MỤC LỤC

LỜI NÓI ĐẦU

Ngày nay nhờ có những tiến bộ nhanh chóng trong khoa học và công nghệ sự phát triển của những mạng bao gồm các cảm biến giá thành rẻ, tiêu thụ ít năng lượng và đa chức năng đã nhận được những sự chú ý đáng kể. Hiện nay người ta đang tập trung triển khai các mạng cảm biến để áp dụng vào trong cuộc sống hàng ngày. Đó là các lĩnh vực về y tế, quân sự, môi trường, giao thông… Trong một tương lai không xa, các ứng dụng của mạng cảm biến sẽ trở thành một phần không thể thiếu trong cuộc sống con người nếu chúng ta phát huy được hết các điểm mạnh mà không phải mạng nào cũng có được như mạng cảm biến.

Tuy nhiên mạng cảm ứng đang phải đối mặt với rất nhiều thách thức, một trong những thách thức lớn nhất đó là nguồn năng lượng bị giới hạn khả năng xử lý thấp, giá thành thấp, giải thông bé, tín hiệu yếu và hoạt động dưới tần số chia sẻ. Hiện nay rất nhiều nhà nghiên cứu đang tập trung vào việc cải thiện khả năng sử dụng hiệu quả năng lượng của mạng cảm biến trong từng lĩnh vực khác nhau.

Trong quá trình tìm hiểu và nghiên cứu về mạng cảm biến, em đã lựa chọn đề tài đánh giá hiệu quả năng lượng một số giao thức điều khiển xâm nhập môi trường trong mạng cảm biến không dây làm đồ án tốt nghiệp

Đồ án này gồm 4 chương:

Chương I: Tổng quan về mạng cảm biến không dây.

Chương II: Một số giao thức MAC trong mạng cảm biến không dây.

Chương III: Phần mềm mô phỏng mạng OMNET++.

Chương IV: Mô phỏng và đánh giá hiệu quả năng lượng của CSMA, SMAC, TMAC.

LỜI CẢM ƠN

Để có thể hoàn thành được đồ án tốt nghiệp này, em đã được học hỏi những kiến thức quí báu từ các thầy, cô giáo của Trường Đại Học Dân Lập Hải Phòng trong suốt bốn năm đại học. Em vô cùng biết ơn sự dạy dỗ, chỉ bảo tận tình của các thầy, các cô trong thời gian học tập này.

Em xin bày tỏ lòng biết ơn tới thầy Nguyễn Trọng Thể - Khoa công nghệ thông tin – Trường Đại Học Dân Lập Hải Phòng đã tận tình chỉ bảo và định hướng cho em nghiên cứu đề tài này. Thầy đã cho em những lời khuyên quan trọng trong suốt quá trình hoàn thành đồ án. Cuối cùng, em xin cảm ơn gia đình và bạn bè luôn tạo điều kiện thuận lợi, động viên và giúp đỡ em trong suốt thời gian học tập, cũng như quá trình nghiên cứu, hoàn thành đồ án này.

Do hạn chế về thời gian thực tập, tài liệu và trình độ bản thân, bài đồ án của em không thể tránh khỏi những thiếu sót, rất mong các thầy cô góp ý và sửa chữa để bài đồ án tốt nghiệp của em được hoàn thiện hơn. Em xin chân thành cảm ơn!

CHƯƠNG I: TỔNG QUAN VỀ MẠNG CẢM BIẾN KHÔNG DÂY

1. Định nghĩa:

Mạng cảm biến không dây (WSN) có thể hiểu đơn giản là mạng liên kết các node với nhau bằng kết nối sóng vô tuyến, trong đó các node mạng thường là các thiết bị đơn giản, nhỏ gọn, giá thành thấp… và có số lượng lớn, được phân bố một cách không có hệ thống trên một diện tích rộng, sử dụng nguồn năng lượng hạn chế và có thể hoạt động trong môi trường khắc nghiệt (chất độc, ô nhiễm, nhiệt độ cao…).

2. Cấu trúc của WSN:

Node cảm biến.

Một node cảm biến được cấu tạo bởi 3 thành phần cơ bản sau: Vi điều khiển,

Sensor, bộ phát radio. Ngoài ra còn có các cổng kết nối máy tính.

2.1.1 Vi điều khiển.

Bao gồm: CPU; bộ nhớ ROM, RAM; bộ phận chuyển đổi tín hiệu tương tự thành tín hiệu số và ngược lại.

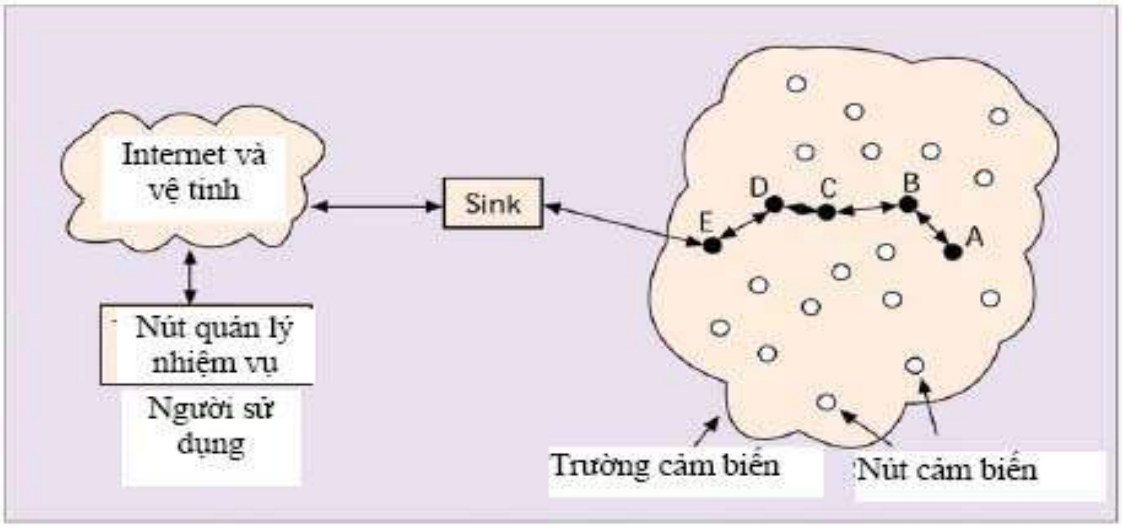
2.1.2 Sensor.

Chức năng: cảm nhận thế giới bên ngoài, sau đó chuyển dữ liệu qua bộ phận chuyển đổi để xử lý.

2.1.3 Bộ phát radio.

Node cảm biến là thành phần quan trọng nhất trong WSN, do vậy việc thiết kế các node cảm biến sao cho có thể tiết kiệm được tối đa nguồn năng lượng là vấn đề quan trọng hàng đầu.

Mạng cảm nhận.



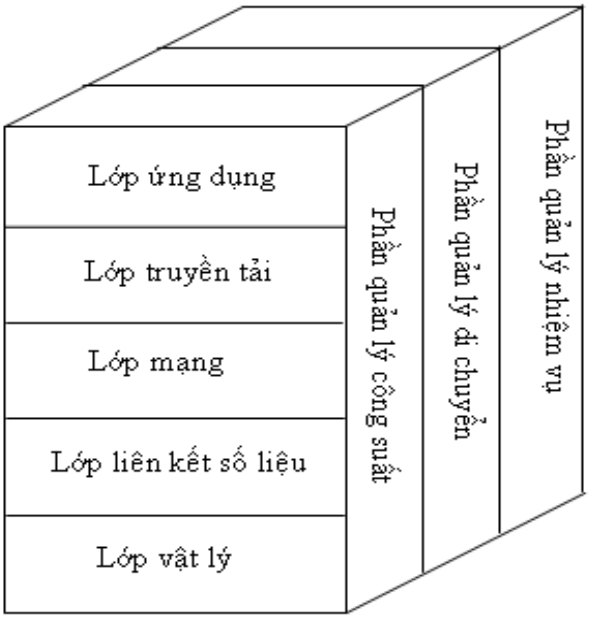
Hình 1.1 Phân bố node cảm biến trong trường hợp cảm biến.

Hình 1.1 chúng ta thấy, mạng cảm nhận bao gồm rất nhiều các node cảm biến được phân bố trong một trường cảm biến. Các node này có khả năng thu thập dữ liệu thực tế, sau đó chọn đường (theo phương pháp đa bước nhảy) để chuyển những dữ liệu này về node gốc. Node gốc liên lạc với node quản lý nhiệm vụ thông qua Internet hoặc vệ tinh. Việc thiết kế mạng cảm nhận như trong mô hình 1.1 phụ thuộc vào nhiều yếu tố như:

- Khả năng chịu lỗi: Một số các node cảm biến có khả năng không hoạt động nữa do thiếu năng lượng, do những hư hỏng vật lý hoặc do ảnh hưởng của môi trường. Khả năng chịu lỗi thể hiện ở việc mạng vẫn hoạt động bình thường, duy trì những chức năng của nó ngay cả khi một số node mạng không hoạt động.
- Khả năng mở rộng: Khi nghiên cứu một hiện tượng, số lượng các node cảm biến được triển khai có thể đến hàng trăm nghìn node, phụ thuộc vào từng ứng dụng mà con số này có thể vượt quá hàng trăm nghìn node. Do cấu trúc mạng có khả năng mở rộng để phù hợp với từng ứng dụng cụ thể.
- Giá thành sản xuất: Vì mạng cảm nhận bao gồm một số lượng lớn các node cảm biến nên chi phí mỗi node là rất quan trọng

trong việc điều chỉnh chi phí mạng. Do vậy chi phí ở mỗi node cảm biến phải giữ ở mức thấp.

- Tích hợp phần cứng: Vì số lượng node cảm biến trong mạng là nhiều nên node cảm biến cần phải có các rằng buộc phần cứng sau: Kích thước nhỏ, tiêu thụ năng lượng ít, chi phí sản xuất ít, thích hợp với môi trường, có khả năng tự cấu hình và hoạt động không cần giám sát.
- Môi trường hoạt động: Các node cảm biến thường khá dày đặc và phân bố trực tiếp trong môi trường (kể cả môi trường ô nhiễm, độc hại hay dưới nước…). Node cảm biến phải thích ứng với nhiều loại môi trường và sự thay đổi của môi trường.
- Các phươngg tiện truyền dẫn: Ở mạng cảm nhận, các node được kết nối với nhau trong môi trường không dây, môi trường truyền dẫn có thể là sóng vô tuyến, hồng ngoại hoặc những phương tiện quang học. Để thiêt lập được sự hoạt động thống nhất chung cho các mạng này thì các phương tiện truyền dẫn phải được chọn phù hợp trên toàn thế giới.
- Cấu hình mạng cảm nhận: Mạng cảm nhận bao gồm một số lượng lớn các node cảm biến, do đó phải thiết lập một cấu hình ổn định.
- Sự tiêu thụ năng lượng: Mỗi node cảm biến được trang bị nguồn năng lượng giới hạn. Trong một số ứng dụng, việc bổ sung nguồn năng lượng là không thê thực hiện được. Vì vậy thời gian sống của mạng phụ thuộc vào thời gian sống của node cảm biến, thời gian sống của node cảm biến lại phụ thuộc vào thời gian sống của pin. Do vậy, hiện nay các nhà khoa học đang nỗ lực tìm ra các giải thuật và giao thức thiết kế cho các node mạng nhằm tiết kiệm nguồn năng lượng hạn chế này.
- Kiến trúc giao thức mạng cảm nhận:



Hình 1.2. Kiến trúc giao thức mạng cảm biến.

Kiến trúc giao thức áp dụng cho mạng cảm nhận được trình bày trong hình 1.2. Kiến trúc này bao gồm các lớp và các mặt phẳng quản lý. Các mặt phẳng quản lý này làm cho các node có thể làm việc cùng nhau theo cách có hiệu quả nhất, định tuyến dữ liệu trong mạng cảm nhận di động và chia sẻ tài nguyên giữa các node cảm biến.

+Lớp vật lý: có nhiệm vụ lựa chọn tần số, tạo ra tần số sóng mang,phát hiện tín hiệu, điếu chế và mã hoá tín hiệu.

+ Lớp liên kết số liệu: Có nhiệm vụ ghép các luồng dữ liệu, phát hiện các khung dữ liệu, cách truy cập đường truyền và điều khiển lỗi. Vì môi trường có tạp âm và các node cảm biến có thể di động, giao thức điều khiển truy nhập môi trường (MAC) phải xết đến vấn đề công suất và phải có khả năng tối ưu hoá việc va cham với thông tin quảng bá của các node lân cận.

