạt động rất mạnh: nó hợp nhất các tập con của hai *relation* cùng với một vài điều kiện. Mặc dù JOIN không liên quan đến việc xử lí luận lý của mỗi *tuple*, nhưng nó cần thiết để xem xét các tuple của hai relation. Do đó, mức độ phức tạp có thể lên tới O (| L || R |), trong đó L biểu thị *relation* bên trái và R biểu thị *relation* bên phải. Đây là sự phức tạp của một thuật toán tham gia đơn giản - đó là các vòng lặp lồng nhau - với mỗi *tuple* trong *relation* bên trái nó phải quét tất cả các *tuple* trong các *relation* bên phải.

Chất lượng thực hiện của các hoạt động JOIN trong Antelope bị hạn chế bởi không gian của bộ đệm trong các thiết bị cảm biến. Vì vậy, chúng ta chỉ cho phép JOIN khi các *attribute* tham gia ở *relation* bên phải được đánh chỉ số. Khi thực hiện JOIN, Antelope lặp qua mỗi *tuple* trong *relation* bên trái. Nó sẽ lấy ra những giá trị của *attribute* tham gia trong tuple đó và tìm kiếm chỉ số của *attribute* tương ứng trong *relation* bên phải cho giá trị này. Mô-đun chỉ số sẽ trả về một vòng lặp qua tất cả các *tuple* thỏa mãn, những *tuple* này đã được xử lí và bao gồm ở trong kết quả tham gia *relation.*

*Hoạt động xóa (remove):* các *tuple* dữ liệu có thể được xóa thông qua các hoạt động REMOVE. Hoạt động này chọn các điều kiện tương tự như cú pháp đã được sử dụng trong hoạt động SELECT, và xóa toàn bộ các *tuple* thỏa mãn các điều kiện đó. Bên trong, Antelope dịch hoạt động này thành một truy vấn SELECT, với một điều kiện nghịch đảo, và gán kết quả vào một *relation* tạm thời. Các *relation* cũ sau đó được xóa, và các *relation* tạm thời sẽ lấy tên của các *relation* cũ. Do đó, chi phí của hoạt động REMOVE tương đương với một hoạt động SELECT có phân công, đưa ra một số lượng *tuple* để xử lí .

## 4.4 Hạt nhân cơ sở dữ liệu

Kiến trúc của Antelope xoay quanh hạt nhân cơ sở dữ liệu. Thành phần này đánh giá tất cả các hoạt động cơ sở dữ liệu quan hệ của AQL. Hạt nhân cơ sở dữ liệu thường được gọi ra bằng các mô-đun thực thi truy vấn sau khi bộ phân tích cú pháp AQL đã dịch hoàn thành các hoạt động, từ biểu diễn văn bản thành biểu diễn của một dữ liệu trừu tượng có khả năng tính toán hiệu quả hơn. Nó cho phép thực hiện nhiều truy vấn đồng thời bằng cách cung cấp một đối tượng truy vấn đến người sử dụng.



10Hình 4.4: Mô hình làm thực thi cơ sở dữ liệu

Hạt nhân cơ sở dữ liệu được thiết kế trên mô hình biến lặp, bởi vì nó đòi hỏi tại một thời điểm chỉ có một tuple được lưu trữ vào trong bộ nhớ, và bởi nó có những đặc tính khiến cho nó phù hợp với lịch trình hợp tác. Mỗi truy vấn yêu cầu các xử lí có thể bị tạm dừng giữa một *tuple* được xử lí, cho phép các ứng dụng có thể từ bỏ quền điều khiển từ bộ định thời. Hình 4.4 biểu diễn một luồng làm việc đặc trưng của một truy vấn cơ sở dữ liệu. Trong đó, ứng dụng lấy các *tuple* đã được xử lí, và có thể kiểm soát năng suất tùy chọn tới hệ điều hành của bộ định thời. Mô hình xử lí này thích hợp với các hệ thống nhúng, những hệ thống thường sử dụng

## 4.5 Logic VM

LogicVM là một máy ảo dùng để đánh giá các mệnh đề logic. Mục đích chủ yếu của LogicVM là giúp nâng cao hiệu suất của các hoạt động SELECT trong *relation*. LogicVM trình bày ngắn gọn bằng định dạng bytecode sử dụng ký hiệu Polish. Ký hiệu Polish tạo điều kiện thực hiện đệ quy qua stack cho mỗi trường hợp của các máy ảo. LogicVM đánh giá các trạng thái logic được biên dịch, có thể chứa số học và xuất ra một kết quả Boolen

### 4.5.1 Ngôn ngữ bytecode

Logic VM có thể đánh giá các phép toán thuộc về một trong ba loại: liên kết luận lý (∧, ∨, ¬), toán tử quan hệ (≤, <, ≥, >, =, ≠), và toán tử số học (+, −, ×, ÷). Liên kết luận lý lấy các toán hạng có thể thuộc về toán tử quan hệ hoặc liên kết luận lý. Toán tử quan hệ chấp nhận các toán hạng là những biểu thức số học có giá trị là các số nguyên, chuỗi, kiểu float; và các biến. Các biến số phản ánh các *attribute* của relation.

### 4.5.2 Sự thực thi

Bằng cách sử dụng một mô hình thực hiện dựa trên ngăn xếp, có thể cung cấp chính xác bộ nhớ nó cần trên ngăn xếp cho từng trường hợp cụ thể của thiết bị, hơn là cấp phát vùng nhớ tự động hoặc cấp phát bộ nhớ đệm tĩnh nảy sinh bởi các mô hình cấp thiết. Một trong những điều thúc đẩy cho sự lựa chọn này chính là các mô hình thực hiện dựa trên ngăn xếp sẽ đơn giản hơn bằng cách sử dụng tương đối ít các lệnh xử lý: kích thước mã biên dịch cho LogicVM ít hơn 3 kB cho bộ vi xử lý MSP430 16-bit.

Quá trình thực hiện trong ngăn xếp được tạo ra bởi bộ dịch AQL sử dụng một API được đưa ra bởi LogicVM. Trước khi xây dựng quá trình thực hiện trong ngăn xếp, bộ dịch AQL ghi các biến được tham chiếu bởi các biểu thức logic. Sau khi bộ phân tích đã hoàn thành xây dựng thực hiện trong ngăn xếp, Hạt nhân cơ sở dữ liệu thực hiện các hoạt động SELECT, và với mỗi *tuple* lấy ra, tất cả các giá trị của biến sẽ được thay thế và thực hiện các LogicVM một lần nữa. Kết quả Boolean của sự thực thi của một máy ảo quyết định xem hạt nhân cơ sở dữ liệu bao gồm *tuple* ở trong quan hệ kết quả.

### Tầm thuộc tính

Để tăng hiệu suất xử lý truy vấn, chúng ta giới thiệu một thuật toán phân tích logic trong LogicVM. Thuật toán này có thể suy ra các phạm vi chấp nhận được đối với tất cả các *attribute* trong một biểu thức logic. Hạt nhân cơ sở dữ liệu sử dụng thông tin này nhằm giới hạn các bộ dữ liệu để tìm kiếm thông qua điều kiện thỏa mãn một truy vấn. Các thông tin tạo ra bởi thuật toán này được cung cấp cho bất kỳ thuật toán chỉ số nào sử dụng cho các *attribute* của biểu thức, cho phép các truy vấn phức tạp được thực thi một cách hiệu quả. Không giống như tìm kiếm bình đẳng và tìm kiếm trong phạm vi rõ ràng, những phân tích như vậy là không đáng kể, một truy vấn SELECT trong AQL có thể chứa một biểu thức logic phức tạp hơn; ví dụ (a > 1000 ∧ a < 2000)- ∧ (b + a = 1500).

Biểu thức như vậy mặc nhiên nêu một loạt các giá trị có thể cho từng *attribute* bên trong chúng. Đối với mỗi *attribute* trong biểu thức, LogicVM là có thể thu hẹp đáng kể xuống phạm vi cụ thể hơn, trong đó tất cả các giá trị thỏa mãn các biểu thức có thể được tìm thấy. Vấn đề này liên quan chặt chẽ đến thõa mãn vấn đề Boolean (SAT), đó là NP – đầy đủ. Tuy nhiên, bằng cách hạn chế các biểu thức và dựa vào một vài mệnh đề AQL, các thuật toán hoạt động trong một thời gian nhỏ trong tổng thời gian xử tuple.

LogicVM thực thực hiện tầm suy luận trực tiếp trên VM bytecode. Do đó nó có thể sử dụng các thuật toán đệ quy giống như được sử dụng khi thực hiện một biểu thức logic. Các thuật toán suy luận trả về tập hợp của tầm chấp nhận được cho các *attribute* của từng phân lớp. Sự suy luận bắt đầu từ mức sâu nhất của biểu thức, và kết hợp các dãy kết quả cho đến khi đạt mức đỉnh của biểu thức. Các phương pháp để sử dụng cho phạm vi kết hợp phụ thuộc vào liên kết luận lý trong mệnh đề là ∧ hoặc ∨. Đối với các phương pháp trước đây, các bộ được sáp nhập bằng cách tạo ra một giao điểm của chúng, trong khi sau này, chúng được sáp nhập bằng cách tạo ra một tập hợp.

Ví dụ, nếu chúng ta muốn tìm các tầm thích hợp cho α và β trong điều kiện α > 10 ∧ α ≤ 20 ∧ β> 100, chúng ta kết luận rằng β ∈ {101, ..., max β}. Sau đó chúng ta xử lý mệnh đề con α ≤ 20, với kết quả α ∈ {αmin, ..., 20}. Trong mệnh đề con cuối cùng, α > 10, chúng ta có được ∈ α {11, ..., αmax}. Như vậy, α ∈ {αmin, ..., 20} ∩ {11, ..., αmax} = {11, ..., 20}. Các giá trị lớn nhất và nhỏ nhất của các giá trị tương ứng với α và β sau đó có thể được sử dụng bởi các hạt nhân cơ sở dữ liệu để xác định giá trị tìm kiếm trong bất kỳ chỉ số nào đang có về các *attribute*.

## Hiệu quả năng lượng của việc đánh chỉ số

Một chỉ số là một cấu trúc dữ liệu phụ với mục đích ban đầu của nó là tối ưu hóa quá trình thực hiện truy vấn cơ sở dữ liệu thông qua việc giới hạn các tuple mà nó phải xử lý. Người quản trị cơ sở dữ liệu chọn những *attribute* cần thiết phải tạo chỉ số dựa mục đích sử dụng của cơ sở dữ liệu. Một chỉ số lưu trữ một cặp từ khóa - giá trị, trong đó từ khóa biểu thị một giá trị *attribute*, và các giá trị biểu thị một định danh của *tuple* chứa giá trị *attribute* đó.

Để đạt được sự hiệu quả của việc đánh chỉ số trong các thiết bị cảm biến là một thách thức, bởi vì các chỉ số phải tốn một bộ nhớ nhỏ và một mô hình xuất nhập có thể giảm thiểu năng lượng tiêu thụ và sự hao mòn trên bộ nhớ flash. Chỉ số thường được thiết kế cho những cơ sở dữ liệu lớn, và giả định rằng các mẫu xuất nhập dựa trên mô hình khối của các đĩa từ thì chỉ số được loại bỏ vì bộ nhớ flash hạn chế sửa đổi dữ liệu bằng cách ghi văn bản.Thay vào đó, chỉ số như FlashDB, Lazy-Adaptive Tree, và chỉ số MicroHash đã được phát triển cho bộ nhớ flash. Vô số các thuật toán chỉ số (với mỗi thuật toán có sự đánh đổi khác nhau về độ phức tạp về không gian, năng lượng tiêu thụ, và hiệu quả lưu trữ) có thể thúc đẩy việc cung cấp một ngôn ngữ truy vấn và một giao diện lập trình ứng dụng có thể tách các ứng dụng từ việc thực hiện đánh chỉ số.

Với mục đích này, chúng ta không những phải đảm bảo các chỉ số có thể tự động được xây dựng và hủy bỏ trong thời gian chạy, mà còn phải đảm bảo tất cả các chỉ số có thể được truy cập bằng cách sử dụng cùng một loại giao diện truy vấn. Đây là một sự khác biệt có lợi từ việc triển khai chỉ số chuyên dụng cho các thiết bị cảm biến, yêu cầu các ứng dụng phải được liên kết chặt chẽ với một thuật toán chỉ số nhất định.

### 4.6.1 Những hướng thiết kết

Để tách rời việc triển khai chỉ số từ các hạt nhân cơ sở dữ liệu, tất cả các chỉ số thực hiện một giao diện lập trình tổng quát. Giao diện này được điều chỉnh theo mô hình lặp, nó được sử dụng rộng rãi trong hạt nhân cơ sở dữ liệu. Giao diện này bao gồm các hoạt động như : Creat, Load, Destroy, Insert, Delete, Get Iterator, và GetNext.

Bảng 4.2 liệt kê ba thuật toán chỉ số khác có sẵn trong Antelope. Mỗi một chỉ số thực hiện các giao diện lập trình ứng dụng đánh chỉ số theo từng cách khác nhau. Chỉ số MaxHeap được tối ưu hóa để giảm sự hao mòn bộ nhớ flash và năng lượng tiêu thụ. Chỉ số Inline được tối ưu hóa cho chuỗi dữ liệu cảm biến và bộ dữ liệu đã được sắp lệnh và do đó thích hợp với cho các trường cảm biến đặc trưng.Chỉ số Hash chỉ có thể lưu trữ một lượng nhỏ các cặp từ khóa-giá trị trong bộ nhớ RAM, vì vậy chỉ số này rất lý tưởng cho các *relation* nhỏ được truy vấn thường xuyên.

2Bảng 4.2: Các thuật toán chỉ số trong Antelope.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thuật toán | Cấu trúc | Vị trí | Không gian | Tìm kiếm |
| MaxHeap  Inline  Hash | Dynamic  Dynamic  Static | External  Inline  RAM | O(N)  O(1)  O(N) | O(log N)  O(log N)  O(1) |

### 4.6.2 Chỉ số MaxHeap

Chỉ số MaxHeap là một chỉ số mới cómục đích chung là sử dụng cho NOR flash và thẻ nhớ SD, nó hỗ trợ cả truy vấn bằng và truy vấn phạm vi.Các chỉ số mới nhất (ví dụ: FlashDB và Lazy- Adaptive Tree )thường được thiết kế cho flash NAND. Cả hai được xây dựng trên cấu trúc cây cân bằng, nên nó yêu cầu phải rằng những yêu tố chèn thêm vào đôi khi bị di chuyển ở trong flash. Vì khả năng ghi của NAND flash khá hạn chế, nên họ sử dụng cấu trúc ghi để có thể đặt lên một bộ nhớ lớn khi sử dụng cho bộ nhớ flash. Số lượng của những từ khóa được đánh chỉ số bị giới hạn rất nhiều trên những nền tảng thiết bị như TelosB, với dung lượng RAM của nó là 10 kb. Khi lập chỉ số có 30,000 từ khóa với FlashDB, bộ nhớ được sử dụng sẽ vượt quá bộ nhớ này của RAM. Ngược lại, chỉ số MaxHeap được thiết kế chủ yếu cho các loại bộ nhớ bị hạn chế có thể ghi thoải mái hơn so với NAND flash, mà nó yêu cầu một bộ nhớ nhỏ hơn.