+ Lớp mạng: Quan tâm đến việc chọn đường số liệu được cung cấp bởi lớp truyền tải.

+ Lớp truyền tải: giúp duy trì luồng số liệu nêứu ứng dụng mạng cảm nhận yêu cầu. Lớp truyền tải chỉ cần thiết khi hệ thống có kế hoạch được truy cập thông qua mạng Internet và các mạng bên ngoài khác.

+ Lớp ứng dụng: tuỳ theo nhiệm vụ cảm biến, các loại phần mềm ứng dụng khác nhau có thể được xây dựng và sử dụng ở lớp ứng dụng.

+ Mặt phẳng quản lý công suất: Điều khiển việc sử dụng công suất của node cảm biến. Ví dụ:

- Node cảm biến có thể tắt bộ thu sau khi nó nhận một bản tin để tránh tạo ra các bản tin giống nhau.
- Khi mức công suất của node cảm biến thấp, nó sẽ phát quảng bá sang các node cảm biến bên cạnh thông báo rằng mức năng lượng của nó thấp và nó không thể tham gia vào quá trình định tuyến. Công suất còn lại được giành cho nhiệm vụ cảm biến.

+ Mặt phẳng quản lý di chuyển: Có nhiệm vụ phát hiện và đăng ký sự chuyển động của các node. Từ đó có thể xác định xem ai là hàng xóm của mình.

+ Mặt phẳng quản lý nhiệm vụ: Có nhiệm vụ cân bằng và sắp xếp nhiệm vụ cảm biến giữa các node trong vùng đó đều thực hiện nhiệm vụ cảm biến tai cùng một thời điểm.

3. Ứng dụng của WSN.

WSN bao gồm các node cảm biến nhỏ. Thích ứng được môi trường khắc nghiệt. Những node cảm biến này, cảm nhận được môi trường xung quanh, sau đó gửi những thông tin thu được đến trung tâm để xử lý theo ứng dụng. Các node không nhừng có thể liên lạc với các node xung quanh nó, mà còn có thể xử lý dữ liệu thu được trước khi gửi đến các node khác. WSN cung cấp rất nhiều các ứng dụng hữu ích ở nhiều lĩnh vực trong cuộc sống.

- Các ứng dụng trong bảo vệ môi trường

  + Phát hiện mìn, chất độc trong môi trường.

  + Giám sát lũ lụt, bão, gió, mưa…

  + Phát hiện ô nhiễm, chất thải.

  + Phát hiện hoạt động núi lửa.

  + Phát hiện độnh đất.

+ Giám sát cháy rừng.

- Các ứng dụng trong y tế.

+ Định vị theo dõi bệnh nhân.

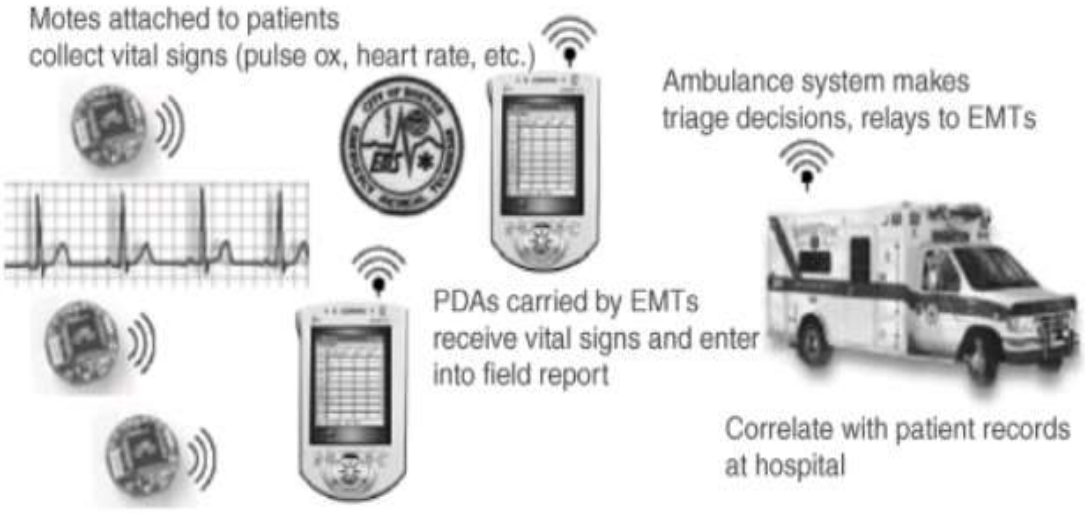
+ Hệ thống báo động khẩn cấp.

+ Cảm biến gắn trực tiếp lên cơ thể con người.

+ phân tích nồng độ các chất.

+ Chăm sóc sức khoẻ.

+ Hỗ trợ chăm sóc bệnh nhân.



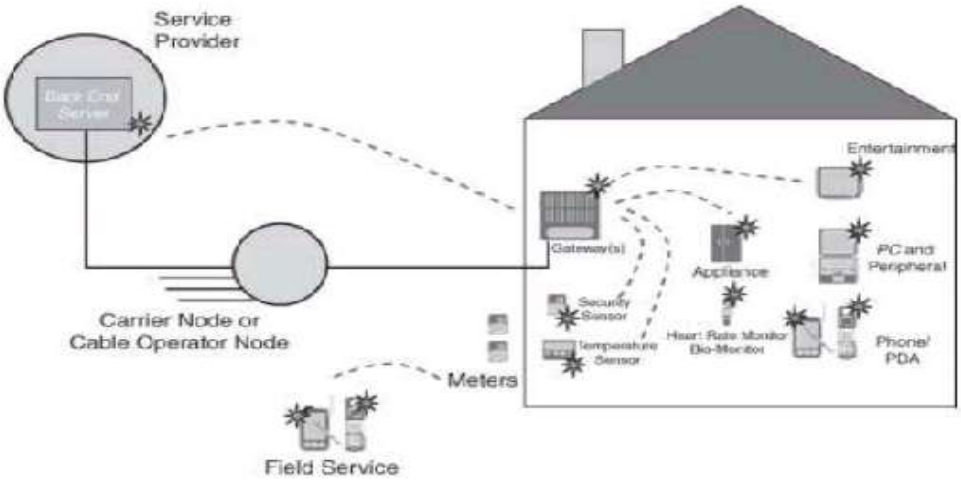
Hình 1.3. Ứng dụng trong y tế.

- Các ứng dụng trong gia đình.

+ Hệ thống giao tiếp và điều khiển từ xa các thiết bị.

+ Hệ thống cảnh báo an ninh…

….



Hình 1.4. Ứng dụng điều khiển trong gia đình.

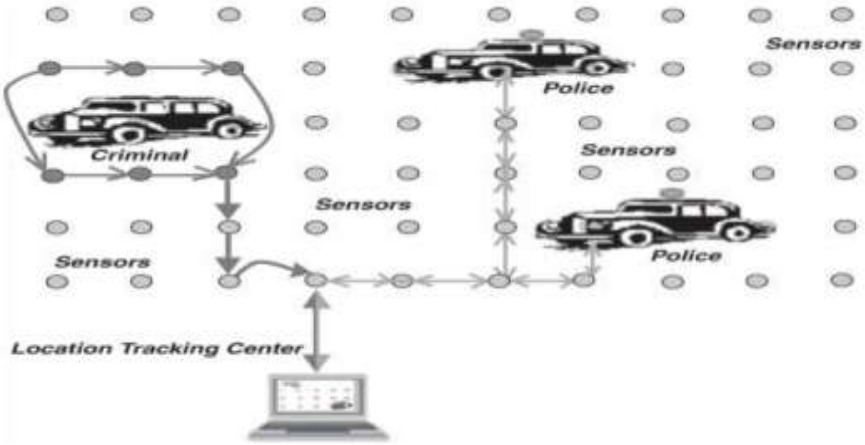
- Hệ thống giao thông thông minh

  + Giao tiếp giữa biển báo và phương tiện giao thông.

  + Hệ thống điều tiết lưu lượng công cộng.

  + Hệ thống báo hiệu tai nạn, kẹt xe…

  + Hệ thống định vị phương, trợ giúp điều khiển tự động phương tiện giao thông.



Hình 1.5. Ứng dụng định vị phương tiện giao thông.

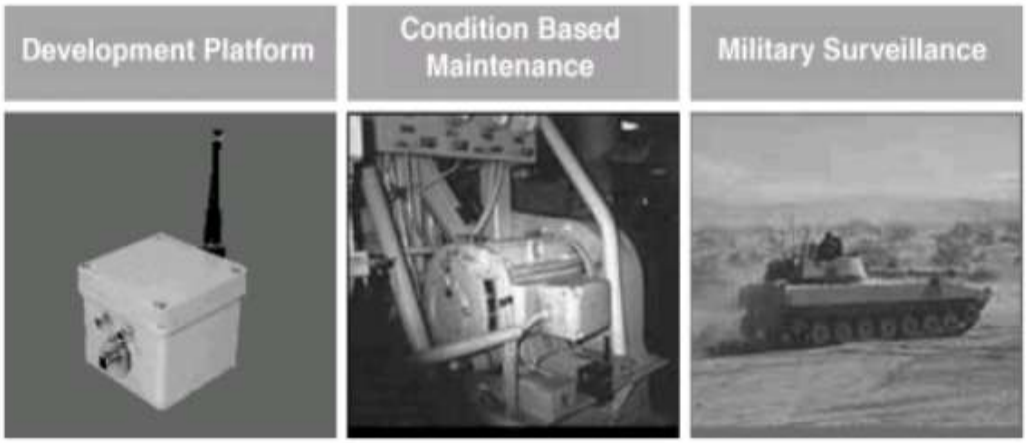
- Ứng dụng trong quân sự, an ninh

  + Định vị, theo dõi di chuyển của các thiết bị quân sự.

  + Điều khiển tự động các thiết bị quân sự, robot..

  + Kích hoạt thiết bị, vũ khí quân sự.

+ theo dõi biên giới kếyt hợp với vệ tinh.



Hình 1.6. Ứng dụnh cảm biến trong quân sự.

- Ứng dụng trong thương mại

  + Quản lý kiến trúc và xây dựng.

  + Quản lý sản xuất.

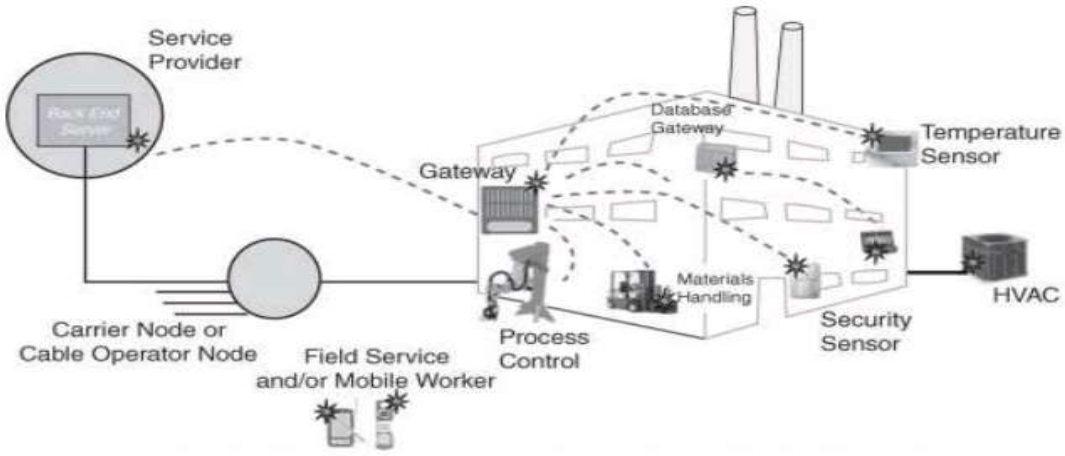
  + Hệ thống xử lý vật liệu.

  + Quản lý tải trong tiêu thụ điện năng.

  + Điều khiển nhiệt độ.

  + Hệ thống tự động.

  + Thu thập dữ liệu thơi gian thực.



Hình 1.7. Các ứng dụng trong công nghiệp.

4. Những thách thức của WSN.

Để WSN thực sự trở lên rộng khắp trong các ứng dụng, một số thách thức và trở ngại cần vượt qua:

- Vấn đề về năng lượng.
- Năng lực xử lý, tính toán.
- Bộ nhớ lưu trữ
- Thích ứng tốt với môi trường
- Ngoài ra còn có một số thách thức trở ngại thứ yếu như: Vấn đề mở rộng mạng, giá thành các node, quyền sở hữu…

5. Sự khác nhau giữa WSN và mạng truyền thống

Dựa vào trình bày ở trên, ta dễ dàng nhận thấy sự khác nhau giữa WSN và các mạng truyền thống:

- Số lượng node cảm biến trong một mạng cảm nhận lớn hơn nhiều lần so với node khác trong các mạng truyền thống.
- Các node cảm biến thường được triển khai với mật độ dày hơn.
- Những node cảm biến dễ hỏng, ngừng hoạt động hơn.
- Cấu trúc mạng cảm nhận thay đổi khá thường xuyên.
- Mạng cảm nhận chủ yếu sử dụng truyền thông quảng bá, trong khi đó đa số các mạng truyền thống là điểm – điểm.
- Những node cảm biến có giới hạn về năng lượng, khả năng tính toán, bộ nhớ.
- Những node cảm biến có thể không có số định dạng toàn cầu(global identification) (ID).
- Truyền năng lượng hiệu quả qua các phương tiện không dây.
- Chia sẻ nhiệm vụ giữa các node láng giềng.

CHƯƠNG II: MỘT SỐ GIAO THỨC MAC TRONG MẠNG CẢM BIẾN KHÔNG DÂY

I. Giao Thức Mac

Mạng cảm biến không dây là loại mạng đặc biệt với số lượng lớn nút cảm biến được trang bị bộ vi xử lý, thành phần cảm biến và thành phần quản lý sóng vô tuyến. Các nút cảm biến cộng tác với nhau để hoàn thành một nhiệm vụ chung. Trong nhiều ứng dụng, các nút cảm biến sẽ được triển khai phi cấu trúc như mạng ad hoc. Chúng phải tự tổ chức để hình thành một mạng không dây đa bước nhảy. Thách thức chung trong mạng không dây là vấn đề xung đột do hai nút gửi dữ liệu cùng lúc trên cùng kênh truyền.

Giao thức điều khiển truy nhập đường truyền (MAC) đã được phát triển để giúp đỡ mỗi nút quyết định khi nào và làm sao để truy nhập kênh. Vấn đề này cũng được biết như sự định vị kênh hoặc đa truy nhập. Lớp MAC được xem xét bình thường như một lớp con của lớp liên kết dữ liệu trong giao thức mạng. Những giao thức MAC đã nghiên cứu rộng rãi trên những lĩnh vực truyền thống của truyền thông tiếng nói và dữ liệu không dây. Đa truy nhập phân chia theo thời gian (Time Division multiple Access - TDMA), Đa truy nhập phân chia theo tần số (Frequency Division Multiple Access - FDMA) và đa truy nhập phân chia theo mã (Code Division Multiple Access - CDMA) là những giao thức MAC được sử dụng rộng rãi trong những hệ thống truyền thông tế bào hiện đại.

Ý tưởng cơ bản của các phương pháp trên truy nhập trên một kênh dung chung, kết quả trong sự phối hợp xác suất có điều kiện, không cần cấp phát sẵn kênh truyền.

Xung đột có thể xảy ra trong thời gian thủ tục cạnh tranh trong những hệ thống như vậy.

Mạng cảm biến khác với mạng dữ liệu không dây truyền thống trên một vài khía cạnh. Trước hết, đa số các nút trong những mạng cảm biến hoạt động dựa trên nguồn điện pin, và rất khó để nạp điện cho những nguồn pin của tất cả các nút. Thứ hai, những nút thường được triển khai trong một kiểu cách đặc biệt

phi cấu trúc; chúng phải tự tổ chức hình thành một mạng truyền thông. Ba là, nhiều ứng dụng cần phải sử dụng số lượng lớn những nút, và mật độ nút sẽ thay đổi tại những địa điểm và thời gian khác nhau, với cả những mạng mật độ thưa lẫn những nút với nhiều lân cận. Cuối cùng, đa số các lưu thông trong mạng được thúc đẩy bởi những sự kiện cảm ứng, phân bố không đều và rất co cụm. Tất cả những đặc trưng này cho thấy những giao thức MAC truyền thống không thích hợp cho những mạng cảm biến không dây nếu không có những sự cải biến.

1.Yêu cầu thiết kế giao thức MAC cho mạng cảm biến không dây

Tránh xung đột

Tính tránh xung đột (Collision Advoidance) là một yêu cầu cơ bản của tất cả các giao thức MAC, nó xác định khi nào một nút có thể truy nhập đường truyền và thực hiện trao đổi dữ liệu.

Hiệu quả năng lượng

Tính hiệu năng (Energy Efficiency) là một trong những thuộc tính quan trọng nhất những giao thức MAC mạng cảm biến. Như đã đề cập ở trên, đa số các nút cảm biến hoạt động bằng pin, rất khó để thay đổi hoặc nạp điện lại cho pin của những nút này. Thực tế, nhiều mục đích thiết kế của những mạng cảm biến được xây dựng bằng những nút đủ rẻ để vứt bỏ hơn là nạp lại. Trong tất cả các trường hợp, việc kéo dài cả cuộc đời của mỗi nút là một vấn đề then chốt. Dù với nền tảng phần cứng nào, năng lượng cho thu phát sóng vô tuyến là nguồn tiêu thụ năng lượng chính. Lớp MAC trực tiếp điều khiển hoạt động thu phát sóng vô tuyến, và sự tiêu thụ năng lượng của nó như thế nào ảnh hưởng đáng kể tới cả cuộc đời của nút.

Khả năng thích ứng và biến đổi được

Tính biến đổi được và khả năng thích ứng (Scalability and Adaptivity) là những thuộc tính liên quan của một giao thức MAC điều tiết những sự thay đổi trong kích thước mạng, mật độ và topo mạng. Nhiều nút có thể không hoặc ngừng hoạt động trong thời gian dài; vài nút mới có thể tham gia về sau; một vài nút khác có thể di chuyển tới những vị trí khác. Một giao thức MAC tốt cần phải

điều tiết những sự thay đổi như vậy một cách hợp lý. Tính biến đổi được và khả năng thích ứng để thay đổi trong kích thước, mật độ và topo mạng là những thuộc tính quan trọng, bởi vì những mạng cảm biến được triển khai phi cấu trúc và thường hoạt động trong những môi trường không chắc chắn.

Khả năng sử dụng kênh

Và độ *fairness* đối với từng nút hoặc từng người dùng trở nên ít quan trọng hơn.

Tóm lại, các vấn đề nêu ở trên là những thuộc tính thể *Sự sử dụng kênh (Channel utilization)* phản chiếu toàn bộ băng thông của kênh được dùng trong truyền thông ra sao, nó cũng được đề cập như sự sử dụng băng thông hoặc dung lượng kênh truyền. Đó là một vấn đề quan trọng đối với hệ thống điện thoại tế bào hoặc mạng cục bộ không dây (WLANs), khi băng thông là tài nguyên quý giá nhất trong những hệ thống như vậy và các nhà cung cấp dịch vụ đều muốn càng nhiều người dùng càng tốt. Mặt khác, số những nút hoạt động trong mạng cảm biến chủ yếu về được xác định bởi loại ứng dụng. Sự sử dụng kênh thường là một mục tiêu thứ nhì trong những mạng cảm biến.

Độ trễ

Độ trễ (Latency) đó là sự trì hoãn một nút gửi có một gói tin để gửi cho đến khi gói tin được nhận thành công bởi nút nhận. Trong mạng cảm biến, sự quan trọng của độ trễ phụ thuộc vào ứng dụng. Trong những ứng dụng như giám sát hoặc theo dõi, các nút cảm biến không hoạt động phần lớn thời gian cho đến khi một sự kiện nào đó được phát hiện. Những ứng dụng này có thể thường bỏ qua sự trễ thông điệp bổ sung nào đó, bởi vì tốc độ mạng nhanh hơn tốc độ của một đối tượng vật lý. Tốc độ cảm biến đối tượng đặt một ranh giới trên về tốc độ phản ứng mà mạng phải đạt được. Trong khoảng thời gian không có sự kiện cảm ứng, có rất ít dữ liệu trao đổi trong mạng. Sự trễ ở mức nhỏ hơn một giây cho một khởi tạo một thông báo sau thời kỳ nhàn rỗi thì không quan trọng bằng sự tiết kiệm năng lượng và thời gian hoạt động của thiết bị. Nhưng ngược lại, sau

khi cảm biến xác định được sự kiện, hoạt động với độ trễ thấp thành quan mục tiêu quan trọng.

Thông lượng

Thông lượng (Throughput) đề cập tới số lượng của dữ liệu chuyển thành công từ một nơi gửi đến một nơi nhận trong một khoảng thời gian cho trước. Nhiều nhân tố ảnh hưởng đến thông lượng, bao gồm hiệu quả của sự tránh xung đột, sự sử dụng kênh, độ trễ, và xử lý thông tin điều khiển. Giống với độ trễ, sự quan trọng của thông lượng phụ thuộc vào loại ứng dụng. Những ứng dụng cảm biến mà yêu cầu vòng đời lâu thường chấp nhận độ trễ nhiều hơn và thông lượng thấp hơn.

Công bằng

Fairness thể hiện khả năng những người dùng, những nút hoặc những ứng dụng khác nhau cùng nhau chia sẻ kênh truyền một cách công bằng. Nó là một thuộc tính quan trọng trong mạng tiếng nói hoặc những mạng dữ liệu truyền thống, một khi mỗi người dùng mong muốn một cơ hội như nhau để gửi hoặc nhận dữ liệu cho những ứng dụng của chính mình. Tuy nhiên, trong những mạng cảm biến, tất cả các nút hợp tác cho một nhiệm vụ chung đơn lẻ. Ở tại thời điểm đặc biệt, một nút có thể có nhiều dữ liệu hơn để gửi so với các nút khác, như vậy, hơn là đối xử với mỗi nút công bằng, thành công được đo bởi sự thực hiện của ứng dụng, hiện những đặc trưng của một giao thức MAC. Đối với mạng cảm biến không dây, những yếu tố quan trọng nhất là sự tránh xung đột có hiệu quả, hiệu quả năng lượng, tính biến đổi và thích ứng được với mật độ và số lượng nút.

2. Các nguyên nhân gây nên lãng phí năng lượng

Xung đột

Sự xung đột (Collision) là nguyên nhân đầu tiên gây tiêu phí năng lượng. Khi hai gói được truyền cùng thời điểm sẽ xảy ra xung đột, chúng bị hỏng và phải được loại bỏ. Yêu cầu truyền lại gói tin sau đó sẽ làm phát sinh sự tiêu hao năng lượng. Do đó tất cả các giao thức MAC cố gắng tránh xung đột bằng mọi cách.

Nghe khi rỗi

Nguyên nhân thứ hai gây tiêu hao năng lượng là vấn đề *nghe khi rỗi (Idle Listening)*. Nó xảy ra khi thành phần sóng vô tuyến thực hiện "nghe" kênh xem có dữ liệu không để nhận. Sự tiêu hao này đặc biệt cao trong những ứng dụng mạng cảm biến, nơi không có dữ liệu trao đổi trong thời gian không có sự kiện được cảm biến.

Nhiều giao thức MAC (như CSMA và CDMA) luôn luôn nghe kênh khi hoạt động dù không có dữ liệu để gửi. Chi phí chính xác của vấn đề nghe khi rỗi phụ thuộc vào phần cứng và chế độ hoạt động thành phần sóng vô tuyến. Đa số các mạng cảm biến được thiết kế để hoạt động trong thời gian dài và các nút cảm biến cũng sẽ trong ở trạng thái nghe khi rỗi một thời gian dài. Trong những trường hợp như vậy, nghe khi rỗi là một yếu tố chính trong vấn đề tiêu thụ năng lượng của thành phần sóng vô tuyến.

Nghe thừa

Nguyên nhân thứ ba là vấn đề *nghe thừa (overhearing)* xuất hiện khi một nút nhận được những gói tin mà được dành cho những nút khác. Phải nghe thừa những lưu thông không cần thiết, không giành cho mình có thể là một nhân tố chính gây tiêu hao năng lượng khi lưu lượng, tải truyền tăng và mật độ phân bố nút cao.

Nguyên nhân cuối cùng mà chúng ta xem xét là sự xử lý gói tin điều khiển. Sự gửi, nhận, và nghe những gói tin điều khiển cũng tiêu thụ năng lượng. Khi những gói điều khiển không trực tiếp chuyên chở dữ liệu, chúng cũng làm giảm *goodput*.

Một giao thức MAC thiết kế cho mạng cảm biến phải đạt được yêu cầu tiết kiệm năng lượng bởi việc điều khiển thành phần sóng vô tuyến để tránh hoặc giảm bớt tiêu phí năng lượng do những nguyên nhân trên. Việc tắt thành phần sóng vô tuyến khi nó chưa được cần đến là một chiến lược quan trọng cho việc tiết kiệm năng lượng. Một lược đồ quản lý năng lượng đầy đủ phải xem xét tất cả các nguồn làm tiêu phí năng lượng, không phải là chỉ là thành phần sóng vô tuyến.