Chỉ số MaxHeap sử dụng một cấu trúc khối xếp nhị phân lớn nhất, cho phép ánh xạ tự nhiên tới các tập tin mở rộng động trong hệ thống tập tin nằm bên dưới. Đối với mỗi *attribute* được đánh chỉ số, các chỉ số thực thi lưu trữ hai tập tin: mô tả khối xếp và một vùng chứa bộnén (*bucket*). Vùng chứa bộ nén chứa các bộ nén, trong bộ nén lưu trưc các cặp từ khóa – giá trị thực sự của chỉ số.

Khi một cặp từ khóa-giá trị được chèn vào chỉ số, nó được đưa vào bộ nén có phạm vi giá trị nhỏ nhất bao gồm cả từ khóa. Khối xếp là một cấu trúc bền vững, nó chỉ cho phép mở rộng xuống phía dưới, những từ khóa chèn sẽ được phân tán đồng đều qua tất cả nút trên khối xếp để tối đa hóa không gian sử dụng. Tuy nhiên, từ khóa có thể được phân bố không đồng đều trên các miền *attribute* hoặc được đưa vào với một thứ tự không rõ ràng chẳng hạn như một dãy tăng. Chúng ta xử lý vấn đề này bằng cách chọn một bộ nén dựa trên các giá trị khóa *hash*, nhưng lưu trữ từ khóa *unhashed*. Các cặp từ khóa-giá trị sẽ được lưu trữ trong các vị trí còn trống kế tiếp trong bộ nén đã được lựa chọn.

Khi một bộ nén đã được sử dụng hết, hai khối xếp có thứ tự ưu tiên mới sẽ được phân bổ thêm vào bên dưới nó để tách không gian của bộ nén gốc thành hai nửa. Các từ khóa được lưu trữ trong bộ nén ba mẹ được giữ lại, nhưng quá trình chèn thêm được thực hiện trong các bộ nén bên dưới nó. Tuy nhiên, một mặt tiêu cực bị ảnh hưởng của thiết kế này là truy vấn thường yêu cầu xử lý cùng lúc của nhiều bộ nén; khi tìm kiếm chỉ số cho một từ khóa, chúng ta bắt đầu từ bộ nén thấp nhất trong cấu trúc cây với phạm vi tìm kiếm nhỏ nhất với từ khóa lựa chọn.

Chỉ số MaxHeap cần O (n + 4k) byte của bộ nhớ, trong đó n là số nút trong bộ cấu trúc có thứ tự ưu tiên và k là số lượng các từ khóa trong mỗi nút. Số nút là 2m - 1, m ≥ 1, trong đó m là tham số cấu hình trong lúc biên soạn. Ví dụ, để chứa 30.000 từ khóa, chúng ta có thể thiết lập m = 8 và k = 128, kết quả trong một chân bộ nhớ chứa 854 byte các chỉ số.

### 4.6.3 Chỉ số InLine

Trong trường cảm biến, thứ tự của các *tuple* chèn thêm đôi khi có thể bị hạn chế trở thành một dãy tăng đơn điệu. Thực tế, một chuỗi dài của các mẫu cảm biến thông thường được tạo chỉ số bởi *attribute* nhãn thời gian. Một trường hợp khác là khi người quản trị cơ sở dữ liệu kiểm soát thứ tự chèn. Ví dụ, khi chèn mô tả của các chip khác nhau trên nút, hoặc thông tin về các nút lân cận khác nhau trong một mạng mà đã được biết đến trước khi triển khai.

Chúng ta tận dụng các ràng buộc về thứ tự để tạo ra một chỉ số gọn nhẹ có tên chỉ số Inline. Bản chất của chỉ số Inline là một vỏ bọc của thuật toán tìm kiếm, chỉ số Inline hoạt động không sử dụng lưu trữ bên ngoài, và có một bộ nhớ tiêu tốn cố định bất kể số lượng dữ liệu được đánh chỉ số. Chỉ số Inline hiện đang được thực hiện như là một biến thể của tìm kiếm nhị phân, nó tìm kiếm sự bắt đầu và kết thúc của một khoảng trong toàn bộ các tuple trong một *relation* được lưu trữ. Do đó, chỉ số Inline phá vỡ hạt nhân cơ sở dữ liệu bằng cách truy cập trực tiếp vào lớp lưu trữ trừu tượng, từ đó nó lấy các *tuple* cụ thể được tạo chỉ số bởi số lượng các hàng trong lưu trữ vật lý. Vì không có cơ cấu chỉ số bên ngoài hoặc nội bộ, chỉ số Inline có có không gian lưu trữ *O(1)***-** không gian dành cho chi phí phụ.

Tương tự như các chỉ số Inline, chỉ số MicroHash được thiết kế dựa trên ý tưởng của việc sắp xếp các chỉ số theo thứ tự thời gian của các mẫu cảm biến. Tuy nhiên, sự khác biệt của MicroHash là nó chuyên hướng tới một loại bộ nhớ flash và cấu trúc lưu trữ dữ liệu cảm biến nhất định, trong khi đó chỉ số Inline được sử dụng để tránh một sự phức tạp trong quá trình thực hiện bằng cách sử dụng các bộ nhớ lưu trữ trừu tượng để chia sẻ với các thành phần khác của Antelope, và Inline được thiết kế để chỉ số của *attribute* bất kỳ chèn vào có sự ràng buộc về thứ tự.

### 4.6.4 Chỉ số Hash

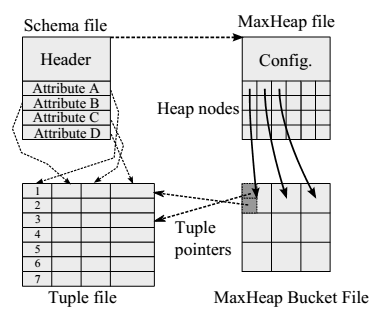
Để thay thế cho hai loại chỉ số nói trên, trong đó yêu cầu truy cập vào bộ nhớ flash bên dưới, chỉ số Hash cung cấp công cụ tìm kiếm nhanh cho các giá trị *attribute* bằng cách sử dụng bảng hash được lưu trong bộ nhớ RAM. Loại chỉ số này rất lý tưởng cho các ứng dụng tạo sử dụng các truy vấn thường xuyên đến những *relation* có số *tuple* thấp - giới hạn số tuple chính xác nhất là được cấu hình tại thời gian biên dịch, nhưng mặc định là 100 *tuple* hoặc ít hơn. Nếu thiết lập lại hệ thống máy chủ, Antelope sẽ tự động phục hồi lại các chỉ số hash của *relation* mà chúng sở hữu trong lần truy cập đầu tiên.

## 4.7 Quá trình thực hiện trong Antelope

Antelope được thực hiện bằng ngôn ngữ lập trình C và một phần nhỏ của dịch vụ phân cấp hệ điều hành (OS - level) được cung cấp bởi hệ điều hành Contiki. Triển khai Antelope đến hệ điều hành khác cho những thiết bị cảm biến có nguồn tài nguyên hạn chế là vấn đề chính của việc thiết lập lớp lưu trữ tới tập tin hệ thống của các hệ điều hành.

### 4.7.1 Giao diện lưu trữ

Do sự đa dạng của các nền tảng phần cứng được triển khai trong các mạng cảm biến, Antelope được thiết kế để có thể di động trên một máy chủ của hệ thống tập tin. Do đó cần phải cung cấp một lớp lưu trữ giữa các hệ thống cơ sở dữ liệu và hệ thống tập tin nằm bên dưới nó. Lớp này có chức năng điều chỉnh các hoạt động lưu trữ của Antelope. Bên cạnh chức năng chung là để quản lý siêu dữ liệu và nội dung cơ sở dữ liệu, các API lưu trữ bản đồ trực tiếp đến các mô hình lặp được sử dụng ở các lớp cao hơn trong hạt nhân cơ sở dữ liệu.



11Hình 4.5. Bố trí lưu trữ trong Antelope với một *relation* và chỉ số MaxHeap tương ứng

### 4.7.2 Lưu trữ trừ tượng

Lưu trữ này được thực hiện bằng cách sử dụng hệ thống tập tin Coffee. Ngoài ra còn có các hệ thống tập tin khác cho các thiết bị nguồn lực hạn chế mà có thể cung cấp hầu hết các tính năng cần thiết cho Antelope là ELF và TFFS, nhưng Antelope sử dụng Coffee vì nó có bộ nhớ liên tục và được chia nhỏ phù hợp với kích thước của tập tin. Một cách bố trí lưu trữ cơ bản, bao gồm một *relation* duy nhất với chỉ số MaxHeap tương ứng được minh họa trong hình 4.5. Antelope tận dụng khả năng của hệ thống tập tin để quản lý được nhiều tập tin hơn. Bằng cách tách dữ liệu theo cách này, nó cũng có thêm cơ hội để tối ưu hóa việc truy cập vào các tập tin cá nhân bởi vì chúng có thể được cấu hình theo tiểu sử xuất nhập của tải làm việc trong Coffee.

### 4.7.3 Điều chỉnh hệ thống tập tin

Coffee cung cấp một phương pháp cấu trúc ghi từng phần để xử lý và sửa chữa các tập tin, nhưng chức năng này là không cần thiết cho một thuật toán chỉ số thiết kế đặc biệt cho bộ nhớ flash. Các thuật toán chỉ số có thể sử dụng một cấu trúc ghi nội bộ, hoặc nó có thể sử dụng bộ nhớ flash như một cách để xóa bỏ khi cần. Chỉ số MaxHeap sử dụng chế độ này để tránh phát sinh các chi phí đối với các bản ghi siêu nhỏ của Coffee khi ghi cùng với khối xếp bộ nén trên NOR flash.

### 4.7.4 Tính nhất quán và phục hồi lỗi

Các lớp lưu trữ có trách nhiệm kiểm tra tính chính xác của dữ liệu và báo cáo bất kỳ lỗi nào cho hạt nhân cơ sở dữ liệu. Những giao dịch phức tạp được thực hiện cho một hệ thống tập tin cụ thể như : giao dịch hỗ trợ, kiểm tra tính chính xác, và khôi phục lỗi đã được xác định. Những giao dịch này chỉ có trong TFFS (*Transactional Flash File System*).

Nếu một lỗi phần mềm dẫn đến làm hỏng cơ sở dữ liệu, hoặc một hệ thống máy chủ bị khởi động lại trong quá trình thực hiện cơ sở dữ liệu, tập tin được gọi ra có thể là không hoàn thành hoặc dữ liệu hỏng. Dự trên sự khởi động lại của hệ thống cơ sở dữ liệu, lớp lưu trữ sẽ phát hiện những *tuple* đó là đánh dấu chúng nhãn dữ liệu hỏng. Trong trường hợp quá trình chèn giá trị cho các *attribute* đã được đánh chỉ số, rất nhiều tập tin cần được thay đổi. Nếu chỉ số chèn vào bị hỏng, hạt nhân cơ sở dữ liệu sẽ dừng ngay hoạt động INSERT và trả về lỗi. Nếu nó thành công, nhưng sự thêm vào của miêu tả vật lí của *tuple* bên trong tập tin *tuple* bị hỏng, chỉ số sẽ được chỉ tới *tuple* bị hỏng. Trong khi lấy thêm những

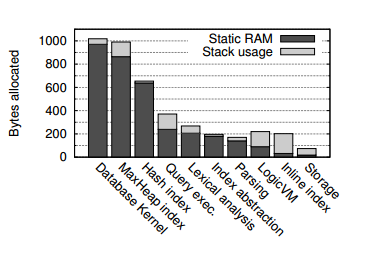
### 4.7.5 Sự phức tạp trong quá trình thực hiện

Bảng 4.3 biểu diễn bộ nhớ ROM và số lượng SLOC của hệ thống tập tin. Với tất cả các chức năng được đính kèm, tổng kích thước ROM là 17 kB, nghĩa là thấp hơn giới hạn của một số các thiết bị cảm biến có nguồn tài nguyên hạn chế nhất, ví dụ như Tmote Sky.

Hình 4.6 mô tả dung lượng bộ nhớ cần cấp cho các tập tin trong thời gian chạy. Việc phân bổ tĩnh luôn chiếm một dung lượng cố định là 3.4 kB trong suốt quá trình thực hiện, trong khi việc sử dụng giao thức ngăn xếp chỉ mất một dung lượng rất nhỏ, dao động 0-328 byte cho toàn bộ hệ thống.

3 Bảng 4.3. Sự phức tạp trong quá trình thực hiên.

|  |  |  |
| --- | --- | --- |
| **Component** | **ROM (bytes)** | **SLOC** |
| Database Kernel  Parsing  LogicVM  MaxHeap index  Storage abstraction  Index abstraction  Query execution  Lexical analysis  Inline index  Hash index | 3570  3310  2798  2044  1542  1118  1006  790  620  512 | 908  641  776  556  358  304  367  183  145  119 |
| **Sum** | **17302** | **4357** |

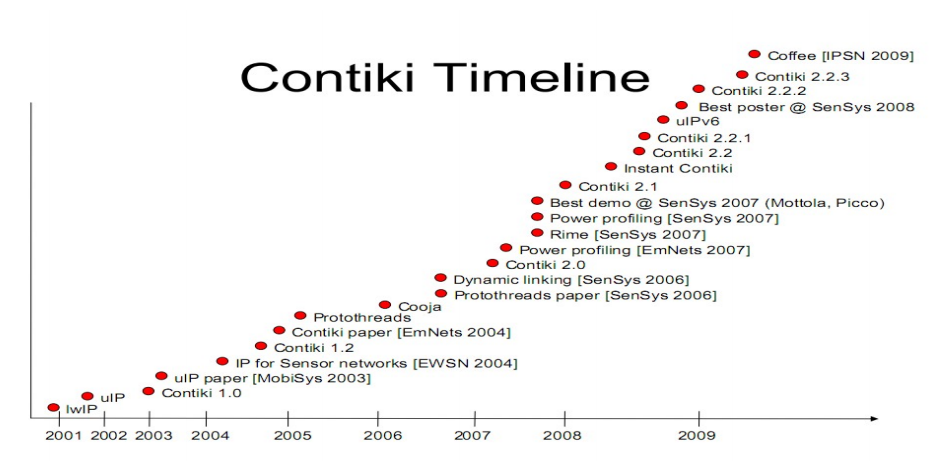


12Hình 4.6 Phân bổ bộ nhớ cho các tập tin trong Antelope

# CHƯƠNG 5: HỆ ĐIỀU HÀNH CONTIKI

## 5.1 Giới thiệu về hệ điều hành Contiki

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  các đối tượng thông minh. Contiki bắt đầu được nghiên cứu từ năm 2001 và phát hành phiên bản đầu tiên Contiki 1.0 năm 2003. Hình 5.1 cho thấy lịch sử phát triển của Contiki trong những năm qua. Phiên bản hiện nay của Contiki là 3.0, với nhiều thay đổi, bổ sung và phát triển vượt bậc. 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 đối tượng thông minh nói chung vá mạng cảm biến không dây nói riêng.