3. Các giao thức MAC trong mạng cảm nhận không dây

3.1 CSMA

Các giao thức mà trong đó các trạm làm việc lắng nghe đường truyền trước khi đưa ra quyết định mình phải làm gì tương ứng với trạng thái đường truyền đó được gọi là các giao thức có "cảm nhận" đường truyền (carrier sense protocol). Cách thức hoạt động của CSMA như sau: lắng nghe kênh truyền, nếu thấy kênh truyền rỗi thì bắt đầu truyền khung, nếu thấy đường truyền bận thì trì hoãn lại việc gởi khung.

Thế nhưng trì hoãn việc gởi khung cho đến khi nào?

Có ba giải pháp:

- Theo dõi không kiên trì (Non-persistent CSMA): Nếu đường truyền bận, đợi trong một khoảng thời gian ngẫu nhiên rồi tiếp tục nghe lại đường truyền.

- Theo dõi kiên trì (persistent CSMA): Nếu đường truyền bận, tiếp tục nghe đến khi đường truyền rỗi rồi thì truyền gói tin với xác suất bằng 1.

- Theo dõi kiên trì với xác suất p (P-persistent CSMA): Nếu đường truyền bận, tiếp tục nghe đến khi đường truyền rỗi rồi thì truyền gói tin với xác suất bằng p.

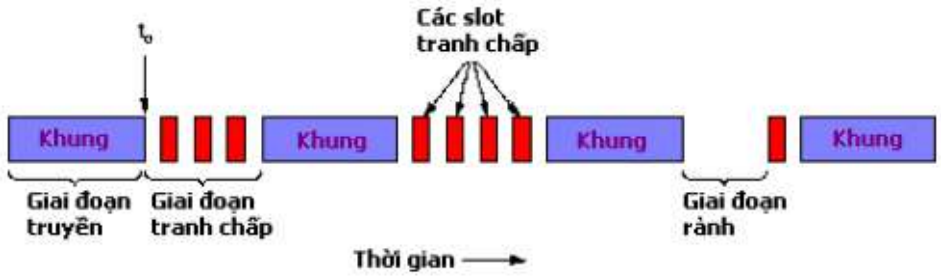
Dễ thấy rằng giao thức CSMA cho dù là theo dõi đường truyền kiên trì hay không kiên trì thì khả năng tránh xung đột vẫn tốt hơn là ALOHA. Tuy thế, xung đột vẫn có thể xảy ra trong CSMA.

Tình huống phát sinh như sau: khi một trạm vừa phát xong thì một trạm khác cũng phát sinh yêu cầu phát khung và bắt đầu nghe đường truyền. Nếu tín hiệu của trạm thứ nhất chưa đến trạm thứ hai, trạm thứ hai sẽ cho rằng đường truyền đang rảnh và bắt đầu phát khung. Như vậy xung đột sẽ xảy ra.

Hậu quả của xung đột là: khung bị mất và toàn bộ thời gian từ lúc xung đột xảy ra cho đến khi phát xong khung là lãng phí.

Bây giờ phát sinh vấn đề mới: các trạm có quan tâm theo dõi xem có xung đột xảy ra không và khi xung đột xảy ra thì các trạm sẽ làm gi?

CSMA/CD (CSMA với cơ chế theo dõi xung đột) về cơ bản là giống như CSMA: lắng nghe trước khi truyền. Tuy nhiên CSMA/CD có hai cải tiến quan trọng là: phát hiện xung đột và làm lại sau xung đột.

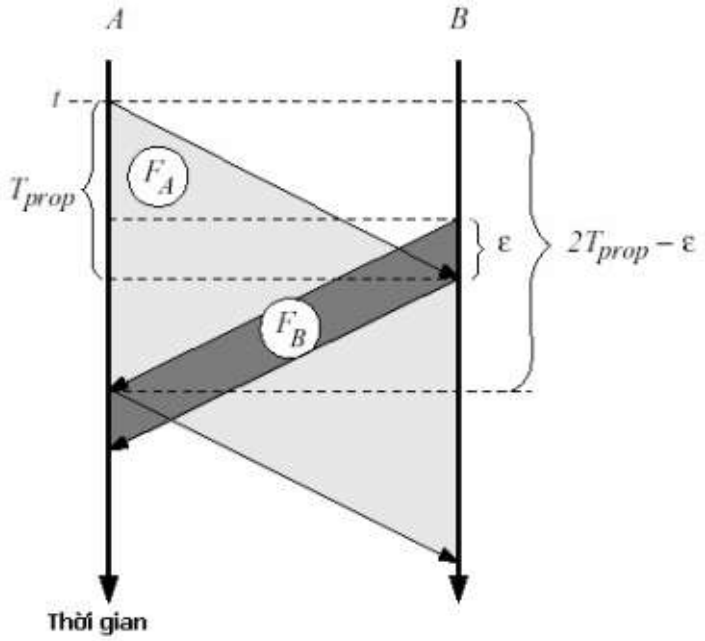


Hình 2.1. CSMA/CD có thể ở một trong ba trạng thái:
Tranh chấp, truyền, rảnh

Phát hiện xung đột: Trạm vừa truyền vừa tiếp tục dò xét đường truyền. Ngay sau khi xung đột được phát hiện thì trạm ngưng truyền, phát thêm một dãy nhồi (dãy nhồi này có tác dụng làm tăng cường thêm sự va chạm tín hiệu, giúp cho tất cả các trạm khác trong mạng thấy được sự xung đột), và bắt đầu làm lại sau xung đột.

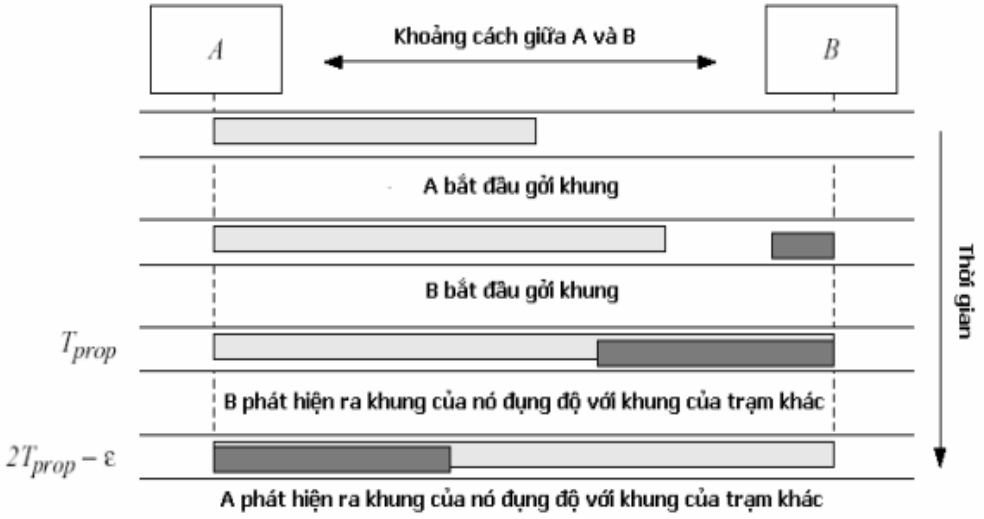
CSMA/CD, cũng giống như các giao thức trong LAN khác, sử dụng mô hình quan niệm như trong hình sau:

Tại thời điểm t_0, một trạm đã phát xong khung của nó. Bất kỳ trạm nào khác có khung cần truyền bây giờ có thể cố truyền thử. Nếu hai hoặc nhiều hơn các trạm làm như vậy cùng một lúc thì sẽ xảy ra xung đột. Xung đột có thể được phát hiện bằng cách theo dõi năng lượng hay độ rộng của xung của tín hiệu nhận được và đem so sánh với độ rộng của xung vừa truyền đi.



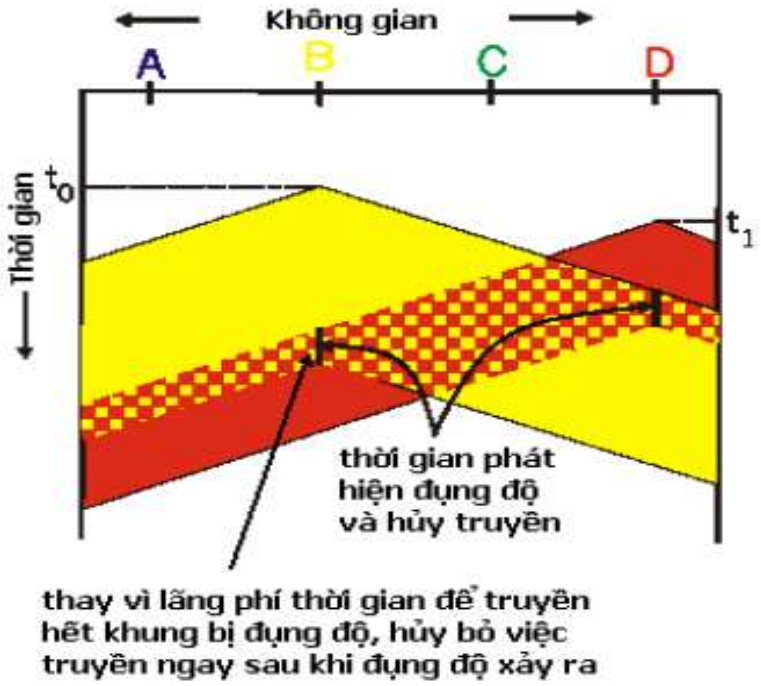
Hình 2.2. Thời gian cần thiết để truyền một khung

Bây giờ ta đặt ra câu hỏi: Sau khi truyền xong khung (hết giai đoạn truyền), trạm sẽ bỏ ra thời gian tối đa là bao lâu để biết được là khung của nó đã bị xung đột hoặc nó đã truyền thành công?



Hình 2.3. Phát hiện xung đột khi truyền tin

.



Hình 2.4. Xử lý khung xung đột

Hình 2.2 sẽ mô phỏng chi tiết về thời gian phát khung giữa hai trạm A và B ở hai đầu mút xa nhất trên đường truyền tải.

Việc hủy bỏ truyền khung ngay khi phát hiện có xung đột giúp tiết kiệm thời gian và băng thông, vì nếu cứ tiếp tục truyền khung đi nữa, khung đó vẫn hư và vẫn phải bị hủy bỏ.

Làm lại sau khi xung đột: Sau khi bị xung đột, trạm sẽ chạy một thuật toán gọi là back-off dùng để tính toán lại lượng thời gian nó phải chờ trước khi gởi lại khung. Lượng thời gian này phải là ngẫu nhiên để các trạm sau khi quay lại không bị xung đột với nhau nữa.

3.2. Sensor-MAC

S-MAC được giới thiệu bởi các tác giả: Wei Ye, Jonh Heidermann, Deborah Estrin tại Hội nghị INFOCOM lần thứ 21, năm 2002. Được xây dựng trên nền tảng của các giao thức cạnh tranh như 802.11, S-Mac cố gắng kế thừa sự linh hoạt, tính khả biến của giao thức trên nền cạnh tranh trong khi cải tiến tính hiệu quả sử dụng năng lượng trong mạng đa bước nhảy. S-MAC cố gắng giảm bớt tiêu thụ năng lượng từ tất cả các nguồn được xác định là nguyên nhân gây tiêu hao năng lượng, đó là: nghe khi rỗi *(idle listening)*, xung đột *(collision)*,

nghe thừa *(overhearing)* và xử lý thông tin điều khiển *(overhead)*. Để đạt được mục đích như thiết kế, S-MAC được thiết kế gồm có ba vấn đề chính: thực hiện chu kỳ thức - ngủ; tránh xung đột và nghe thừa; xử lý thông điệp.