13Hình 5.1 .Lịch sử phát triển Contiki

Hệ điều hành Contiki được lập trình bằng ngôn ngữ C và có những đặc điểm phù hợp với các đối tượng thông minh:

* Contiki được chia thành nhiều module hoạt động độc lập. Nhờ đó các ứng dụng có thể sử dụng các module một cách linh động và chỉ nạp những module 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 thông qua uIP stack được xây dựng dựa trên nền TCP/IP.
* Có những module cho phép ước lượng và quản lý năng lượng một cách hiệu quả.
* Các giao thức tương tác giữa các lớp và các nút 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.

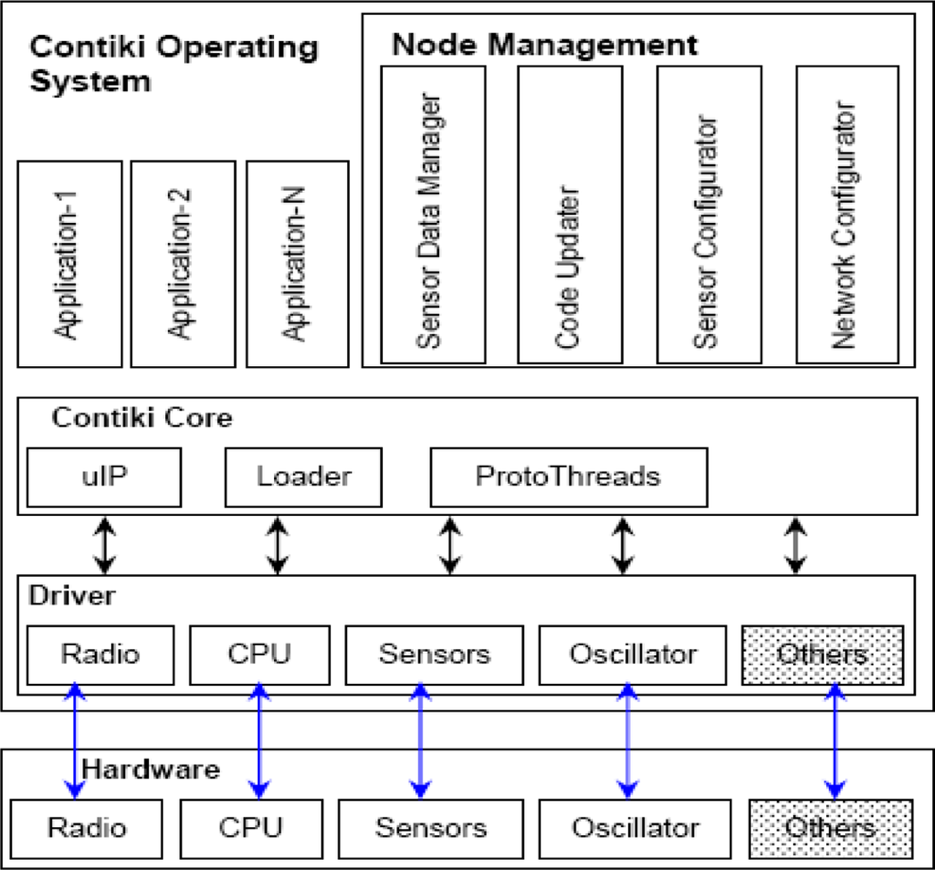
## 5.2 Cấu trúc hệ điều hành Contiki :

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

* **Thư mục 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 các đối tượng thông minh. Ứng dụng tiêu biểu trong thư mục này là trình duyệt web, máy chủ Web, FTP, email....
* **Thư mục Core**: Như tên gọi cho thấy, nó chứa các hạt nhân của hệ điều hành Contiki. Nó chứa khoảng 300 files, 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.
* **Thư mục  CPU:** Chứa các bộ xử lý cụ thể cho việc thực hiện các chức năng khác nhau được sử dụng trong hệ điều hành.
* **Thư mục  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.
* **Thư mục  Examples:**  Chứa  các  chương  trình ví dụ đơn  giản  bắt đầu với “Hello-world”, như là bước đầu tiên hướng tới lập trình ứng dụng trên Contiki.
* **Thư mục  Platform:**  Bao gồm thông tin cụ thể liên quan đến nền tảng phần cứng cho các nút cảm biến như ESB, Sky mote,…
* **Thư mục Tools:**  Là thư mục chứa các công cụ phần mềm đặc biệt. Ví dụ như 'Cooja' là một chương trình Java để mô phỏng cho Contiki.Thư mục này cũng chứa các công cụ cho các nền tảng phần cứng cụ thể. Ví dụ điển hình là các công cụ cho nút cảm biến Tmote Sky của Sentilla.

## 5.3 Kiến trúc phân lớp hệ điều hành Contiki

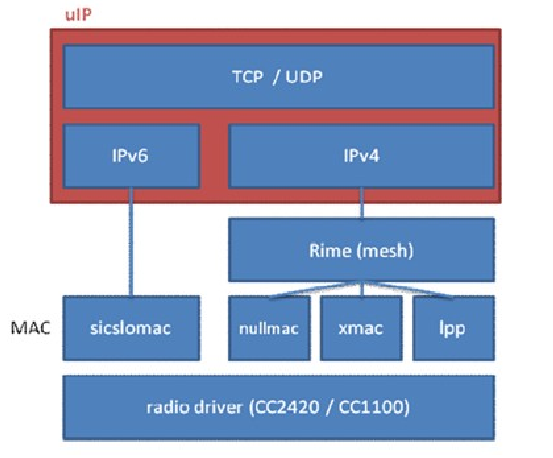
Hệ điều hành Contiki theo kiểu kiến trúc module. Tại mức hạt nhân nó theo mô hình điều khiển hướng sự kiện (event-driven), nhưng lại cung cấp các phương tiện tùy chọn luồng tới các tiến trình riêng lẻ. Nhân Contiki bao gồm một bộ lập lịch sự kiện làm nhiệm vụ gửi đi các sự kiện tới các tiến trình đang chạy. Các tiến trình thực thi được kích hoạt bằng các sự kiện gửi đi bởi hạt nhân tới các tiến trình hoặc bằng cơ chế hỏi vòng. Cơ chế hỏi vòng được sử dụng để tránh các điều kiện tranh đua (race conditions). Bất kì sự kiện nào đã được lập lịch sẽ chạy cho đến khi nó hoàn thành. Tuy nhiên, các trình xử lý sự kiện cũng có thể sử dụng các cơ chế nội để ưu tiên. Có hai loại sự kiện được hỗ trợ bởi hệ điều hành Contiki: Các sự kiện đồng bộ và không đồng bộ. Sự khác nhau giữa hai loại sự kiện này đó là, sự kiện loại đồng bộ được gửi đi ngay lập tức tới tiến trình đích bởi vì nó đã được lập lịch. Còn đối với các sự kiện không đồng bộ thì chậm hơn, thủ tục gọi được xếp vào hàng đợi và sau đó cũng được gửi đến tiến trình đích. Cơ chế hỏi vòng được sử dụng trong Contiki có thể xem như là các sự kiện có ưu tiên cao nó đã được lập lịch giữa mỗi sự kiện không đồng bộ. Khi một hỏi vòng đã được lập lịch thì tất cả các tiến trình đó thực hiện một trình xử lý hỏi vòng được gọi là thứ tự ưu tiên của chúng. Tất cả các công cụ của hệ điều hành ví dụ như, bộ cảm biến xử lý dữ liệu, truyền thông, và các trình điều khiển thiết bị được cung cấp dưới dạng các dịch vụ. Mỗi dịch vụ có một giao diện và cài đặt của nó. Các ứng dụng sử dụng một dịch vụ cụ thể cần hiểu được giao diện dịch vụ. Một ứng dụng không quan tâm đến cài đặt của một dịch vụ. Hình 5.2 minh họa kiến trúc phân lớp của hệ điều hành Contiki.



14Hình 5.2: Kiến trúc hệ điều hành Contiki

## 5.4 Ngăn xếp truyền thông

Contiki cơ bản gồm 2 stack truyền thông là uIP với TCP/UDP, IPV4, IPV6 giúp hệ điều hành truyền thông qua mạng Internet và Rime được thiết kế cho những liên kết không dây năng lượng thấp, nó cung cấp một phạm vi rộng lớn các truyền thông nguyên thủy từ những cách thức quảng bá nội vùng hiệu quả cao đến flooding dữ liệu đáng tin cậy trên nhiều nút mạng.



15Hình 5.3: Kiến trúc giao thức mạng trong hệ điều hành Contiki

### Ngăn xếp uIP

Trong những năm gần đây, cùng với sự thành công của Internet, giao thức  
TCP/IP đã trở thành tiêu chuẩn toàn cầu trong lĩnh vực truyền thông, TCP/IP là  
giao thức cơ bản được sử dụng cho những mục đích truyền tải các trang web, gửi  
và nhận email, truyền dữ liệu…Các hệ thống nhúng sử dụng TCP/IP có khả năng  
kết nối những hệ thống trực tiếp đến một mạng nội bộ, hoặc thậm chí là một mạng  
toàn cầu. Những thiết bị nhúng có khả năng đáp ứng được đầy đủ những đặc tính  
củaTCP/IP sẽ là những thiết bị có tính ưu việt, có khả năng giao tiếp một cách đầy  
đủ với tất cả các thiết bị khác trong mạng. Tuy nhiên, sự triển khai giao thức TCP/IP truyền thống đòi hỏi quá nhiều tài nguyên gồm cả dung lượng code và bộ nhớ sử dụng, không thể được đáp ứng trong các hệ thống nhúng 8 hoặc 16 bit.

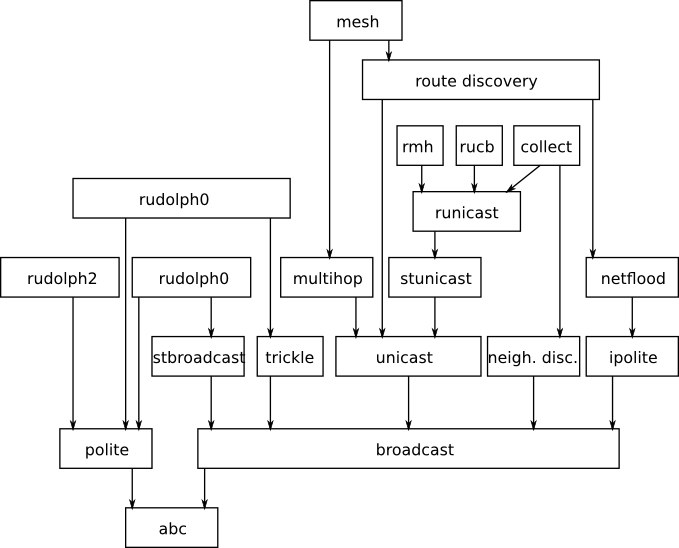
Xuất phát từ ý tưởng đó, uIP được thiết kế dựa trên ngôn ngữ C với mục  
tiêu tối ưu hóa tuyệt đối các đặc tính cần thiết cho một stack TCP/IP đầy đủ. uIP  
chỉ có thể hoạt động với một giao diện mạng duy nhất và bao gồm các giao thức :  
IP, ICMP, UDP, TCP.

**Một số hàm trong API – uIP:**

* Khởi tạo một kết nối : tcp\_connect(), tcp\_listen(), udp\_new().
* Ngắt một kết nối : uip\_close(), uip\_abort(). 23
* Gửi một gói kiểu TCP: uip\_send().
* Gửi một gói kiểu UDP : uip\_udp\_packet\_send().
* Nhận một gói từ mạng : uip\_newdata(), uip\_datalen().
* Gửi lại một gói dữ liệu : uip\_rexmit().
* Mở và Kiểm tra cổng kết nối : uip\_listen() – mở một cổng kết nối ;
* uip\_connected()- kiểm tra kết nối.