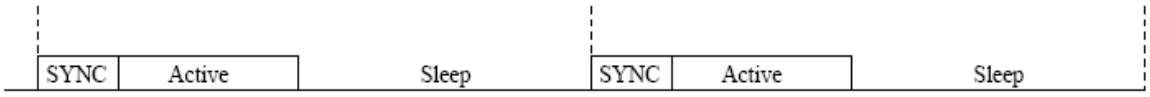
Thực hiện chu kỳ thức/ngủ

Trong những ứng dụng của mạng cảm biến, nút cảm biến thường ở trạng thái nhàn rỗi trong phần lớn thời gian nếu không xuất hiện sự kiện cảm biến. Thực tế tốc độ trao đổi dữ liệu rất thấp do vậy không cần thiết để các nút cảm biến ở trạng thái thức trong tất cả thời gian. S-MAC được thiết kế để giảm bớt thời gian thức bằng cách để cho nút cảm biến định kỳ chuyển sang trạng thái ngủ. Ví dụ, trong chu kỳ một giây, nút cảm biến ở trạng thái ngủ nửa giây và ở trạng thái nghe ở nửa giây còn lại thì chu trình hoạt động giảm bớt tới 50%. Như vậy có thể tiết kiệm được 50% năng lượng.

a, Lược đồ cơ bản

Mỗi nút cảm biến chuyển vào trạng thái "ngủ" trong một khoảng thời gian, sau đó tỉnh dậy và nghe xem liệu có nút nào muốn "nói chuyện" với nó. Trong thời gian ngủ, nút cảm biến tắt bộ phận thu phát vô tuyến và đặt thời gian để quay về trạng thái thức.

Khoảng thời gian cho việc thức và ngủ có thể được lựa chọn theo những ứng dụng khác nhau.



Hình 2.5. Lược đồ S-MAC

Lược đồ trên yêu cầu có định kỳ sự đồng bộ giữa các nút cảm biến trong vùng tránh sai lệch thời gian. Tất cả các nút cảm biến đều tự do lập lịch cho mình chu kỳ thức/ngủ. Tuy nhiên, để giảm bớt phải xử lý những gói tin điều khiển, tốt hơn là để cho các nút trong vùng đồng bộ cùng nhau. Có nghĩa là chúng thức cùng lúc và chuyển sang trạng thái ngủ cùng lúc. Nhưng cũng cần chú ý trong một mạng đa bước nhảy không phải tất cả các nút lân cận có thể

đồng bộ hóa cùng nhau. Hai nút lân cận A và B có thể có lịch khác nhau vì chúng tiến hành đồng bộ với những nút khác nhau, C và D (Hình 2.6).

Hình 2.6. Đồng bộ giữa các nút. Hai nút lân cận A, B có lịch khác nhau vì A đồng bộ với C, B đồng bộ với D

Các nút cảm biến trao đổi với nhau thông tin lịch làm việc của chúng bằng cách phát quảng bá cho tất cả các nút lân cận hiện thời. Điều này bảo đảm rằng tất cả các nút trong vùng vẫn có thể nói chuyện được với nhau dù chúng có lịch làm việc khác nhau. Ví dụ trong Hình 2.6, nếu nút A muốn nói chuyện với nút B, nó chỉ cần đợi cho đến khi B ở trạng thái thức. Nếu có nhiều nút trong vùng lân cận muốn nói chuyện với một nút, thì chúng cần tiến hành cạnh tranh chiếm đường truyền khi nút nhận ở trạng thái thức, sử dụng gói tin RTS *(Request to Send)* và CTS *(Clear to Send)*. Nút nào gửi gói tin RTS ra trước sẽ giành quyền truy nhập và nút nhận sẽ trả lời với một gói CTS. Sau đó chúng bắt đầu sự truyền dữ liệu, lúc này chúng không tuân theo lịch làm việc trước đó của chúng cho đến khi chúng kết thúc truyền dữ liệu.

Mặt trái của lược đồ là sự gia tăng độ trễ do duy trì chu kỳ ngủ *(sleep)* của mỗi nút. Hơn nữa, độ trễ có thể tích lũy qua mỗi chặng (*hop)*, nên yêu cầu giới hạn độ trễ của ứng dụng tạo ra giới hạn thời gian ngủ trong chu kỳ làm việc của các nút cảm biến.

b, Tiến trình lựa chọn và duy trì lịch làm việc

Trước khi mỗi nút bắt đầu chu kỳ thức/ngủ, nó cần phải chọn một lịch biểu làm việc (khi nào thức, khi nào ngủ) và trao đổi lịch này với các nút lân cận. Mỗi nút duy trì một bảng lưu giữ tất cả các thời gian biểu của các nút lân cận mà nó biết.

Rất hiếm khi xảy ra các nút phải duy trì nhiều thời gian biểu. Các nút sẽ cố gắng chọn một thời gian biểu đã tồn tại trước khi tự chọn cho mình một thời gian biểu độc lập. Mặt khác, xảy ra trường hợp các nút lân cận thất bại trong

việc khám phá, phát hiện ra nhau tại thời điểm ban đầu do xung đột khi quảng bá thời gian biểu, thì chúng vẫn có thể tìm thấy nhau trong chu kỳ kế tiếp.

Một tùy chọn khác để cho những nút trên vùng biên chấp nhận duy nhất một thời gian biểu là chấp nhận cái đến trước tiên. Khi nó biết thời gian biểu khác mà một số nút lân cận của nó theo, nó có thể vẫn còn nói chuyện . Tuy nhiên, với những gói quảng bá, nó cần gửi hai lần với hai thời gian biểu khác nhau. Ưu điểm của phương pháp này là các nút nằm trong vùng biên sẽ có cùng chu kỳ nghe ngủ với những nút khác.

c, Thực hiện đồng bộ

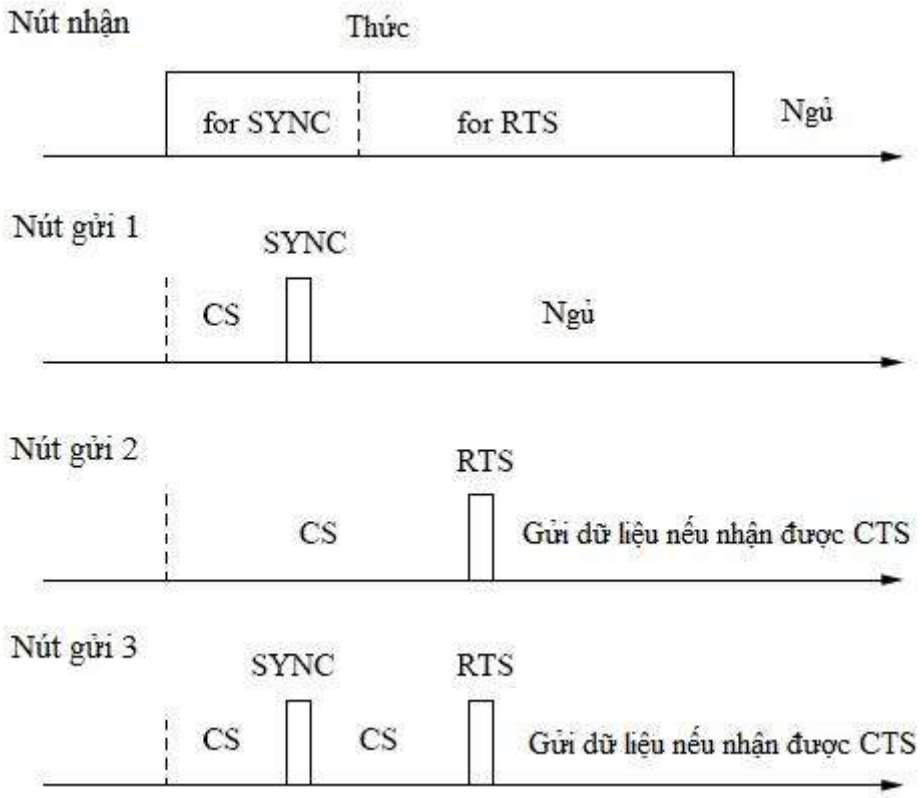
Lược đồ thức/ngủ yêu cầu sự đồng bộ giữa những nút trong vùng lân cận. Việc các nút trong vùng lân cận định kỳ cập nhật lẫn nhau thời gian biểu của chúng là cần thiết để ngăn ngừa sự sai lệch thời điểm của chu kỳ nghe/ngủ.

Việc cập nhật thời gian biểu được thực hiện bằng trao đổi gói tin đồng bộ SYNC. Gói tin SYNC rất ngắn, và bao gồm địa chỉ của nút gửi và thời điểm chuyển sang trạng thái ngủ tiếp theo của nó.

Để một nút nhận được cả những gói đồng bộ lẫn những gói dữ liệu, chúng ta chia khoảng thức *(active time)* của nó thành hai phần. Phần đầu tiên để nhận những gói tin đồng bộ, phần hai để nhận những gói RTS (Hình 2.7). Mỗi phần được chia tiếp thành nhiều khe thời gian cho những nút gửi để thực hiện cảm nhận sóng mang. Ví dụ, nếu một nút gửi muốn gửi một gói tin đồng bộ thì nó khởi động cảm nhận sóng mang khi nút nhận bắt đầu nghe. Nó ngẫu nhiên lựa chọn một khe thời gian để kết thúc cảm nhận sóng mang. Nếu nó không phát hiện ra bất kỳ sự truyền nào vào khoảng cuối khe, thì nó chiếm được đường truyền và bắt đầu gửi gói tin đồng bộ của nó ở tại thời điểm ấy. Việc thực hiện truyền gói dữ liệu cũng được thực hiện tương tự.

Hình 2.7 cũng thể hiện mối quan hệ định thời của ba trường hợp có thể khi một nút gửi thực hiện truyền tới một nút nhận. CS là cảm ứng sóng mang. Trong lược đồ, Nút gửi 1 chỉ gửi một gói tin đồng bộ SYNC. Nút gửi 2 chỉ muốn gửi dữ liệu. Nút gửi 3 gửi một gói tin đồng bộ và một gói tin RTS.

Mỗi nút định kỳ quảng bá những gói tin đồng bộ tới các lân cận của nó kể cả khi nó không có nút đồng bộ theo. Điều này cho phép một nút mới gia nhập nhóm lân cận đã hình thành trước đó. Nút mới thực hiện thủ tục để chọn một thời gian biểu có sẵn làm thời gian biểu của nó. Quãng thời gian nghe đủ dài để nó có khả năng học và theo một thời gian biểu có sẵn trước khi nó tự chọn cho mình một thời gian biểu độc lập.



Hình 2.7. Quan hệ định thời giữa nút nhận và các nút gửi.

. Tránh xung đột và nghe thừa

Tránh xung đột là một nhiệm vụ cơ bản của giao thức MAC. S-MAC sử dụng một lược đồ tránh xung đột trên nền cạnh tranh. Khi một nút phát đi một gói tin, gói tin đó được thu bởi tất cả các nút lân cận của nó mặc dù chỉ một trong số chúng là nút nhận, đó chính là nghe thừa. Phải nghe thừa làm cho giao thức MAC trên nền cạnh tranh kém hiệu quả về tiết kiệm năng lượng hơn so với những giao thức TDMA, vậy nên nó cần phải tránh.

a, Tránh xung đột

Khi nhiều nút có nhu cầu gửi số liệu vào cùng một thời điểm, chúng cần cạnh tranh để quyết định một nút được quyền gửi (chiếm đường truyền). Trong số những giao thức cạnh tranh, 802.11 thực hiện rất tốt việc tránh xung đột. S-MAC sử dụng các kỹ thuật như chuẩn 802.11, bao gồm cảm nhận sóng mang vật lý, cảm nhận sóng mang ảo lẫn thực hiện trao đổi RTS/CTS.

Có một trường độ dài phát (*duration field)* trong mỗi gói tin được truyền đi để chỉ rằng việc truyền này sẽ duy trì trong thời gian bao lâu. Như vậy nếu một nút nhận được một gói tin dành cho nút khác, thì nó biết việc nó phải giữ yên lặng bao lâu. Nút ghi giá trị này trong một biến gọi là vectơ thời gian chiếm giữ mạng (Network allocation Vector - NAV) và đặt một đồng hồ tính giờ cho nó. Vào mọi thời điểm khi đồng hồ NAV hoạt động, nút cảm biến tuần tự giảm giá trị của NAV cho đến khi nó về giá trị 0. Khi một nút có dữ liệu để gửi, đầu tiên nó kiểm tra đồng hồ NAV. Nếu giá trị của NAV khác 0, thì nút xác định rằng đường truyền bận và sẽ không thực hiện phát dữ liệu. Kỹ thuật này được gọi là cảm nhận sóng mang ảo *(Vitual Carrier Sense)*.

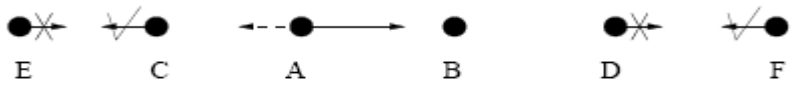
Cảm nhận sóng mang vật lý được thực hiện ở tại lớp vật lý bằng cách thực hiện nghe kênh để truyền. Thời gian ngẫu nhiên cho việc cảm nhận sóng mang rất quan trọng cho viêc tránh xung đột. Đường truyền chỉ được xác định là rỗi nếu cả cảm nhận sóng mang vật lý lẫn cảm nhận sóng mang ảo đều xác định đường truyền rỗi.