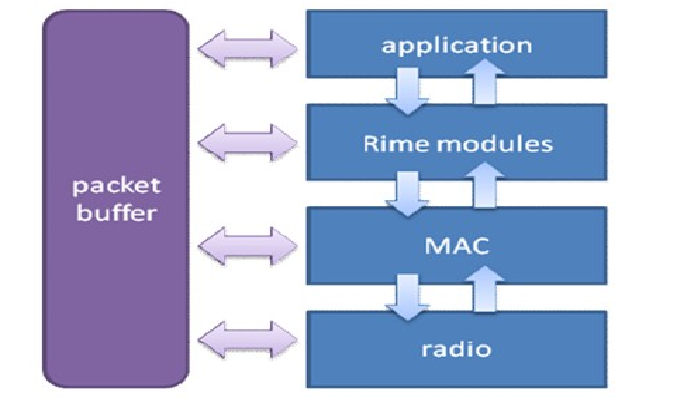
### Ngăn xếp Rime

Rime stack cung cấp một cấu trúc phân tầng của giao thức mạng cảm biến  
không dây, từ một bộ phát quảng bá đơn giản tới việc định tuyến rắc rối trong toàn mạng. Rime triển khai một giao thức phức tạp, với nhiều phần, mỗi phần lại gồm những module phức tạp được tạo nên từ những module nhỏ lẻ đơngiản hơn. Dưới đây là toàn thể tổ chức của giao thức Rime



16Hình 5.4: Tổ chức của Rime

* **Abc**: phát sóng quảng bá, nó chỉ gửi một gói tin qua các trình điều khiển vô tuyến và nhận tất cả các gói tin từ các trình điều khiển vô tuyến khác.
* **Broadcast:** phát sóng xác định, nó thêm địa chỉ người gửi để gửi đi các gói dữ liệu và chuyển nó vào module abc.
* **Unicast**: module này cho biết thêm một địa chỉ đích cho các gói tin được truyền cho khối phát sóng. Ở bên nhận, nếu địa chỉ đích của gói tin không phù hợp với địa chỉ của nút thì gói tin đó sẽ bị loại bỏ.
* **Stunicast:** là các unicast “cứng đầu “, khi được hỏi để gửi một gói tin đến mộtnút, nó sẽ gửi nhiều lần với một khoảng thời gian nhất định cho đến khi yêu cầu dừng lại.
* **Runicast:** là các unicast đáng tin cậy, nó sẽ gửi một gói tin bằng cách sử dụng các stunicast chờ một gói tin xác nhận. Khi nhận được, nó dừng việc truyền tải liên tục của các gói tin. Một số lượng tối đa các gói tin truyền lại phải được xác định, để tránh gửi vô hạn.
* **Polite và ipolite:** hai module gần như giống hệt nhau, khi một gói tin đã được gửi đi trong một khung thời gian nhất định, module chờ một nửa thời gian, kiểm tra xem nó có nhận được gói tin nó định gửi hay không. Nếu trùng, gói tin không được gửi đi, nếu không nó sẽ gửi gói tin. Điều này rất hữu ích cho các kỹ thuật flooding để tránh việc truyền lại không cần thiết.
* **Multihop:** module này đòi hỏi chức năng bảng định tuyến, và khi định gửi một gói tin, nó yêu cầu bảng định tuyến cho hop tiếp theo và gửi gói tin đến nó bằng cách unicast. Khi nó nhận được một gói tin, nếu hop đó là đích, gói tin sẽ được truyền tới các lớp trên, nếu không nó sẽ yêu cầu thông tin về hop tiếp theo từ bảng định tuyến và chuyển tiếp các gói tin đến nó. Khi gửi gói, các ứng dụng lưu gói vào bộ nhớ đệm và gọi các hàm xử lý liên quan để gửi gói đi. Khi nhận được một gói, gói nhận được được lưu trong bộ đệm gói, đồng thời Rime stack gọi các hàm “callback” tương ứng để xử lý gói đầu vào.



17Hình 5.5: Quá trình truyền dữ liệu của Rime

## Kỹ thuật lập trình

Contiki hoạt động dựa trên cơ chế điều khiển sự kiện. Các Process hay các trình xử lý được gọi mỗi khi có một sự kiện xảy ra như các sự kiện về cảm biến, trình khởi động, trình kết thúc… Quá trình gọi các Process hoàn toàn không bị ngăn chặn hay cản trở trong các chương trình. Process hoạt động dựa trên các Protothread, tạo ra các luồng điều khiển liên tiếp theo cơ chế event -driven. Protothread là sự kết hợp giữa hai cơ chế điều khiển: “Multithreads” – Lập trình đa luồng và “event – driven” – Lập trình hướng sự kiện.

**Lập trình Event-driven**

Lập trình hướng sự kiện (Event-driven) là một phương pháp hiệu quả về bộ nhớ để viết phần mềm cho các nút cảm biến. Với kiểu lập trình này, phần mềm được biểu diễn như là các trình xử lý sự kiện: Các đoạn mã ngắn gọn mô tả làm thế nào mà hệ thống đáp ứng được các sự kiện. Ví dụ về các sự kiện như là một gói tin vô tuyến đến từ một nút lân cận, sự kiện đọc dữ liệu cảm biến từ một trongcác bộ cảm biến, và sự kiện từ bộ định thời. Khi sự kiện diễn ra, nút cảm biến trả lời bằng cách thực thi một phần phần mềm của nó. Lập trình hướng sự kiện đòi hỏi ít bộ nhớ hơn so với lập trình đa luồng bởi vì không có luồng nào yêu cầu bộ nhớ ngăn xếp. Toàn bộ hệ thống có thể chạy như một luồng duy nhất, trong đó chỉ yêu cầu một ngăn xếp duy nhất. Kiểu lập trình hướng sự kiện cũng phù hợp tự nhiên với bản chất hướng sự kiện của các nút cảm biến. Vì nút cảm biến tương tác với một môi trường điều khiển hướng sự kiện, nên mô hình lập trình nắm bắt được hành vi có thể quan sát được của hệ thống.

**Lập trình Multi-threaded**

Đa luồng là một kỹ thuật lập trình cho phép nhiều chương trình có thể chạy đồng thời trên một bộ xử lý duy nhất. Trong lập trình đa luồng, mỗi chương trình 26 được định sẵn một luồng điều khiển riêng của nó, và luồng đó chạy cùng với tất cả các luồng khác trong hệ thống. Mỗi luồng được định sẵn thời gian nhất định để chạy trên bộ vi xử lý. Để cho phép chạy nhiều chương trình cùng một lúc, hệ điều hành chuyển đổi giữa các luồng để chúng cùng nhau nhận được sự chia sẻ công bằng của bộ vi xử lý.Lập trình đa luồng được sử dụng rộng rãi trong các hệ thống điều hành thông dụng, ở đó các luồng tự bảo vệ lẫn nhau sao cho một luồng không thể tiếp cận một luồng khác mà không đi qua các giao diện đã quy định. Khi các luồng tự bảo vệ lẫn nhau, chúng thường được gọi là các tiến trình thay vì gọi là các luồng. Đối với các đối tượng thông minh, tồn tại một vấn đề đa luồng là mỗi luồng yêu cầu một phần bộ nhớ của riêng mình để giữ trạng thái của các luồng này, được gọi là ngăn xếp của luồng. Các ngăn xếp chứa các biến cục bộ mà luồng sử dụng và các giá trị trả về cho các hàm mà luồng gọi đến, nhưng cũng bao gồm một lượng tương đối lớn bộ nhớ không sử dụng. Bộ nhớ này phải được cấp phát bởi vì nó chưa biết trước được có bao nhiêu bộ nhớ ngăn xếp mà mỗi luồng cần dùng. Do đó, bộ nhớ ngăn xếp thường vượt quá sự cấp phát.

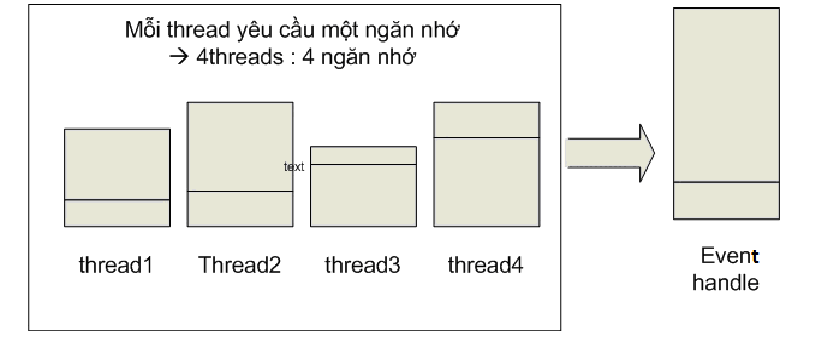
**Lập trình Protheads**

Protothreads là một cách để kết hợp ưu điểm của các mô hình lập trình hướng sự kiện và đa luồng. Protothreads là cơ chế lập trình được phát triển cho các hệ thống có bộ nhớ hạn chế, nó kết hợp mô hình lập trình hướng sự kiện và đa luồng theo một phương thức hiệu quả về bộ nhớ. Với Protothreads, chương trình được cấu trúc theo tuần tự, giống như trong mô hình đa luồng, nhưng sử dụng ít bộ nhớ tương tự như mô hình hướng sự kiện. Protothreads có thể thực hiện được hiệu quả trong ngôn ngữ lập trình C mà không cần bất kỳ ngôn ngữ lập trình bậc thấp hay các thay đổi nào với trình biên dịch. Điều hạn chế là các lập trình viên  
phải lưu trữ các biến một cách rõ ràng khi các Protothreads dừng. Bởi vì các Protothreads được thực hiện bởi ngôn ngữ C, nên chúng rất tiện lợi trên các nền tảng phần cứng khác nhau.

**So sánh ba mô hình lập trình trong Contiki**

Mô hình lập trình Multi-threaded có khả năng thực hiện đồng thời một chuỗi các threads. Tuy nhiên, các threads đòi hỏi phải được thực hiện trên những ngăn nhớ riêng, tạo ra chuỗi các luồng điều khiển tuần tự. Multi-threaded được xây dựng như một thư viện, cung cấp cho các Process khi cần thiết.

Trong khi đó, mô hình lập trình Event-driven chỉ hoạt động trên một ngăn nhớ và thực hiện các luồng điều khiển tùy theo các sự kiện đến. Do đó Event-driven đòi hỏi bộ nhớ ít hơn và cung cấp cơ chế điều khiển linh hoạt theo các sự kiện.

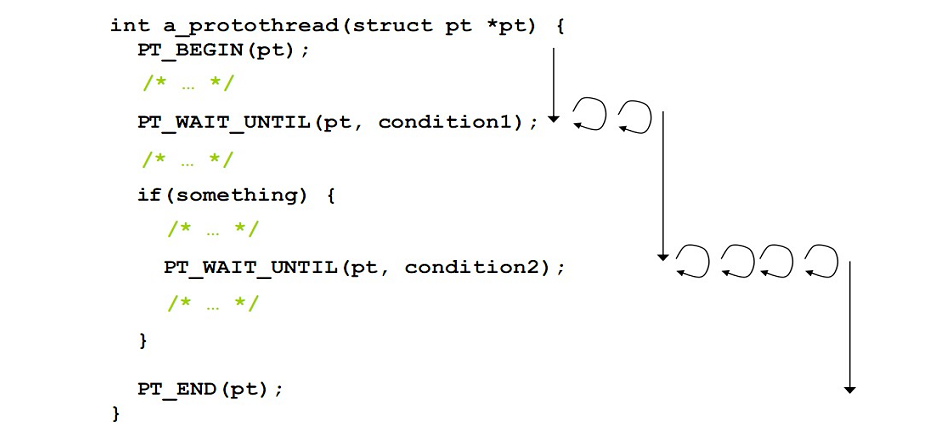
**

18Hình 5.6: Phương thức sử dụng bộ nhớ của Multi-threaded và Event-driven

Nhờ sự kết hợp các đặc tính của hai cơ chế Multi-threaded và Event-driven, Protothreads có khả năng cung cấp cơ chế điều khiển kiểu event-driven, cung cấp các luồng điều khiển liên tục, đồng thời sử dụng dung lượng bộ nhớ nhỏ với một ngăn nhớ duy nhất.

Trong API của Contiki bao gồm 4 loại Protothreads cơ bản:

* **PT\_INIT (pt)**: khởi tạo một Protothread.
* **PT\_BEGIN (pt)**: bắt đầu một Protothread.
* **PT\_WAIT\_UNTIL (pt, điều kiện)**: điều khiển đợi một sự kiện
* **PT\_END (pt)**: kết thúc một Protothread.



19Hình 5.7: Ví dụ về lập trình Protothreads

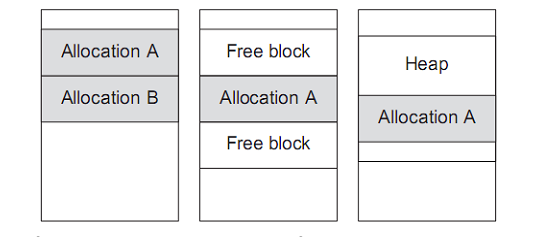
## Hoạt động định thời

Trong Contiki sử dụng 4 loại định thời:

* Timer: là loại định thời thụ động, chỉ sử dụng để lưu lại vết các thời điểm khi bộ định thời hết hạn.
* Rtimer: là loại định thời thời gian thực, sử dụng để gọi một hàm tại một  
  thời điểm cụ thể nào đó.
* Event timer (etimer): là loại định thời hoạt động, được kích hoạt trong các Process và sử dụng để gửi một sự kiện đến Process khi bộ định thời hết hạn.
* Callback timer (ctimer): là loại định thời hoạt động, có thể được sử dụng ở bất kỳ vị trí nào trong chương trình, có chức năng gọi một hàm xử lý mỗi khi bộ định thời hết hạn. Ctimer được sử dụng trong module RIME của Contiki.

## Quản lí bộ nhớ

Do những hạn chế nghiêm trọng như năng lượng tiêu thụ, kích thước vật lý và chi phí của các bộ vi điều khiển được sử dụng cho các nút cảm biến, nên bộ nhớ bị hạn chế. Do đó, bộ nhớ phải được quản lý một cách hiệu quả. Có một số kỹ thuật được thực hiện cho hầu hết các bộ nhớ hạn chế trong một nút cảm biến. Không giống như các máy tính thông thường, mà ở đó bộ nhớ có thể hoán đổi một cách linh động vào một ổ đĩa cứng, thì bộ nhớ trong vi điều khiển của các đối tượng thông minh thường không thể được di chuyển tới bộ lưu trữ thứ hai.  
Ở phần mềm, bộ nhớ được cấp phát có thể là tĩnh tại thời điểm biên dịch hoặc cấp phát động ở thời điểm chạy. Bộ nhớ được cấp phát tĩnh cho phép các lập trình viên biết trước là chương trình sẽ phù hợp với bộ nhớ của vi điều khiển, nhưng nó không cho phép hệ thống đáp ứng một cách linh hoạt với các yêu cầu trong thời gian chạy. Mặt khác, sự cấp phát bộ nhớ động có thể đáp ứng được bộ nhớ mà hệ thống yêu cầu thực tế, nhưng nó không thể dự đoán được hành vi của hệ thống sẽ như thế nào.

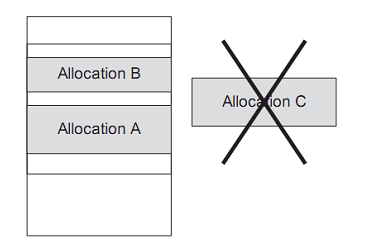


20 Hình 5.12: Cấp phát bộ nhớ tĩnh (trái), sự cấp phát động từ một vùng bộ nhớ tĩnh  
(giữa) và cấp phát động từ một Heap (phải).

Bởi vì những ưu điểm và những hạn chế khác nhau của các phương pháp cấp phát động và tĩnh, nên một phương pháp lai thường được sử dụng. Trong phần này, chúng ta hãy xem ba phương pháp:

* Cấp phát tĩnh: Tất cả bộ nhớ được cấp phát tại thời điểm biên dịch và không bộ nhớ nào được cấp phát trong thời gian chạy.
* Cấp phát động từ vùng bộ nhớ tĩnh: Bộ nhớ có thể được cấp phát động  
  trong thời gian chạy từ một tập hợp cố định của các vùng bộ nhớ tĩnh. Kích thước của mỗi cấp phát được xác định trước và không thể thay đổi trong thời gian chạy.
* Cấp phát động từ một vùng lưu trữ đặc biệt Heap: Bộ nhớ có thể được cấp phát động trong thời gian chạy và kích thước của mỗi cấp phát có thể được xác định tại thời gian chạy.

Hình 5.12 cho thấy bộ nhớ được cấp phát với cấp phát tĩnh, cấp phát động từ một vùng bộ nhớ tĩnh và cấp phát động từ một vùng lưu trữ đặc biệt Heap. Hình này cho thấy các vùng nhớ A và B được cấp phát với ba phương pháp khác nhau. Với cấp phát tĩnh, hai cấp phát xuất hiện trong bộ nhớ từ khi hệ thống được khởi động đến khi hệ thống được tắt. Bộ nhớ được dành riêng cho hai cấp phát và không thể được sử dụng cho bất cứ chương trình nào khác.

Với cấp phát động, thì bộ nhớ cho các cấp phát không được dành riêng cho chúng, mà vùng nhớ đó cũng có thể được sử dụng bởi các cấp phát khác. Khi bộ nhớ được cấp phát động từ một vùng bộ nhớ tĩnh, thì các vùng bộ nhớ đã được cấp phát tĩnh. Các cấp phát tĩnh này sau đó sẽ được chia thành các phân đoạn cố định về kích thước. Bộ nhớ có thể được cấp phát từ các phân đoạn có kích thước cố định này. Sau khi một phân đoạn đã được cấp phát, nó chỉ có thể được sử dụng bởi chương trình đã được cấp phát nó. Khi chương trình thực hiện xong với phân đoạn, chương trình sẽ trả lại phân đoạn đến vùng bộ nhớ. Bộ cấp phát bộ nhớ đánh dấu sự phân đoạn là nhàn rỗi và có thể cấp nó tới một chương trình khác đề nghị  
lấy nó. Cấp phát động từ một vùng lưu trữ đặc biệt Heap phức tạp hơn cấp phát động từ một vùng nhớ. Với cấp phát động từ một vùng lưu trữ đặc biệt Heap, bộ nhớ được cấp phát là từ một phần của bộ nhớ được gọi là Heap. Bất kỳ kích thước nào của bộ nhớ có có thể được cấp phát từ Heap, miễn là có đủ Byte trống liên tiếp trên Heap. Mỗi lần một phần nào của Heap được cấp phát, phần này của bộ nhớ không thể được di chuyển hoặc được cấp phát bởi chương trình khác. Khi chương trình thực hiện xong với bộ nhớ của nó, nó trả lại bộ nhớ cho Heap

21 Hình 5.13: Vấn đề với cấp phát động Heap: cấp phát cho C không thể được

cấp phát, thậm chí nếu có đủ bộ nhớ trên Heap, bởi vì bộ nhớ đã bị phân mảnh.

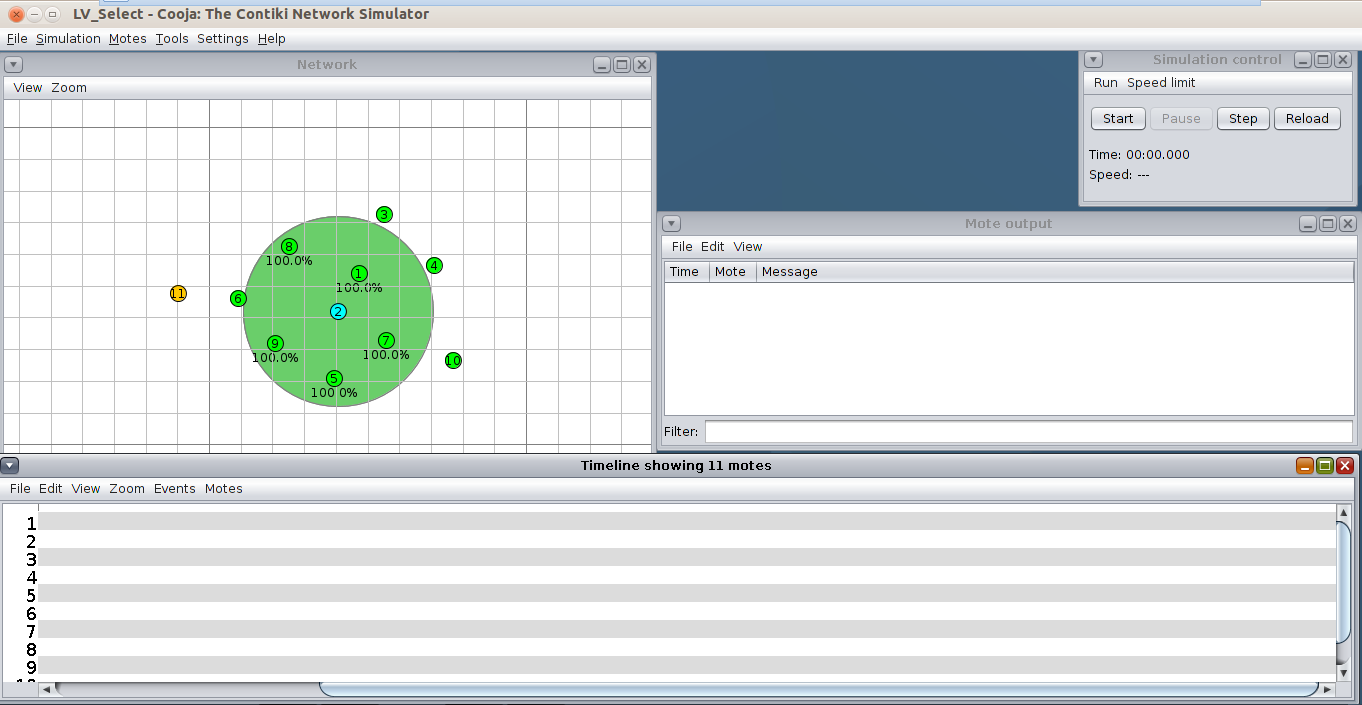
Lợi ích của việc cấp phát động Heap là các phân đoạn bộ nhớ có kích thước bất kỳ có thể được cấp phát. Cái giá cho cho ưu điểm này là Heap có thể bị phân mảnh do đó bộ nhớ không thể được cấp phát từ Heap, ngay cả khi có đủ số Byte nhàn rỗi còn lại. Điều này được minh họa trong hình 5.13, nơi cấp phát bộ nhớ cho C không thể được cấp phát bởi vì không có đủ Byte liên tiếp lại còn lại trên Heap. Thậm chí cả khi số lượng các Byte nhàn rỗi trên Heap lớn hơn kích thước cấp phát cho C, bộ nhớ có thể không được cấp phát do sự phân mảnh. Bởi vì các vấn đề về phân mảnh trong cấp phát Heap, nên hầu hết các nút cảm biến sử dụng cấp phát tĩnh cho hầu hết các mục đích và sử dụng việc cấp phát vùng nhớ động khi cấp phát bộ nhớ động là cần thiết. Bởi vì các nút cảm biến thường được thiết kế cho một nhiệm vụ duy nhất, nên cấp phát tĩnh thường là một chiến lược để cấp phát bộ nhớ. Nhưng bởi vì khối lượng công việc có thể thay đổi, nên cũng cần thiết một lượng nhất định các cấp phát động.

## 5.8 Phần mềm mô phỏng COOJA

**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:



22Hình 5.14: Giao diện mô phỏng Cooja.

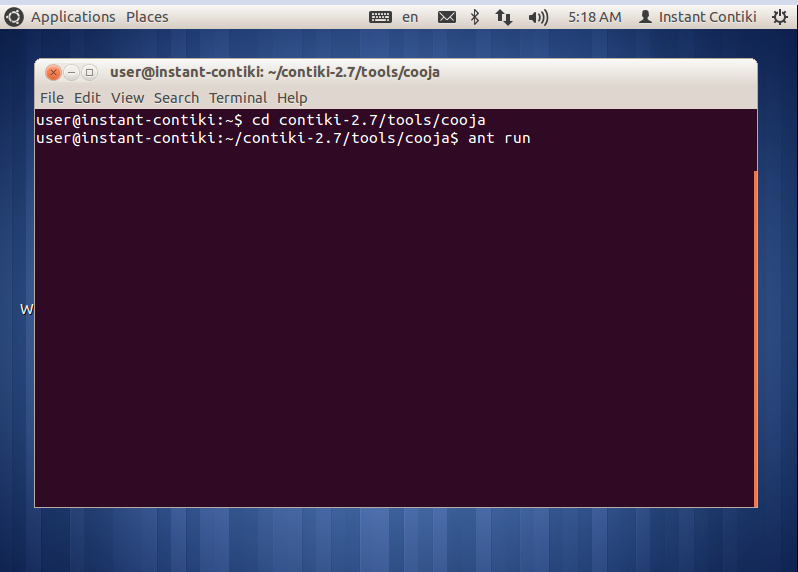
Từ hình có thể nhận thấy giao diện của chương trình thân thiện và dễ sử dụng, với một màn hình cho phép hiển thị các quá trình hoạt động của nút, có khả năng thay đổi vị trí, phạm vi phủ sóng của mỗi nút. Bên cạnh đó Cooja cung cấp một số các cửa sổ theo dõi sự kiện như Log listener, Radio listener cho phép người sử dụng tìm kiếm những sự kiện theo một số thông số nhất định, theo dõi sự giao tiếp giữa một số nút 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**.**

**Cài đặt :**

Click vào Terminal, cửa sổ hiện gõ lần lượt các dòng lệnh sau:

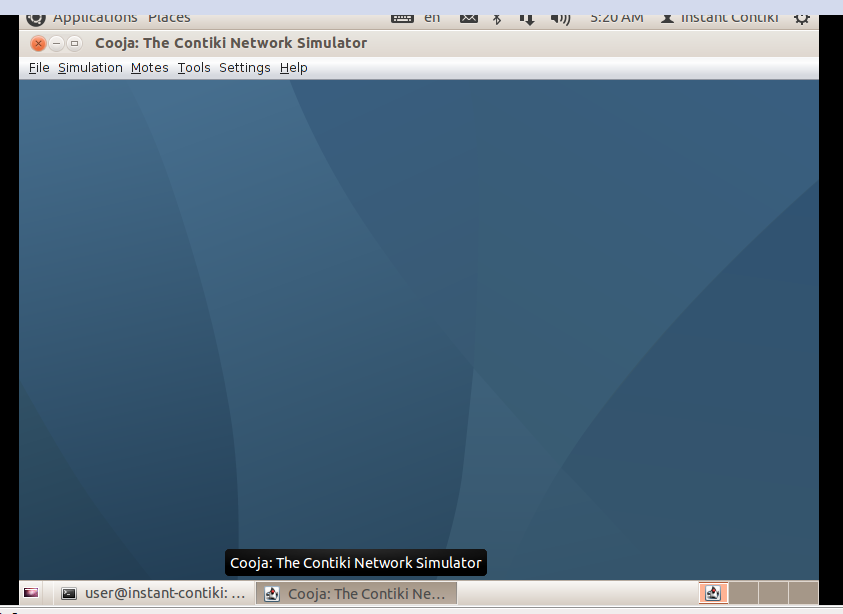
*cd contiki-2.7/tools/cooja*

*ant run*

**

23Hình 5.15: Cứa sổ terminal trong contiki

Cửa sổ cooja hiện ra :



24Hình 5.16: Màn hình mô phỏng Cooja

# CHƯƠNG 6: MÔ PHỎNG VÀ KẾT QUẢ

## Mô hình mô phỏng

Mô hình mô phỏng được xây dựng dựa trên các mục đích khảo sát, sẽ có hai loại nút được thiết lập xây dựng cùng với các chức năng tương ứng. Công cụ mô phỏng dùng để thực hiện trong chương này chính là Cooja, nhằm đánh giá và so sánh hiệu quả về công suất tiêu tốn, thời gian xử lí của các nút cảm biến trong những thuật toán đánh chỉ số khác nhau.

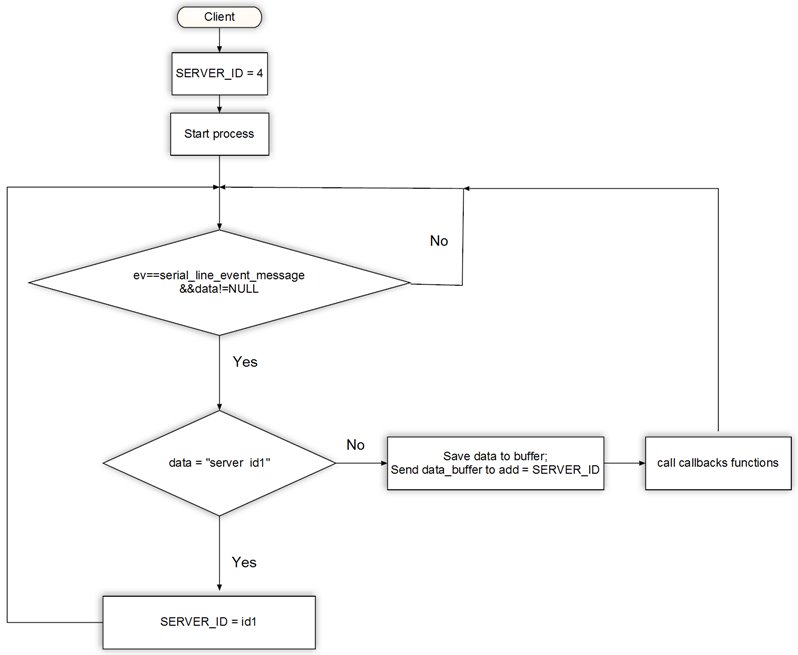
Có hai loại nút được đưa vào mô phỏng đó là:

* **Client** : là nút yêu cầu, mục đích chính gửi các truy vấn tới tới server và nhận phản hồi từ các nút server
* **Server**: xem như là một máy chủ cơ sở dữ liệu, có khả năng xây dựng cơ sở dữ liệu một cách tự động, có khả năng nhận và xử lí truy vấn từ client và gửi trả về kết quả.

Chương trình của client cung cấp một giao diện cổng nối tiếp, cùng với sự giúp đỡ của nó, chúng ta có thể nhập vào các truy vấn. Quy trình shell của chương trình client sẽ xử lí các gói tin này (các truy vấn ) và gửi các truy vấn tới chương trình server (đang chạy trên các nút khác). Mỗi khi nó nhận được một phản hồi từ truy vấn, nó sẽ in ra thông điệp xác nhận ra shell. Sơ đồ khối của client được biểu diễn trên hình 6.1.