Tất cả các nút gửi thực hiện cảm nhận sóng mang trước khi bắt đầu phát dữ liệu. Nếu một nút thất bại trong việc thăm dò đường truyền, thì nó chuyển sang trạng thái ngủ và thức giấc tại thời điểm nút nhận ở trạng thái nghe và đường truyền rỗi trở lại. Những gói tin quảng bá được gửi mà không sử dụng kỹ thuật RTS/CTS. Những gói tin Unicast sẽ theo tuần tự RTS/CTS/Data/ACK giữa nút gửi và nút nhận.

b, Tránh nghe thừa

Ở chuẩn 802.11, mỗi nút duy trì trạng thái nghe cho việc truyền tới tất cả các nút lân cận của nó để thực hiện có hiệu quả việc cảm nhận sóng mang ảo. Kết quả là mỗi nút phải nghe thừa nhiều gói không gửi cho nó. Đây là một trong những nguyên nhân chính cho việc tiêu phí năng lượng không cần thiết, đặc biệt khi mật độ nút lớn và lưu lượng mạng tăng.

S-MAC được thiết kế với mục tiêu cố gắng tránh nghe thừa bằng cách để cho những nút có khả năng gây nhiễu không tham gia vào quá trình truyền phát dữ liệu, chuyển sang trạng thái ngủ sau khi chúng nhận được một gói RTS hoặc CTS. Khi những gói dữ liệu luôn dài hơn gói tin điều khiển, cách tiếp cận là ngăn cản các nút lân cận nghe thừa những gói dữ liệu dài và sử dụng gói tin ACK theo sau. Phần tiếp theo sẽ mô tả cách truyền có hiệu quả một gói tin dài kết hợp tránh nghe thừa.



Hình 2.8. Thực hiện tránh nghe thừa. Nút nào nên chuyển tới trạng thái ngủ.

Trong Hình 2.8, nút A, B, C, D, E, Và F hình thành một mạng đa bước nhảy mà từng nút chỉ có thể nghe thông tin truyền từ lân cận hiện thời của nó. Giả thiết nút A đang truyền một gói dữ liệu tới nút B. Câu hỏi đặt ra những nút nào phải chuyển sang trạng thái ngủ.

Xung đột dễ xảy ra ở nút nhận, nút D cần phải ngủ vì sự truyền của nó ảnh hưởng tới sự tiếp nhận tín hiệu của B. Cũng dễ để nhận ra nút E và nút F không phát sinh nhiễu, vì vậy chúng không cần phải ngủ. Nút C có nên đi ngủ hay không? C là cách hai bước tới B, và sự truyền của nó không gây nhiễu tới sự tiếp nhận của B, như vậy nó tự do được phép truyền tới lân cận của nó, ví dụ như E. Tuy nhiên, C không thể nhận bất kỳ sự trả lời nào từ E, vì sự truyền của E xung đột với sự truyền của A tại nút C. Như vậy sự truyền của C đơn giản là một sự tiêu phí năng lượng. Tóm lại, tất cả lân cận tức thời của cả nút gửi và nút

nhận cần phải chuyển trạng thái ngủ khi chúng nghe thấy gói RTS hoặc CTS cho đến khi sự truyền hiện thời kết thúc.

Mỗi nút duy trì NAV để chỉ báo hoạt động trong khu lân cận của nó. Khi một nút nhận một gói dành cho tới những nút khác, nó cập nhật NAV của nó tại trường *duration* trong định dạng gói tin. Một giá trị NAV lớn hơn 0 chỉ báo rằng có một nút đang gửi số liệu trong khu vực lân cận của nó. Giá trị NAV giảm dần theo thời gian. Như vậy một nút cần phải ở trạng thái ngủ để tránh nghe thừa khi giá trị NAV của nó khác 0.

Xử lý thông điệp

Truyền dữ liệu dài trong một gói tin thì chi phí cho việc truyền lại khi chỉ có một vài bít lỗi trong lần truyền đầu tiên là rất cao. Tuy nhiên, nếu chúng ta chia nhỏ thông điệp vào trong nhiều gói nhỏ độc lập, chúng ta phải xử lý quá nhiều gói tin điều khiển do vậy độ trễ truyền sẽ tăng.

S-MAC xử lý vấn đề trên bằng cách chia nhỏ thông điệp dài thành nhiều phân đoạn nhỏ và truyền chúng trong một cụm (burst) nhưng chỉ sử dụng một gói tin RTS và một gói tin CTS. Chúng chiếm dụng đường truyền truyền tất cả các đoạn. Mỗi lần một đoạn dữ liệu được truyền, nơi gửi đợi một xác nhận ACK từ nơi nhận. Nếu nó không nhận được ACK, nó sẽ mở rộng thời gian chiếm dụng đường truyền cho đủ một phân đoạn nữa, và truyền lại ngay phân đoạn dữ liệu hiện thời.

Như đã biết, tất cả các gói tin đều có trường thời gian, bây giờ nó là thời gian cần cho sự phát tất cả các phân đoạn dữ liệu còn lại và những gói ACK. Nếu một nút trong vùng lân cận nhận được một gói RTS hoặc CTS, thì nó sẽ chuyển sang trạng thái ngủ trong khoảng thời gian truyền tất cả các phân đoạn.

Mục đích của việc sử dụng ACK sau mỗi phân đoạn dữ liệu nhằm ngăn ngừa vấn đề nút ẩn (hidden terminal). Có thể một nút lân cận thức dậy hoặc một nút mới gia nhập vùng lân cận trong quá trình truyền. Nếu nút chỉ là lân cận của nút nhận nhưng không phải nút gửi, thì nó sẽ không nghe thấy các phân đoạn dữ liệu đang được phát từ nút gửi. Nếu nút nhận không gửi ACK thường xuyên, thì

nút mới có thể gây nhiễu vì cảm nhận sóng mang trong việc thăm dò đường truyền sẽ thông báo đường truyền rỗi. Nếu nó khởi động tiến trình phát, thì quá trình truyền hiện thời sẽ bị hỏng ở tại nút nhận.

Mỗi phân đoạn dữ liệu và gói tin ACK cũng có trường *thời gian*. Bằng cách này, nếu một nút tỉnh dậy hoặc một nút mới gia nhập trong quá trình truyền, thì nó chuyển sang trạng thái ngủ bất kể nó là lân cận của nút gửi hay nút nhận. Ví dụ, giả sử một nút lân cận nhận được một RTS của nút gửi hoặc một CTS từ nút nhận, nó sẽ chuyển trạng thái ngủ trong toàn bộ thời gian được cung cấp trong thông điệp. Nếu bên gửi mở rộng thời gian truyền do mất phân đoạn dữ liệu hoặc do lỗi, vì ngủ nên các nút lân cận không ý thức được sự mở rộng này ngay lập tức. Tuy nhiên, các nút sẽ biết được điều này từ những phân đoạn mở rộng hoặc những gói tin ACK khi nó tỉnh dậy.

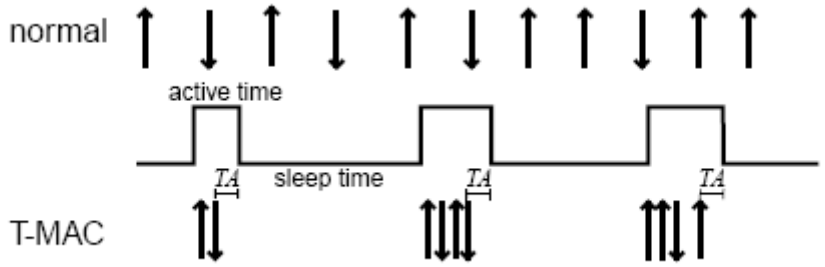
3.3.. Time out-MAC

Mặc dù thực hiện giảm tiêu hao năng lượng bằng việc giảm thời gian chờ nghe qua giải pháp thực hiện chu trình *thức/ngủ* cố định, nhưng giải pháp này của S-MAC chưa đạt hiệu quả tối ưu. S-MAC có hai tham số quan trọng: độ lớn của khung thời gian *(frame time)* và độ dài thời gian thức *(active time)*. Độ lớn khung thời gian bị giới hạn bởi yêu cầu về độ trễ cho phép và độ lớn bộ đệm. Độ lớn thời gian thức phụ thuộc chủ yếu trên tốc độ phát sinh thông điệp: nó phải đủ lớn để nút cảm biến có thể phát đi tất cả các thông điệp của nó trong khoảng thời gian thức. Trong khi yêu cầu độ trễ và không gian bộ đệm nói chung là cố định thì tốc độ phát sinh thông điệp thường thay đổi. Để đảm bảo tất cả các thông điệp được phát như mong muốn, nút cảm biến phải được cài đặt một thời gian thức sao cho có thể xử lý ở mức thông lượng cao nhất. Nhưng khi thông lượng xuống thấp thì thời gian thức sẽ không được sử dụng tối ưu và do đó năng lượng sẽ bị lãng phí do vấn đề nghe khi rỗi *(idle listening)*.

Giao thức điều khiển truy nhập T-MAC (Timeout-MAC) do hai tác giả Tijs van Dam và Koen Langendoen, khoa Công nghệ thông tin và các hệ thống, Trường đại học công nghệ Delft, Hà Lan, giới thiệu tại Hội nghị quốc tế về các

hệ thống mạng cảm biến nhúng lần thứ nhất tại Los Angeles, Mỹ, năm 2003 (Sensys'03), là sự cải tiến S-MAC để khắc phục nhược điểm trên. Ý tưởng mới của giao thức T-MAC là giảm bớt thời gian nghe khi rỗi bằng việc truyền tất cả các thông điệp trong những cụm *(burst)* có độ dài thay đổi tùy theo, và thực hiện ngủ giữa các cụm, xác định một cách mềm dẻo độ dài tối ưu thời gian thức theo sự thay đổi của lưu lượng đường truyền.

Những vấn đề cơ bản



Hình 2.9. Lược đồ cơ bản T-MAC với thời gian thức thay đổi

Hình 2.9 cho thấy lược đồ cơ bản của giao thức T-MAC. Mỗi nút định kỳ tỉnh dậy liên lạc các nút lân cận, sau đó ngủ tiếp cho đến khi khung tiếp theo. Trong lúc đó, những thông điệp mới được đưa vào hàng đợi. T-MAC cũng sử dụng kỹ thuật *RTS, CTS, Data, ACK* để tránh xung đột và truyền số liệu tin cậy.

Một nút sẽ được đặt ở chế độ nghe và sẵn sàng thực hiện truyền số liệu khi nó đang ở trong trạng thái thức. Trạng thái thức sẽ kết thúc khi không có một sự kiện kích hoạt *(activation event)* nào xuất hiện một khoảng thời gian TA. Một sự kiện kích hoạt là:

+ Sự kết thúc một khung thời gian theo định kỳ.

+ Sự tiếp nhận bất kỳ dữ liệu nào trên sóng vô tuyến.

+ Sự xuất hiện sự kiện cảm biến được phát hiện qua thành phần vô tuyến.

+ Sự kết thúc truyền dữ liệu của một nút có sở hữu gói dữ liệu hoặc sự biên nhận ACK;

+ thông tin về sự kết thúc trao đổi dữ liệu của các nút lân cận qua nhận được các gói RTS, CTS.

Thông số TA xác định thời gian tối thiểu cho việc thức chờ nghe trên một khung thời gian.

Lược đồ timeout chuyển tất cả các giao dịch vào một cụm tại điểm bắt đầu của khung. Khi đó những thông điệp giữa các thời gian hoạt động phải được đưa vào bộ đệm. Độ lớn của bộ đệm xác định cận trên của độ lớn khung thời gian cực đại.

Phân nhóm và đồng bộ

Đồng bộ khung thời gian được thực hiện qua sự hình thành phân nhóm ảo như được mô tả trong giao thức S-MAC. Khi một nút cảm biến bắt đầu quá trình hoạt động của mình, nó bắt đầu bằng việc đợi và nghe. Nếu nó không nghe thấy gì trong một khoảng thời gian nhất định, thì nó tự chọn cho mình một lịch làm việc và truyền một gói tin đồng bộ SYNC chứa đựng thời gian khởi tạo của khung tiếp theo trong lịch làm việc. Nếu nút cảm biến trong thời gian khởi động nghe thấy một gói tin đồng bộ từ nút khác, thì nó sẽ theo lịch làm việc trong gói tin đồng bộ đó và quảng bá gói tin đồng bộ tương ứng của chính mình.