SERVER\_ID là địa chỉ của server nơi nhận các truy vấn từ client. Ở đây nó được gán giá trị ban đầu bằng 4. Antelope sử dụng giao thức định tuyến cho mạng mesh Rime để trao đổi gói tin giữa client và server. Đầu tiên sẽ là quá trình thiết lập và tạo một “shell\_process” cung cấp một giao diện truy vấn tới client. Khi có một sự kiện xuất hiện ở trong giao diện cổng nối tiếp, client sẽ kiểm tra data được đưa vào, để điều chỉnh địa chỉ SERVER\_ID của server.

Sau khi quá trình so sánh được hoàn thành, toàn bộ dữ liệu sẽ được đưa vào một bộ nhớ đệm, và sẽ được gửi tới server thông qua một kết nối mesh.

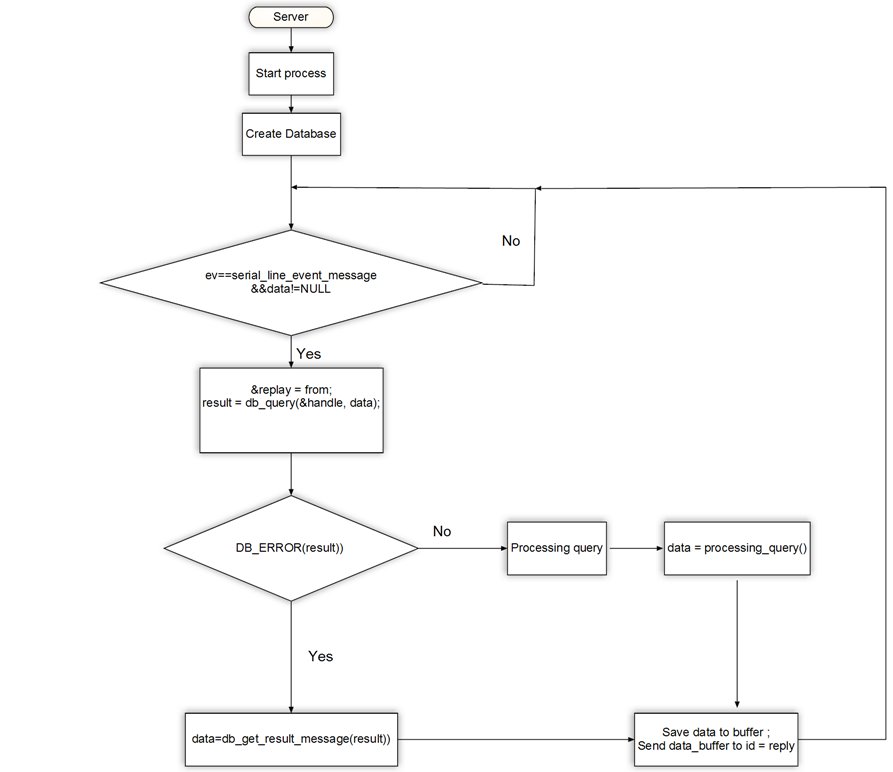


25Hình 6.1 : Sơ đồ giải thuật tại nút client

Trong quá trình gửi, client sẽ gọi hàm callbacks(). Ở đây nó sẽ tạo một cấu trúc callbacks, trong đó chứa các hàm con trỏ chỉ tới các hàm callback, được sử dụng dựa trên các gói tin nhận được. Một protothread được tạo ra cho netdb\_process. Để mở đầu và kết thúc một protothread, chúng ta phải sử dụng **PROCESS\_BEGIN()** và **PROCESS\_END()**. “mesh\_open” mở một kết nối mới và ở đây hệ thống kết nối mesh được tạo trên NETDB\_CHANNEL(70), những hàm sẽ được gọi khi nhận được gói tin trên kênh truyền.

* Hàm *callback – sent:* Hàm này sẽ được gọi khi chúng ta gửi một gói tin bằng mesh\_send() và nó đã được truyền.
* Hàm *callback – timeout*: Hàm này được gọi khi chúng ta cố tình truyền một gói tin bằng mesh\_send() và gói tin này bị hủy bỏ vì hết thời gian timeout. Ở đây chương trình trên client sẽ in ra "Failed to send the packet: timeout".
* Hàm *callback – received*: được gọi khi nhận được một gói tin trên kênh truyền, trong đó sẽ chứa các thuật toán xử lí dữ liệu. Ở trong xử lí của hàm “received”, một điều kiện kiểm tra lỗi được đưa ra (len > MAX\_QUERY\_SIZE) và in ra một lỗi cho những truy vấn quá dài nếu điều kiện trả về true. Nếu không có chuyện gì xảy ra cùng với bản tin trả về của truy vấn, bản tin trả về sẽ được in trên giao diện nối tiếp.

Các giải thuật cơ bản của server thực hiện tương đối giống với các giải thuật thực hiện tại client và đã được trình bày ở trên như cấu trúc dữ liệu mesh\_conn, các hàm callbacks và PROCESS, PROTO\_THREADS. Dưới đây là sơ đồ giải thuật của server.



26Hình 6.2: Sơ đồ giải thuật tại nút server

Đầu tiên quá sau khi khởi động trình xử lí server, nó sẽ tạo một cơ sở dữ liệu quan hệ, dùng để lưu trữ các giá trị, nhằm phục vụ các truy vấn nhận được từ client hay từ chính nó. Một protothread được tạo ra cho process shell khởi động. Ở trong một vòng lặp, nó sẽ đợi một sự kiện xảy ra. Có một lệnh lớn ở trong vòng lặp for là “PROCESS\_WAIT\_EVENT\_UNTIL(ev ==serial\_line\_event\_message && data != NULL);” sẽ làm cho thread chờ cho đến khi có sự kiện xảy ra ở cổng nối tiếp. Bản tin sự kiện trên cổng nối tiếp chính là dữ liệu mà người dùng đưa vào từ giao diện nối tiếp và nó không phải là “NULL”.

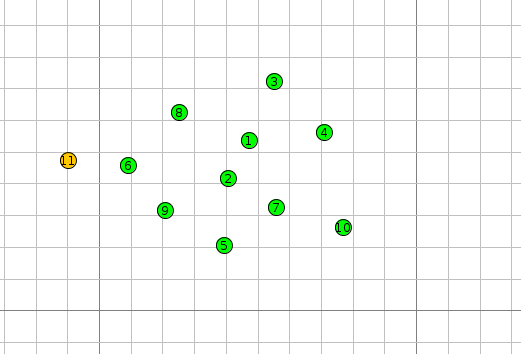
Ở đây nó sẽ kiểm tra chiều dài của dữ liệu với giá trị lớn nhất của bộ nhớ đệm, nếu không nó sẽ lưu địa chỉ của client vào địa chỉ “reply\_addr” làm địa chỉ trả về khi hoàn thành việc phân tích truy vấn bằng hàm received().

Khi nhận được dữ liệu từ cổng nối tiếp, tiếp theo là quá trình kiểm tra lỗi của truy vấn DB\_ERROR(result). Ở đây nó sẽ kiểm tra chiều dài của truy vấn đi vào với giá trị lớn nhất của bộ nhớ đệm, nếu không nó sẽ lưu địa chỉ của client vào địa chỉ “reply” làm địa chỉ trả về khi hoàn thành việc phân tích truy vấn bằng hàm received(). Khi hoàn thành quá trình kiểm tra, các truy vấn sẽ được thực thi, kết quả trả về được lưu vào bộ nhớ đệm và gửi trả về client có địa chỉ ID được lưu tại “reply”.

## Hoạt động lựa chọn

Hoạt động SELECT là loại truy vấn thường hay sử dụng nhất trong các ứng dụng thực tế của Antelope. Trong thí nghiệm này, chúng ta xem xét cách thực hiện SELECT bị ảnh hưởng như thế nào khi thay đổi số các yếu tố của tập hợp được lựa chọn từ các tuple, và khi chuyển đổi các thuật toán chỉ số.

Một loại của các truy vấn SELECT mà được sử dụng phổ biến trong cơ sở dữ liệu cảm biến là tìm kiếm tầm. Ví dụ, để tìm tất cả các tuple chèn giữa hai ngày, người dùng hoạt động SELECT với một điều kiện xác định phạm vi của các giá trị có thể chấp nhận ứng với *attribute* ngày. LogicVM phân tích điều kiện này và cung cấp phạm vi giá trị cho một chỉ số. Chỉ số này do hạt nhân cơ sở dữ liệu tư vấn dùng để phân định các bộ *tuple* đó và phải được quét để tạo ra kết quả.



27Hình 6.3 : Mô hình thực hiện truy vấn select, gồm 1 (ID = 11) client và 10 server

Hình 6.4 cho thấy thời gian hoàn thành các truy vấn với kích thước khác nhau về một *relation* bao gồm 20.000 *tuple*. Mỗi *tuple* gồm một *attribute* của miền INT (2 byte). Chúng ta đo thời gian hoàn thành cần thiết để tạo ra kết quả. Nếu lần đầu tiên chúng ta nhìn vào thời xử lí khi không sử dụng chỉ số trên các *attribute*, chúng ta thấy rằng thời gian hoàn thành hầu như là không đổi. Trong trường hợp này, tất cả các tuple phải được đánh giá theo tuần tự với điều kiện để lựa chọn, bất kể như thế nào một ít trong số chúng sẽ phù hợp với điều kiện.

28Hình 6.4. Hiệu suất của SELECT trong một RELATION với 20,000 TUPLES.

Khi sử dụng chỉ số Maxheap, ta có thể thấy rằng thời gian truy vấn sẽ bằng với trường hợp không dùng chỉ số khi chúng ta dùng select ở một tầm rộng, lí do của nó chính là việc các chỉ số trừu tượng sẽ thành không đánh chỉ số nếu như kết quả tìm kiếm nằm trong một tầm rộng trong *relation.*

## Thời gian thực hiện các hoạt động

Trong thí nghiệm này, chúng ta đánh giá dựa trên một tiêu chuẩnvề thời gian xử lý của các hoạt động khác nhau trong cơ sở dữ liệu. Mục tiêu của thí nghiệm này là để đánh giá hiệu suất và chi phí tối đa của mỗi hoạt động trong cơ sở dữ liệu.

Hình 6.5 cho thấy thời gian thực hiện đo bằng thang mili giây. Đối với hầu hết các hoạt động, thời gian thực hiện có thể dự đoán được. Các thời điểm chèn MaxHeap phụ thuộc vào độ sâu heap cấu hình sẵn. Quan sát các hoạt động bị ảnh hưởng bởi độ phức tạp của các hoạt động hệ thống tập tin nằm bên dưới. Khi sử dụng hệ thống tập tin Coffee đọc và ghi vào một tập tin được thực hiện trong thời gian O (1).

29Hình 6.5. Thời gian thực hiện của các hoạt động trong Antelope

Ở mô hình này chúng ta sẽ tính toán và ghi nhận thời gian hoàn thành việc tạo một cơ sở dữ liệu với thuật toán chỉ số *attribute* cũng như số lượng *tuple* khác nhau.

Hình 6.6 trình bày thời gian hoàn thành việc tạo một cơ sở dữ liệu với số lượng *tuple* và thuật toán đánh chỉ số khác nhau, bao gồm việc tạo *relation, attribute,chỉ số* và chèn *giá trị* vào các *attribute* đó . Có thể thấy rằng thời gian hoàn thành của việc tạo cơ sở dữ liệu có các *attribute* không đánh chỉ số và đánh chỉ số bằng thuật toán INLINE có thời gian hoàn thành gần như bằng nhau, và chỉ chênh lệch nhau do hoạt động tạo chỉ số. Trong khi đó, sử dụng thuật toán MAXHEAP trong việc đánh chỉ số cho *attribute* làm tăng thời gian xử lí của việc tạo cơ sở dữ liệu, chênh lệch chủ yếu phụ thuộc vào hoạt động chèn giá trị, số *tuple* càng lớn thì giá trị chênh lệnh về thời gian càng tăng.

30 Hình 6.6 : Thời gian hoàn thành hoạt động tạo cơ sở dữ liệu với những thuật toán chỉ số khác nhau và số lượng *tuple* khác nhau

## Năng lượng tiêu tốn trên mỗi hoạt động

31 Hình 6.7: Năng lượng tiêu tốn trên mỗi hoạt động

Ở đây, năng lượng tiêu tốn cho việc tạo các chỉ số Maxheaptốn nhiều bộ nhớ nhất, các hoạt động tạo các chỉ số của thuật toán Inline tốn ít chi phí hơn việc tạo chỉ số cho thuộc tính với thuật toán Maxheap. Tuy vậy việc chèn các phần tử của chỉ số Inline lại lớn hơn, do đó, việc tạo ra cơ sở dữ liệu sẽ tốn năng lượng nhiều hơn so với chỉ số Maxheap với số lượng *tuple*  phải chèn lớn.

.

## Năng lượng tiêu tốn

Để hiểu rõ năng lượng tiêu tốn của các truy vấn, và năng lượng tiêu tốn sẽ bị tác động như thế nào khi những yếu tố trong truy vấn được thay đổi. Chúng tôi sẽ tính toán giá trị năng lượng trung bình tiêu tốn trên mỗi *tuple* của hệ thống bao gồm ghi và đọc giá trị từ flash và năng lượng tiêu tốn cho việc nhận truy vấn và gửi về lại các bản tin phản hồi. Trong phần đánh giá này, mô hình sẽ được xây dựng tương tự như trong hình 6.3.

32Hình 6.8: Đồ thì biểu diễn năng lượng tiêu tốn trên mỗi tuple được truy vấn trong những trường hợp khác nhau về số tuple được truy vấn với những thuật toán chỉ số khác nhau.

Hình 6.8 trình bày năng lượng tiêu tốn trên một *tuple* khi thực hiện truy vấn *SELECT* với những thuật toán chỉ số khác nhau. Có thể thấy rằng năng lượng tiêu tốn trên một tuple khi không sử dụng thuật toán chỉ số tăng tuyến tính (theo thang logarit) khi số *tuple* được lựa chọn cho truy vấn giảm. Nguyên nhân là do khi truy vấn với một *attibute* không sử dụng thuật toán chỉ số thì *server* luôn thực hiện tìm kiếm trên tất cả các *tuple* trong *relation* do đó năng lượng xử lý các truy vấn hầu như không đổi nên khi số *tuple* được lựa chọn càng nhỏ thì năng lượng trên một *tuple* càng cao. Thuật toán chỉ số MaxHeap khi sử dụng trong một tìm kiếm ở tầm rộng thì *server* xử lý tương tự như trường hợp không có chỉ số. Do đó đồ thị năng lượng trên *tuple* của truy vấn với *attribute* sử dụng thuật toán chỉ số MaxHeap gần giống với không sử dụng thuật toán chỉ số khi thực hiện truy vấn trong một phạm vi rộng. Trong hai thuật toán chỉ số (InLine và MaxHeap) thì InLine thực hiện truy vấn hiệu quả hơn MaxHeap do năng lượng tiêu tốn trên một tuple khi sử dụng InLine nhỏ hơn (đặc biệt là khi thực hiện truy vấn trong một phạm vi rộng).