Các nút cảm biến thực hiện phát lại ngay gói tin đồng bộ của chúng. Chúng thực hiện nghe đầy đủ một khung một cách không thường xuyên, vì vậy chúng có thể phát hiện ra sự tồn tại của những thời gian biểu khác nhau. Điều này này cho phép các nút mới hoặc các nút di động có thể được chấp nhận gia nhập nhóm đã tồn tại trước đó.

Nếu một nút đã có một thời gian biểu nhưng lại nghe được từ gói tin đồng bộ một thời gian biểu khác (từ nút khác), thì nó chấp nhận cả hai và thực hiện phát một gói tin đồng bộ chứa thời gian biểu của mình để cho các nút khác biết có sự tồn tại thời gian biểu đó. Việc chấp nhận cả hai thời gian biểu làm việc có nghĩa rằng nút sẽ có những sự kiện kích hoạt ở tại thời điểm bắt đầu của cả hai khung.

Muốn truyền dữ liệu, các nút cảm biến phải khởi động tại điểm bắt đầu khoảng thời gian thức quy định trong lịch biểu của chúng. Tại thời điểm đó, cả các nút lân cận có cùng thời gian biểu, và các lân cận mà đã chấp nhận thời gian biểu như sự bổ sung đều ở trạng thái thức. Nếu nó thực hiện ở điểm bắt đầu của một khung của nút lân cận, thì có thể nó phát trong khi lân cận của nó vẫn đang

trong trạng thái ngủ. Với lược đồ này làm có thể thực hiện quảng bá mà chỉ cần phát một lần duy nhất.

Lược đồ đồng bộ được mô tả ở trên được gọi là quá trình phân nhóm ảo, thúc đẩy các nút hình thành nhóm với cùng thời gian biểu mà không bắt buộc thời gian biểu này áp dụng tới tất cả các nút trong mạng. Nó cho phép thực hiện quảng bá có hiệu quả, và tránh sự duy trì thông tin các nút lân cận.

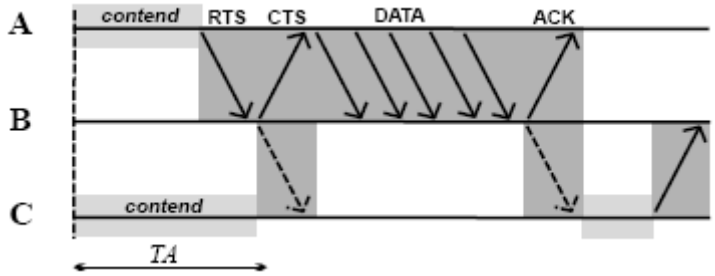
Thực hiện gửi RTS và chọn TA trong T-MAC

T-MAC cần bổ sung một số đặc tính so với S-MAC để thực hiện sự điều chỉnh tối ưu thời gian thức.

a, Khoảng cạnh tranh cố định (Fixed contention interval)

Trong những giao thức trên nền cạnh tranh, như IEEE 802.11, các nút đợi ngẫu nhiên một khoảng thời gian nhất định, gọi là khoảng thời gian cạnh tranh, sau khi phát hiện có xung đột. Chỉ khi đường truyền rỗi trong thời gian ấy chúng mới khởi động lại sự truyền. Thông thường, một lược đồ back-off được sử dụng: khoảng thời gian cạnh tranh tăng thêm khi lưu lượng đường truyền tăng. Lược đồ back-off giảm bớt xác suất xảy ra xung đột khi tải tăng cao, trong khi tối thiểu độ trễ khi tải thấp.

Trong giao thức T-MAC, mỗi nút truyền các thông điệp trong hàng đợi của nó vào một cụm tại điểm bắt đầu của khung. Trong thời gian truyền cụm này, đường truyền là bão hòa: những thông điệp được truyền ở tốc độ cực đại. Mọi nút đều muốn giành quyền truy nhập đường truyền mỗi khi nó gửi một gói tin RTS. Khoảng cạnh tranh ngày càng tăng thì lại không có ích khi tải phần lớn đã cao và không thay đổi. Bởi vậy, sự truyền RTS trong T-MAC bắt đầu bởi việc đợi và nghe một khoảng thời gian ngẫu nhiên trong phạm vi một khoảng cạnh tranh cố định. Khoảng này được điều chỉnh phù hợp với tải cực đại. Khoảng thời gian cạnh tranh luôn luôn được sử dụng dù không có xung đột.



Hình 2.10. Lược đồ trao đổi dữ liệu cơ bản. Nút C nghe được CTS từ nút B và sẽ không làm phiền giao tiếp giữa A và B. TA phải đủ dài để C có thể nghe được phần đầu CTS

b, Thử lại phát lại RTS

Khi một nút phát một gói tin RTS, nhưng không nhận được trở lại một CTS, có thể một trong ba trường hợp xảy ra:

+ Nút nhận không nghe được RTS vì xung đột, hoặc

+ Nút nhận bị ngăn cản trả lời vì nghe được RTS hoặc CTS, hoặc

+ Nút nhận đang ngủ.

Khi nút gửi không nhận câu trả lời trong khoảng TA, nó có thể chuyển sang trạng thái ngủ. Tuy nhiên, điều đó có thể không hợp lý trong những trường hợp 1 và 2: sẽ xảy ra hiện tượng nút muốn gửi chuyển sang trạng thái ngủ trong khi nút nhận vẫn thức. Khi trường hợp này xảy ra thậm chí ở ngay tại thông báo đầu tiên của khung, thông lượng giảm đáng kể. Bởi vậy, một nút cần phải cố gắng gửi lại RTS nếu nó không nhận được câu trả lời. Nếu không có còn sự trả lời sau khi thử lại, nó cần phải từ bỏ ý định truyền và sang trạng thái ngủ.

c, Xác định khoảng TA

Một nút không nên chuyển sang trạng thái ngủ trong khi các nút lân cận của nó vẫn còn trao đổi số liệu, một khi nút lân cận đó có thể là nút nhận của một thông báo kế tiếp. Khi bắt đầu nhận được gói tin RTS hoặc CTS của một nút lân cận cũng đủ thực hiện một tác vụ kích hoạt khởi tạo khoảng TA.

Không nằm trong vùng lân cận, nên một nút sẽ không nhận được thông điệp RTS từ một nút mà khởi tạo truyền thông với lân cận của nó. Khoảng TA phải

đủ dài để nhận ít nhất bắt đầu của gói CTS (Hình 2.10). Sự quan sát này cho chúng ta một cận dưới của độ dài khoảng TA:

$$TA > C + R + T$$

Ở đây C là chiều dài khoảng cạnh tranh, R là độ dài một gói RTS, và T là thời gian turn-around (khoảng thời gian ngắn giữa kết thúc của gói RTS và sự bắt đầu của gói CTS). Chọn thời gian TA lớn sẽ làm tăng sự tiêu phí năng lượng.

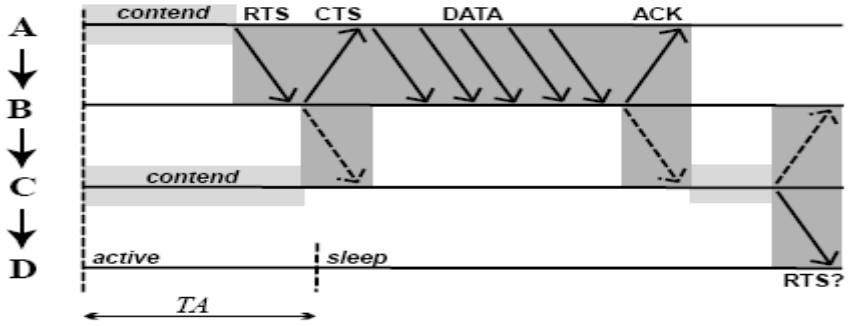
Tránh nghe thừa

Giao thức S-MAC đưa ra ý tưởng nút sẽ sang trạng thái ngủ sau khi nghe được một gói tin RTS hoặc CTS dành cho cho nút khác. Khi đó nút bị ngăn cản việc gửi dữ liệu trong thời gian đó, nó không thể tham gia bất kỳ truyền thông nào và tốt nhất là tắt bộ phận thu phát vô tuyến của nó để tiết kiệm năng lượng.

Tránh nghe thừa là một tùy chọn trong giao thức T-MAC để giảm năng lượng tiêu thụ. Tuy nhiên, chúng sẽ làm xung đột do thông tin điều khiển *(overhead collision)* cao hơn: một nút có thể không nhận được gói tin RTS và CTS trong khi ngủ và làm phiền giao tiếp nào đó khi nó tỉnh dậy trở lại. Do vậy, lưu lượng cực đại giảm bớt. Mặc dầu việc tránh nghe thừa sẽ tiết kiệm điện năng nhưng nó không được sử dụng khi muốn đạt băng thông cực đại.

Truyền thông bất đối xứng

Do lưu lượng trên mạng cảm biến phần lớn là đẳng hướng, như dạng truyền thông từ nhiều nút tới nút gốc (Notes-to-Sink), nên T-MAC xuất hiện hiện tượng làm giảm thông lượng cực đại của mạng. Hiện tượng này được mô tả như sau (Hình 2.11): Các nút từ A đến D hình thành một tế bào với các lân cận của nó. Các thông điệp di chuyển từ trên xuống dưới, như vậy nút A chỉ phát tới B, B chỉ phát tới C, và C chỉ phát tới D.



Hình 2.11. Hiện tượng ngủ sớm. D đi ngủ trước khi C gửi một RTS cho nó

Khi nút C muốn phát dữ liệu tới nút D, nó phải tiến hành cạnh tranh, thăm dò đường truyền để giành quyền phát. Việc thăm dò có thể không tiến hành được vì trước đó nó nhận một thông điệp RTS từ nút B, hoặc nghe được thông điệp CTS từ nút B trả lời tới nút A.

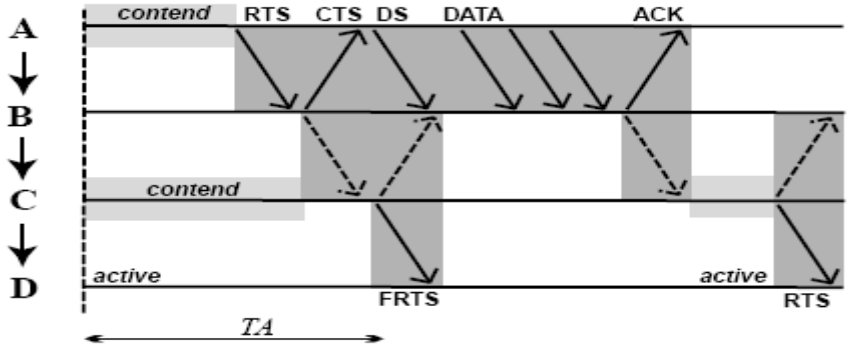
Khi C không tiến hành được việc thăm dò đường truyền do nhận được thông điệp RTS từ nút B, nó sẽ trả lời B một thông điệp CTS, D sẽ nghe được thông điệp này và đặt lịch chuyển sang trạng thái thức khi truyền thông giữa C và B kết thúc. Tuy nhiên, nếu C không tiến hành được là do nghe được thông điệp CTS từ B trả lời A (Hình 2.11), nó phải giữ im lặng. Ở trường hợp này, do D không biết truyền thông giữa A và B, không nhận được thông điệp muốn truyền dữ liệu từ C, nó sẽ chuyển sang trạng thái ngủ khi thời gian thức theo lịch kết. Chỉ ở tại điểm bắt đầu của khung tiếp theo, nút C mới có cơ hội để thực hiện thăm dò và tiến hành trao đổi dữ liệu với nút D.

Những vấn đề quan sát được ở trên được gọi là *hiện tượng ngủ sớm (early sleeping problem)*, tức là một nút chuyển sang trạng thái ngủ khi một nút lân cận vẫn thức và muốn trao đổi dữ liệu với nó. Trong dạng truyền thông từ nút đến nút gốc, ngủ sớm làm giảm thông lượng có thể của T-MAC tới ít hơn một nửa thông lượng cực đại của những giao thức truyền thống, hoặc so với S-MAC. Có hai giải pháp để khắc phục hiện tượng trên.

Gửi sớm RTS (Future request to send)

Ý tưởng của giải pháp gửi sớm RTS là sẽ để cho nút nhận tiềm năng (nút D) biết được có một nút muốn gửi dữ liệu cho nó, nhưng đang trong tình trạng

phải giữ im lặng để không làm ảnh hưởng đến giao tiếp khác. Khi một nút nghe được một thông điệp CTS dành cho cho nút khác, nó ngay lập tức gửi một gói FRTS (nút C trong Hình 2.12). Gói FRTS chứa thông tin về độ dài của khối dữ liệu truyền thông lấy được trong thông điệp CTS.