## Truy vấn từ xa

Trong phần này khảo sát thời gian đáp ứng của truy vấn được gửi qua nhiều hop (multi - hops).

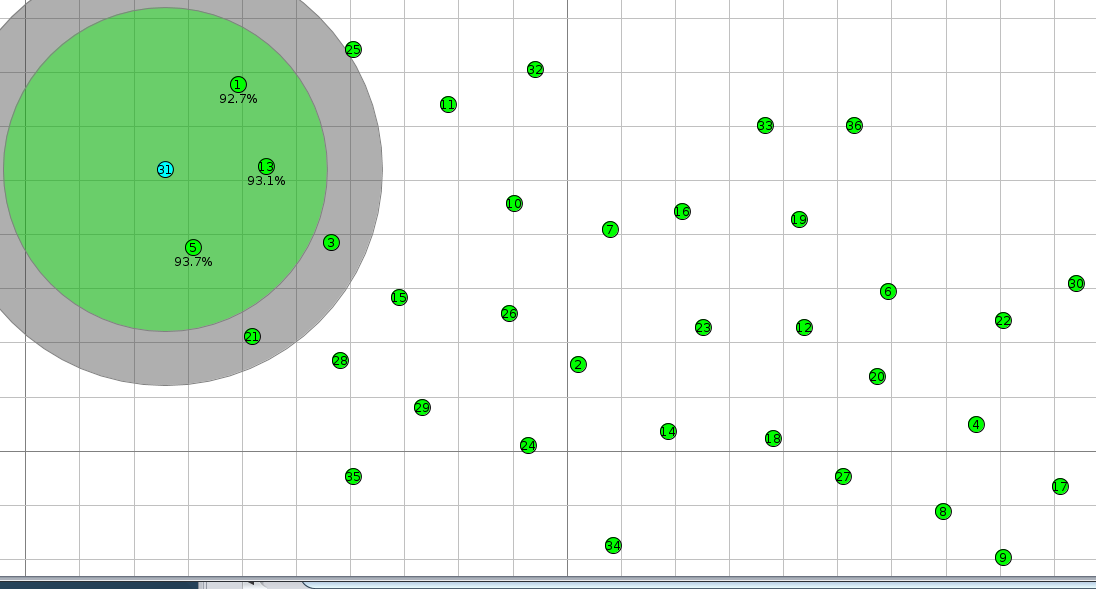
33Hình 6.9: Thời gian đáp ứng của truy vấn khi được thực hiện qua nhiều hop.

Hình 6.9 thể hiện thời gian đáp ứng bao gồm thời gian gửi truy vấn từ nút *client* đến nút *server*, thời gian thực hiện truy vấn tại server và thời gian nhận gói tin phản hồi từ *server* của *client*. Sự khác biệt về thời gian chỉ chịu tác động từ thời gửi truy vấn và thời gian nhận gói tin phản hồi. Chênh lệch này là không đáng kể đối với truy vấn trong phạm vi lớn hoặc truy vấn các *attribute* không sử dụng thuật toán chỉ số, nguyên nhân là do thời gian xử lý truy vấn lớn hơn rất nhiều thời gian gửi truy vấn và nhận gói tin phản hồi. Sự khác biệt chỉ thể hiện rõ ràng hơn khi thực hiện truy vấn trong phạm vi nhỏ và sử dụng các thuật toán chỉ số cho *attribute*. Hình 6.10 biểu diễn sự chênh lệch thời gian đáp ứng của các truy vấn trong phạm vi nhỏ và *attribute* sử dụng thuật toán chỉ số InLine.

34Hình 6.10: Thời gian đáp ứng của truy vấn khi được thực hiện qua nhiều hop sử dụng thuật toán InLine.

## Hiệu suất truy vấn

Gửi 50 truy vấn từ nút client (ID=31) đến tất cả các nút server trong mạng, mô hình mô phỏng trong phần này được thể hiện trong hình 6.11. Sau đó xác định số hop trong gói tin trả về và đếm số truy vấn thành công (nút client nhận được gói tin phản hồi). Lấy trung bình tỷ lệ thành công của các server cùng số hop với các trường hợp tỷ lệ thu phát khác nhau ta được hiệu suất của truy vấn ứng với số hop và tỷ lệ thu phát tương ứng.



35Hình 6.11: Mô hình mô phỏng hiệu suất truy vấn

Hình 6.12 biểu diễn hiệu suất của truy vấn theo số hop trong các trường hợp tỷ lệ thu – phát 100% (môi trường lý tưởng), tỷ lệ thu – phát 95% và tỷ lệ thu – phát 90%. Nhìn chung khi số hop càng cao hiệu suất truy vấn càng giảm chủ yếu là do quá trình định tuyến đến nút server thất bại (vượt quá thời gian timeout) nên làm mất một vài gói tin đầu tiên. Điều này được thể hiện trong đồ thị tỷ lệ truy vấn thành công theo số hop khi tỷ lệ thu – phát 100%, tỷ lệ thành công rất cao dù truy vấn đi qua nhiều hop. Nguyên nhân là do khi đã định tuyến được đường đi cho gói tin thì truy vấn sẻ luôn được thực hiện thành công. Khi tỷ lệ thu – phát dưới 100% ngoài việc mất gói tin do quá trình định tuyến còn xảy ra mất gói tin khi truyền trong môi trường. Vậy nên tỷ lệ thành công sẻ giảm xuống, tỷ lệ thu – phát càng thấp thì hiệu suất càng giảm.

36Hình 6.12: Tỷ lệ truy vấn thành công theo số hops và tỷ lệ thu – phát tương ứng

# CHƯƠNG 7: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Luận văn đã trình bày về Antelope, hệ thống quản lí cơ sở dữ liệu đầu tiên cho các thiết bị cảm biến có nguồn lực hạn chế. Antelope cho phép một lớp mạng cảm biến mà trên nó mỗi cảm biến có thể chưa một cơ sở dữ liệu. Mỗi nút cảm biến có thể lưu bất kì loại dữ liệu nào bao gồm dữ liệu cảm biến, thời gian chạy dữ liệu, chất lượng dữ liệu, trên cơ sỡ dữ liệu của nó. Cơ sở dữ liệu có thể được truy vấn điều khiển thông qua những nút khác hoặc bằng những truy vấn trên chính nó thông qua một chu trình được chạy trên mỗi nút. Chúng ta có thể thấy rằng việc tập hợp những truy vấn có thể giảm bớt được năng lượng tiêu tốn trên trong quá trình truyền tin và liên lạc so với các mạng thu thập dữ liệu truyền thống.

Thêm nữa, chúng tôi tin rằng công nghệ cơ sữ dữ liệu sẽ trở thành một trong những yếu tố quan trọng trong sự phát triển của các ứng dụng của mạng cảm biến. Hiệu suất lưu trữ, đánh chỉ số, quá trình truy vấn có vai trò quan trọng trong các ứng dụng lưu trữ trung tâm mới nổi. Antelope giải quyết những mối quan tâm này bằng cung cấu một cấu trúc cơ sở dữ liệu gồm một thiết bị máy ảo nhỏ gọn, dựa vào ngăn xếp, một hạt nhân cơ sở dữ liệu lặp lại thích hợp cho các hệ thống nhúng có nguồn lực hạn chế và một nhóm các thuật toán chỉ số tăng tốc độ truy vấn một dữ liệu rộng lớn trong những bộ nhớ lưu trữ khác nhau.

# TÀI LIỆU THAM KHẢO

[1] N. Tsiftes, A. Dunkels, “A Database in Every Sensor”, SenSys’11, November , 2011.

[2]“*Antelope(Database Management System) – Contiki”* [Online], Available: <http://anrg.usc.edu/contiki/index.php/Antelope(Database_Management_System)-Contiki>

[3] P. H. Thái, “*Ngôn ngữ lập trình C/C++*”, Khoa Công Nghệ T,hông Tin, Trường Đại Học Công Nghệ, Đại Học Quốc Gia Hà Nội, 2003.

[4] Fredrik Österlind, “*A Sensor Network Simulator for the Contiki OS*” , SICS Technical Report, 2006.

[5] A.J. Sammes , Cranﬁ eld University , Swindon, UK ,“*Computer Communications and Networks”.*

[6] Dunkels A, Gronvall B, Voigt T (2004) Contiki – “*a lightweight and ﬂexible operating system for tiny networked sensors”*. In: Proceedings of the 29th annual IEEE international conference on local computer networks. IEEE Computer Society, Los Alamitos, pp 455–462.

[7] Suraiya Tarannum ,“*Wireless Sensor Networks*”,2011.

[8] “*Heap(DataStructure)*”[Online], Available: <https://en.wikipedia.org/wiki/Heap_(data_structure)>.

[9]N. Tsiftes, A. Dunkels, Z. He, and T. Voigt. Enabling Large-Scale Storage in Sensor Networks with the Coffee File System. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, San Francisco, CA, USA, Apr 2009.

[10] H. Dai, M. N., and R. Han. Elf: an efficient log-structured flash file system for micro sensor nodes. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (ACM SenSys)*,Baltimore, MD, USA, Nov. 2004.

[11] E. Gal and S. Toledo. A transactional flash file system for microcontrollers. In *Proceedings of the USENIX Annual Technical Conference*, Anaheim, CA, USA, Apr. 2005.

[12] M. Rosenblum and J. Ousterhout. The design and implementation of a log structured file system. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, Pacific Grove, CA, USA, 1991

[13] S. Nath and A. Kansal. FlashDB: Dynamic self-tuning database for NAND flash. In *Proceedings of the International Conference on Information Processing in Sensor Networks (ACM/IEEE IPSN)*, Cambridge, MA, USA, Apr. 2007.

# PHỤ LỤC

4Bảng 1 : Thời gian thực hiện truy vấn khi thay đổi chỉ số của thuộc tính.

|  |  |  |  |
| --- | --- | --- | --- |
| Số tuples được lựa chọn | Thời gian thực hiện truy vấn(s) | | |
| NO INDEX | INLINE | MAXHEAP |
| 1 | 21,014 | 0,105 | 0,15 |
| 10 | 21,015 | 0,117 | 0,711 |
| 100 | 21,023 | 0,221 | 6,312 |
| 1000 | 21,087 | 1,261 | 21,091 |
| 10,000 | 21,735 | 11,641 | 21,739 |
| 20,000 | 22,455 | 23,225 | 22,459 |

5Bảng 2 : Thời gian thực hiện truy vấn của thuộc tính INLINE khi thay đổi số hops.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Số tuples  được lựa chọn | Thời gian thực hiện truy vấn(s) | | | | |
| 1 hops | 2 hops | 3 hops | 4 hops | 5 hops |
| 1 | 0,105 | 0,12 | 0,136 | 0,15 | 0,164 |
| 10 | 0,117 | 0,132 | 0,147 | 0,162 | 0,177 |
| 100 | 0,221 | 0,236 | 0,252 | 0,267 | 0,281 |
| 1000 | 1,261 | 1,276 | 1,391 | 1,306 | 1,321 |
| 10,000 | 11,641 | 11,669 | 11,686 | 11,699 | 11,717 |
| 20,000 | 23,225 | 23,24 | 23,258 | 23,269 | 23,29 |

6Bảng 3 : Thời gian thực hiện truy vấn của thuộc tính MAXHEAP khi thay đổi số hops.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Số tuples  được lựa chọn | Thời gian thực hiện truy vấn(s) | | | | |
| 1 hops | 2 hops | 3 hops | 4 hops | 5 hops |
| 1 | 0,15 | 0,165 | 0,18 | 0,195 | 0,21 |
| 10 | 0,711 | 0,725 | 0,74 | 0,754 | 0,77 |
| 100 | 6,312 | 6,326 | 6,341 | 6,356 | 6,372 |
| 1000 | 21,091 | 21,102 | 21,117 | 21,132 | 21,15 |
| 10,000 | 21,739 | 21,749 | 21,765 | 21,78 | 21,798 |
| 20,000 | 22,459 | 22,469 | 22,485 | 22,499 | 22,518 |

7Bảng 4 : Thời gian thực hiện truy vấn của thuộc tính NO INDEX khi thay đổi số hops.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Số tuples  được lựa chọn | Thời gian thực hiện truy vấn(s) | | | | |
| 1 hops | 2 hops | 3 hops | 4 hops | 5 hops |
| 1 | 21,014 | 21,029 | 21,047 | 21,058 | 21,076 |
| 10 | 21,015 | 21,03 | 21,048 | 21,059 | 21,078 |
| \100 | 21,023 | 21,036 | 21,055 | 21,066 | 21,085 |
| 1000 | 21,087 | 21,102 | 21,121 | 21,131 | 21,15 |
| 10,000 | 21,735 | 21,749 | 21,768 | 21,779 | 21,804 |
| 20,000 | 22,455 | 22,47 | 22,488 | 22,499 | 22,518 |

8Bảng 5 : Năng lượng tiêu tốn của truy vấn chỉ số InLine

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tuples | CPU | LPM | Transmit | Listen | Total | EC/Tuples |
| 1 | 6699 | 91481 | 150 | 98029 | 65.82645 | 65.82645003 |
| 2 | 6747 | 91555 | 149 | 98153 | 65.91111 | 32.95555492 |
| 20 | 7420 | 90883 | 150 | 98153 | 65.94756 | 3.297377813 |
| 200 | 13710 | 84592 | 150 | 98151 | 66.28128 | 0.331406412 |
| 2000 | 80306 | 116300 | 153 | 196455 | 135.394 | 0.067697016 |
| 5000 | 188486 | 106424 | 153 | 294758 | 206.7194 | 0.041343889 |
| 10000 | 366543 | 26672 | 154 | 393061 | 281.7677 | 0.028176769 |
| 15000 | 548542 | 41281 | 153 | 589669 | 422.5889 | 0.028172593 |
| 20000 | 730999 | 55430 | 153 | 786278 | 563.4358 | 0.028171788 |

9Bảng 6 : Năng lượng tiêu tốn của truy vấn chỉ số MaxHeap

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tuples | CPU | LPM | Transmit | Listen | Total | EC/Tuples |
| 1 | 8608 | 89692 | 175 | 63474 | 42.95436 | 42.95435529 |
| 2 | 10478 | 87826 | 150 | 98153 | 66.11045 | 33.05522639 |
| 20 | 46474 | 51829 | 149 | 98152 | 68.02664 | 3.401332063 |
| 200 | 425693 | 164129 | 152 | 589672 | 416.0463 | 2.080231633 |
| 2000 | 661193 | 26934 | 154 | 687973 | 494.1537 | 0.247076869 |
| 5000 | 668031 | 20097 | 154 | 687973 | 494.518 | 0.098903598 |
| 10000 | 679627 | 8500 | 154 | 687974 | 495.1364 | 0.049513635 |
| 15000 | 695232 | 91199 | 155 | 786277 | 561.531 | 0.037435403 |
| 20000 | 706965 | 79467 | 153 | 786277 | 562.1548 | 0.028107742 |

10Bảng 7 : Năng lượng tiêu tốn của truy vấn không sử dụng thuật toán chỉ số

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Tuples | CPU | LPM | Transmit | Listen | Total | EC/Tuples |
| 1 | 656633 | 31493 | 153 | 687973 | 493.9102 | 493.9102377 |
| 2 | 656621 | 31505 | 154 | 687973 | 493.9102 | 246.9550968 |
| 20 | 656600 | 31528 | 154 | 687972 | 493.9084 | 24.69542065 |
| 200 | 657000 | 124738 | 154 | 687972 | 494.0854 | 2.470427066 |
| 2000 | 661188 | 26940 | 154 | 687973 | 494.1535 | 0.247076737 |
| 5000 | 671922 | 114509 | 154 | 786277 | 560.2888 | 0.112057751 |
| 10000 | 683432 | 102999 | 154 | 786277 | 560.9019 | 0.056090188 |
| 15000 | 695367 | 91058 | 154 | 786277 | 561.5376 | 0.037435842 |
| 20000 | 710386 | 174347 | 154 | 884582 | 627.9018 | 0.031395092 |

11Bảng 8 : Thời gian xử lý của các hoạt động trong Antelope.

|  |  |
| --- | --- |
| Hoạt động | Thời gian xử lý (ms) |
| INSERT (NO INDEX) | 28 |
| INSERT (INLINE) | 28 |
| INSERT (MAXHEAP) | 30 |
| CREATE INDEX (INLINE) | 32 |
| REMOVE REL | 33 |
| CREATE ATTRI | 34 |
| CREATE REL | 72 |

12Bảng 9 : Thời gian tạo một cơ sở dữ liệu với số tuple và thuật toán chỉ số tương ứng

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Số tuple | 20000 | 10000 | 5000 | 2000 | 1000 | 100 | 10 |
| MAXHEAP | 225 | 103 | 49 | 19 | 9,5 | 2 | 1,4 |
| INLINE | 115,6 | 57,7 | 29,4 | 12,4 | 6,7 | 1,8 | 1,3 |
| NO INDEX | 115,2 | 57,2 | 29,3 | 12,3 | 6,7 | 1,8 | 1,3 |

13Bảng 10 : Hiệu suất truy vấn theo số hop

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Số hop | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Tỉ  lệ  thành công | Tx – Rx : 90% | 67,3 | 38,6 | 32 | 14 | 20,6 | 16,6 | 6 | 8 | 1,33 | 2 |
| Tx – Rx : 95% | 84,6 | 71,3 | 64,6 | 54 | 48 | 32 | 30 | 25,3 | 14 | 16,6 |
| Tx – Rx : 100% | 100 | 100 | 98 | 92,6 | 94 | 94 | 86,6 | 87,3 | 84 | 80 |

14Bảng 11: Năng lượng tiêu tốn trên mỗi hoạt động

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hoạt động | CPU | LPM | Transmit | Listen | EC |
| CREATE RELATION | 5874 | 92428 | 55 | 98249 | 0.065873 |
| CREATE ATTRIBUTE | 4575 | 93727 | 27 | 98277 | 0.065805 |
| CREATE INDEX INLINE | 4553 | 93748 | 26 | 98277 | 0.065804 |
| CREATE INDEX MAXHEAP | 5922 | 92380 | 27 | 98276 | 0.065876 |
| INSERT (NO INDEX) | 4406 | 93896 | 26 | 98276 | 0.065795 |
| INSERT (INLINE) | 4734 | 93569 | 27 | 98277 | 0.065814 |
| INSERT (MAXHEAP) | 4471 | 93830 | 27 | 98276 | 0.065799 |
| REMOVE RELATION | 4648 | 93653 | 27 | 98277 | 0.065809 |

**Khối code của client: netdb-client.c:**

#include <stdio.h>

#include <string.h>

#include "contiki.h"

#include "dev/serial-line.h"

#include "net/rime/rime.h"

#include "net/rime/mesh.h"

#include "antelope.h"

/\*---------------------------------------------------------------------------\*/

#define MAX\_QUERY\_SIZE 100

#define NETDB\_CHANNEL 70

#ifndef SERVER\_ID

#define SERVER\_ID 4

#endif

/\*---------------------------------------------------------------------------\*/

PROCESS(netdb\_process, "NetDB");

AUTOSTART\_PROCESSES(&netdb\_process);

static unsigned server\_id = SERVER\_ID;

static struct mesh\_conn mesh;

/\*---------------------------------------------------------------------------\*/

PROCESS(shell\_process, "Shell Process");

PROCESS\_THREAD(shell\_process, ev, data)

{

linkaddr\_t addr;

PROCESS\_BEGIN();

printf("NetDB client\n");

for(;;) {

PROCESS\_WAIT\_EVENT\_UNTIL(ev == serial\_line\_event\_message && data != NULL);

if(strncmp(data, "server ", 7) == 0) {

server\_id = atoi((char \*)data + 7);

} else {

printf("%lu Transmitting query \"%s\" to node %u\n", clock\_time(), (char \*)data, server\_id);

packetbuf\_copyfrom(data, strlen(data));

addr.u8[0] = server\_id;

addr.u8[1] = 0;

packetbuf\_set\_attr(PACKETBUF\_ATTR\_PACKET\_TYPE,

PACKETBUF\_ATTR\_PACKET\_TYPE\_STREAM);

mesh\_send(&mesh, &addr);

}

}

PROCESS\_END();

}

/\*---------------------------------------------------------------------------\*/

static void

sent(struct mesh\_conn \*c)

{

}

static void

timedout(struct mesh\_conn \*c)

{

printf("Failed to send packet: time out\n");

}

static void

received(struct mesh\_conn \*c, const linkaddr\_t \*from, uint8\_t hops)

{

char \*data;

unsigned len;

static char reply[MAX\_QUERY\_SIZE + 1];

data = (char \*)packetbuf\_dataptr();

len = packetbuf\_datalen();

if(len > MAX\_QUERY\_SIZE) {

printf("Too long query: %d bytes\n", len);

return;

}

memcpy(reply, data, len);

reply[len] = '\0';

printf("%lu Reply received from %d.%d (%d hops): %s",

clock\_time(), from->u8[0], from->u8[1], (int)hops, reply);

}

static const struct mesh\_callbacks callbacks = {received, sent, timedout};

/\*---------------------------------------------------------------------------\*/

PROCESS\_THREAD(netdb\_process, ev, data)

{

PROCESS\_EXITHANDLER(mesh\_close(&mesh));

PROCESS\_BEGIN();

mesh\_open(&mesh, NETDB\_CHANNEL, &callbacks);

process\_start(&shell\_process, NULL);

PROCESS\_END();

}

**Khối code của server: netdb-server.c:**

#include <stdarg.h>

#include <stdio.h>

#include <string.h>

#include "contiki.h"

#include "dev/serial-line.h"

#include "dev/sht11/sht11.h"

#include "lib/random.h"

#include "net/rime/rime.h"

#include "net/rime/mesh.h"

#include "antelope.h"

/\*---------------------------------------------------------------------------\*/

/\* Sampling interval in Hz. \*/

#ifndef SAMPLING\_INTERVAL

#define SAMPLING\_INTERVAL 60

#endif

#ifndef RESPONSE\_LIMIT

#define RESPONSE\_LIMIT 1000

#endif

#ifndef PREPARE\_DB

#define PREPARE\_DB 1

#endif

#ifndef CARDINALITY

#define CARDINALITY 1000

#endif

#define MAX\_BUFFER\_SIZE 80

#define NETDB\_CHANNEL 70

/\*---------------------------------------------------------------------------\*/

PROCESS(netdb\_process, "NetDB");

AUTOSTART\_PROCESSES(&netdb\_process);

static struct mesh\_conn mesh;

static linkaddr\_t reply\_addr;

static uint8\_t buffer\_offset;

static char buffer[MAX\_BUFFER\_SIZE];

/\*---------------------------------------------------------------------------\*/

static void

send\_buffered\_data(void)

{

if(buffer\_offset > 0) {

packetbuf\_copyfrom(buffer, buffer\_offset);

mesh\_send(&mesh, &reply\_addr);

buffer\_offset = 0;

}

}

/\*---------------------------------------------------------------------------\*/

static int

buffer\_db\_data(const char \*format, ...)

{

va\_list ap;

size\_t len;

char tmp[MAX\_BUFFER\_SIZE + 1];

va\_start(ap, format);

len = vsnprintf(tmp, sizeof(tmp), format, ap);

va\_end(ap);

if(len < 0) {

return -1;

}

if(len + buffer\_offset > sizeof(buffer)) {

send\_buffered\_data();

}

memcpy(&buffer[buffer\_offset], tmp, len);

buffer\_offset += len;

return len;

}

/\*---------------------------------------------------------------------------\*/

#if !PREPARE\_DB

static void

take\_sample(void)

{

unsigned seconds;

unsigned humidity;

seconds = clock\_seconds();

humidity = /\*sht11\_humidity()\*/ random\_rand();

if(DB\_ERROR(db\_query(NULL, "INSERT (%u, %u) INTO samples;",

seconds, humidity))) {

printf("DB insertion failed\n");

}

}

#endif /\* !PREPARE\_DB \*/

/\*---------------------------------------------------------------------------\*/

static void

stop\_handler(void \*ptr)

{

printf("END\n");

}

/\*---------------------------------------------------------------------------\*/

PROCESS(query\_process, "Query process");

PROCESS\_THREAD(query\_process, ev, data)

{

static db\_handle\_t handle;

db\_result\_t result;

static tuple\_id\_t matching;

static tuple\_id\_t processed;

#if !PREPARE\_DB

static struct etimer sampling\_timer;

#endif

static unsigned i, errors;

PROCESS\_BEGIN();

printf("NetDB host\n");

db\_init();

db\_set\_output\_function(buffer\_db\_data);

db\_query(NULL, "REMOVE RELATION samples;");

db\_query(NULL, "CREATE RELATION samples;");

db\_query(NULL, "CREATE ATTRIBUTE time DOMAIN INT IN samples;");

db\_query(NULL, "CREATE ATTRIBUTE hum DOMAIN INT IN samples;");

db\_query(NULL, "CREATE INDEX samples.time TYPE INLINE;");

#if PREPARE\_DB

printf("Preparing the DB with %d tuples...\n", CARDINALITY);

errors = 0;

for(i = 1; i <= CARDINALITY; i++) {

PROCESS\_PAUSE();

result = db\_query(NULL, "INSERT (%u, %u) INTO samples;",

i, (unsigned)random\_rand());

if(DB\_ERROR(result)) {

errors++;

}

}

printf("Done. Insertion errors: %d\n", errors);

printf("Ready to process queries\n");

#else

etimer\_set(&sampling\_timer, SAMPLING\_INTERVAL \* CLOCK\_SECOND);

#endif

for(;;) {

PROCESS\_WAIT\_EVENT();

if(ev == serial\_line\_event\_message && data != NULL) {

printf("START %s\n", (char \*)data);

result = db\_query(&handle, data);

if(DB\_ERROR(result)) {

buffer\_db\_data("Query error: %s\n", db\_get\_result\_message(result));

stop\_handler(NULL);

db\_free(&handle);

continue;

}

if(!db\_processing(&handle)) {

buffer\_db\_data("OK\n");

send\_buffered\_data();

stop\_handler(NULL);

continue;

}

packetbuf\_set\_attr(PACKETBUF\_ATTR\_PACKET\_TYPE,

PACKETBUF\_ATTR\_PACKET\_TYPE\_STREAM);

db\_print\_header(&handle);

matching = 0;

processed = 0;

while(db\_processing(&handle)) {

PROCESS\_PAUSE();

if(matching == RESPONSE\_LIMIT) {

buffer\_db\_data("Response suppressed at %u tuples: limit reached\n",

RESPONSE\_LIMIT);

stop\_handler(NULL);

db\_free(&handle);

break;

}

result = db\_process(&handle);

if(result == DB\_GOT\_ROW) {

/\* The processed tuple matched the condition in the query. \*/

matching++;

processed++;

db\_print\_tuple(&handle);

} else if(result == DB\_OK) {

/\* A tuple was processed, but did not match the condition. \*/

processed++;

continue;

} else {

if(result == DB\_FINISHED) {

/\* The processing has finished. Wait for a new command. \*/

buffer\_db\_data("[%ld tuples returned; %ld tuples processed]\n",

(long)matching, (long)processed);

buffer\_db\_data("OK\n");

} else if(DB\_ERROR(result)) {

buffer\_db\_data("Processing error: %s\n",

db\_get\_result\_message(result));

}

stop\_handler(NULL);

db\_free(&handle);

}

}

send\_buffered\_data();

}

#if !PREPARE\_DB

if(etimer\_expired(&sampling\_timer)) {

take\_sample();

etimer\_reset(&sampling\_timer);

}

#endif

}

PROCESS\_END();

}

/\*---------------------------------------------------------------------------\*/

static void

sent(struct mesh\_conn \*c)

{

}

static void

timedout(struct mesh\_conn \*c)

{

printf("packet time out\n");

}

static void

received(struct mesh\_conn \*c, const linkaddr\_t \*from, uint8\_t hops)

{

char \*data;

unsigned len;

static char query[MAX\_BUFFER\_SIZE + 1];

data = (char \*)packetbuf\_dataptr();

len = packetbuf\_datalen();

if(len > MAX\_BUFFER\_SIZE) {

buffer\_db\_data("Too long query: %d bytes\n", len);

return;

}

memcpy(query, data, len);

query[len] = '\0';

printf("Query received from %d.%d: %s (%d hops)\n",

from->u8[0], from->u8[1], query, (int)hops);

linkaddr\_copy(&reply\_addr, from);

process\_post(&query\_process, serial\_line\_event\_message, query);

}

static const struct mesh\_callbacks callbacks = {received, sent, timedout};

/\*---------------------------------------------------------------------------\*/

PROCESS\_THREAD(netdb\_process, ev, data)

{

PROCESS\_EXITHANDLER(mesh\_close(&mesh));

PROCESS\_BEGIN();

mesh\_open(&mesh, NETDB\_CHANNEL, &callbacks);

process\_start(&query\_process, NULL);

PROCESS\_END();

}