Hình 2.12. Thực hiện gửi sớm RTS. Gói tin FRTS giữ D thức

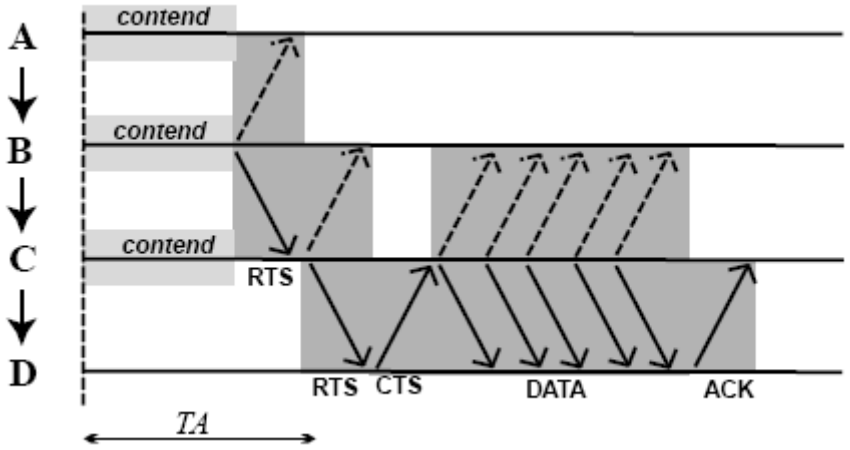
Một nút nhận được gói tin FRTS thì nó biết rằng trong khoảng thời gian *t* tiếp theo nó sẽ nhận được một thông điệp RTS, do vậy phải lập lịch thức trước thời gian ấy. Thông tin thời gian *t* được lấy trong thông điệp FRTS.

Để thông điệp FRTS (do C phát) không gây nhiễu dữ liệu trao đổi (giữa A và B) theo sau thông điệp CTS, dữ liệu phải được hoãn lại khoảng thời gian truyền FRTS. Để không mất kênh truyền, nút khởi tạo RTS ban đầu (Nút A) truyền một gói tin DS (Data-Send). Sau gói DS, nó ngay lập tức gửi dữ liệu bình thường. FRTS có cùng kích thước với DS, nó sẽ chỉ xung đột với gói DS mà không phải với gói dữ liệu. Gói DS bị mất, nhưng không có vấn đề gì vì nó không chứa đựng thông tin.

Với giải pháp FRTS, độ dài của khoảng thời gian TA phải được tăng thêm một khoảng bằng độ dài thông điệp CTS. Việc thực hiện giải pháp gửi sớm RTS sẽ làm tăng thông lượng cực đại trong truyền thông đẳng hướng. Tuy nhiên, vì có DS và FRTS, mức tiêu thụ năng lượng cũng tăng thêm. Cũng có thể sử dụng kỹ thuật FRTS trong các dạng truyền thông khác nhưng chỉ khi muốn tăng thông lượng một cách chính đáng. Vì khi tải ở mức thấp thì tốc độ trao đổi dữ liệu cũng thấp do phải gia tăng xử lý thông tin điều khiển.

Thực hiện ưu tiên gửi khi bộ đệm đầy (taking priority on full buffers)

Khi nào bộ đệm truyền/định tuyến của một nút gần đầy, thì việc gửi sẽ hợp lý hơn là tiếp tục nhận. Khi một nút nhận được RTS dành cho nó, ngay lập tức nó gửi gói RTS của chính mình cho nút khác, thay vì việc trả lời với một CTS như bình thường.



Hình 2.13. Thực hiện ưu tiên gửi khi bộ đệm đầy

Giải pháp này có hai hiệu quả, trước hết khi nút C khi trả lời B bằng thông điệp RTS khi bộ đệm của nó đầy, một mặt nó trả lời B rằng nó không muốn nhận, mặt khác đồng thời nó cũng thông báo cho D là nó muốn gửi dữ liệu. Như vậy xác suất mà vấn để ngủ sớm xảy ra sẽ thấp hơn. Hai là, *thực hiện ưu tiên gửi khi bộ đệm đầy* hình thành một giới hạn điều khiển luồng trong mạng có lợi cho những dạng truyền thông từ nút tới nút gốc. Trong Hình 2.13, nút B bị ngăn gửi cho đến khi nút C có đủ không gian bộ đệm.

Chương 3 - PHẦN MỀM MÔ PHỎNG MẠNG OMNET++

3.1. OMNET++

3.1.1. Giới thiệu

OMNeT++ là viết tắt của cụm từ Objective Modular Network Testbed in C++. OMNeT++ là một ứng dụng cung cấp cho người sử dụng môi trường để tiến hành mô phỏng hoạt động của mạng. Mục đích chính của ứng dụng là mô phỏng hoạt động mạng thông tin, tuy nhiên do tính phổ cập và linh hoạt của nó, OMNeT++ còn được sử dụng trong nhiều lĩnh vực khác như mô phỏng các hệ thống thông tin phức tạp, các mạng kiểu hàng đợi (queueing networks) hay các kiến trúc phần cứng...

OMNeT++ cung cấp sẵn các thành phần tương ứng với các mô hình thực tế. Các thành phần này (còn được gọi là các module) được lập trình theo ngôn ngữ C++, sau đó được tập hợp lại thành những thành phần hay những mô hình lớn hơn bằng một ngôn ngữ bậc cao (NED). OMNeT++ hỗ trợ giao diện đồ hoạ, tương ứng với các mô hình cấu trúc của nó đồng thời phần nhân mô phỏng (simulation kernel) và các module của OMNeT++ cũng rất dễ dàng nhúng vào trong các ứng dụng khác.

3.1.2. Các thành phần chính của OMNET++

- Thư viện phần nhân mô phỏng (simulation kernel)
- Trình biên dịch cho ngôn ngữ mô tả hình trạng (topology description language) - NED (nedc)
- Trình biên tập đồ hoạ (graphical network editor) cho các file NED (GNED)
- Giao diện đồ hoạ thực hiện mô phỏng, các liên kết bên trong các file thực hiện mô phỏng (Tkenv)
- Giao diện dòng lệnh thực hiện mô phỏng (Cmdenv)
- Công cụ (giao diện đồ hoạ) vẽ đồ thị kết quả vector ở đầu ra (Plove)
- Công cụ (giao diện đồ hoạ) mô tả kết quả vô hướng ở đầu ra (Scalars)
- Công cụ tài liệu hoá các mô hình

- Các tiện ích khác
- Các tài liệu hướng dẫn, các ví dụ mô phỏng...

3.1.3. Ứng dụng

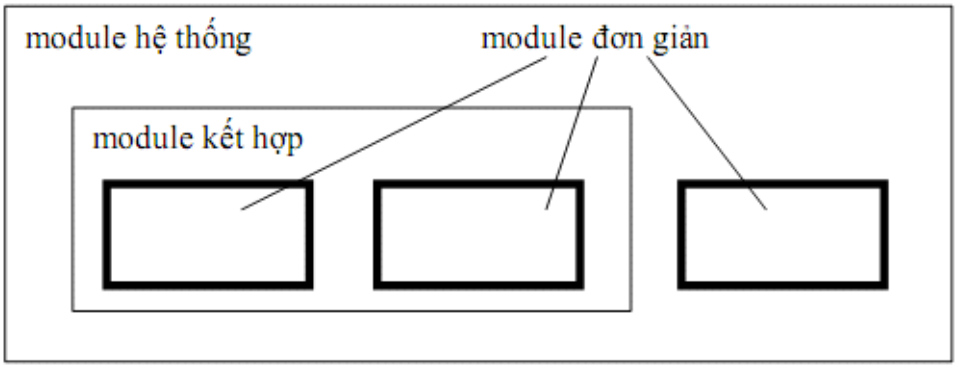
OMNeT++ là một công cụ mô phỏng các hoạt động mạng bằng các module được thiết kế hướng đối tượng. OMNeT++ thường được sử dụng trong các ứng dụng chủ yếu như:

- Mô hình hoạt động của các mạng thông tin
- Mô hình giao thức
- Mô hình hoá các mạng kiểu hàng đợi
- Mô hình hoá các hệ thống đa bộ vi xử lý (multiprocesser) hoặc các hệ thống phần cứng theo mô hình phân tán khác (distributed hardware systems)
- Đánh giá kiến trúc phần cứng
- Đánh giá hiệu quả hoạt động của các hệ thống phức tạp...

3.2. Mô hình trong OMNET++

3.2.1. Cấu trúc phân cấp của các module

Một mô hình trong OMNeT++ chứa các module lồng nhau có cấu trúc phân cấp, trao đổi thông tin với nhau bằng cách gửi các message. Mỗi mô hình này thường biểu diễn cho một hệ thống mạng. Module mức cao nhất trong cấu trúc phân cấp được gọi là module hệ thống. Module này có thể chứa các module con, các module con cũng có thể chứa các module con của riêng nó. Độ sâu phân cấp đối với các module là không giới hạn, điều này cho phép người sử dụng có thể dễ dàng biểu diễn một cấu trúc logic của một hệ thống trong thực tế bằng cấu trúc phân cấp của OMNeT++. Cấu trúc của mô hình có thể được mô tả bằng ngôn ngữ NED của OMNeT++.



Hình 3.1. Các module đơn giản và kết hợp

Các module có thể chứa nhiều module con và được gọi là module kết hợp. Các module đơn giản là các module có cấp thấp nhất trong cấu trúc phân cấp. Các module đơn giản chứa các thuật toán của mô hình. Người sử dụng triển khai các module đơn giản bằng ngôn ngữ C++, sử dụng các thư viện mô phỏng của OMNeT++.

3.2.2. Kiểu module

Tất cả các module dù là đơn giản hay phức tạp đều là các đối tượng cụ thể của các kiểu module. Trong khi mô tả các mô hình, người sử dụng định nghĩa ra các kiểu module; các đối tượng cụ thể của các kiểu module này được sử dụng như các thành phần của các kiểu module phức tạp hơn. Cuối cùng, người sử dụng tạo module hệ thống như một đối tượng cụ thể của kiểu module đã được định nghĩa trước đó, tất cả các module của mạng đều là module con (hoặc là con của module con) của module hệ thống.

Khi một kiểu module được sử dụng như một khối dựng sẵn (building block), sẽ không thể phân biệt đó là một module đơn giản hay phức tạp. Điều này cho phép người sử dụng có thể tách các module đơn giản ra thành nhiều module đơn giản được nhúng trong một module kết hợp, và ngược lại có thể tập hợp các chức năng của một module kết hợp trong một module đơn giản mà không ảnh hưởng gì đến các kiểu module đã được người sử dụng định nghĩa.

Kiểu module có thể được lưu trữ trong một file riêng rẽ. Điều này cho phép người sử dụng có thể nhóm các kiểu module lại và tạo ra một thư viện thành phần.

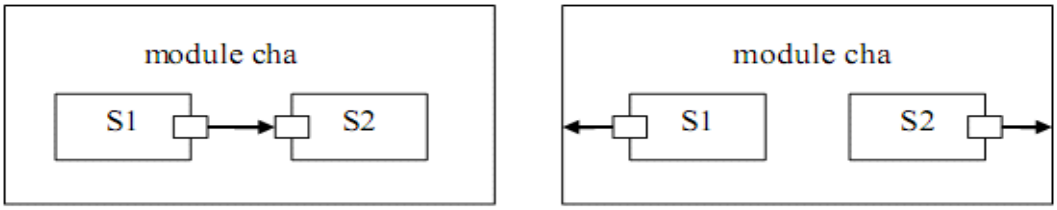
3.2.3. Message, cổng, liên kết

Các module trao đổi thông tin bằng việc gửi các message. Trong thực tế, message có dạng khung (frame) hoặc là các gói tin (packet) được truyền đi trong mạng. Các message có thể có cấu trúc phức tạp tuỳ ý. Các module đơn giản có thể gửi các message đi một cách trực tiếp đến vị trí nhận hoặc gửi đi theo một đường dẫn định sẵn thông qua các cổng và các liên kết.

"Thời gian mô phỏng địa phương" (local simulation time) của một module tăng lên khi module nhận được một message. Message có thể đến từ một module khác hoặc đến từ cùng một module (message của chính bản thân module - self-message được dùng để thực hiện bộ định thời).

Cổng (gate) là các giao tiếp vào ra của module. Message được gửi đi qua các cổng ra và được nhận vào thông qua các cổng vào.

Mỗi kết nối (connection) hay còn gọi là liên kết (link) được tạo bên trong một mức đơn trong cấu trúc phân cấp của các module: bên trong một module kết hợp, một kết nối có thể được tạo ra giữa các cổng tương ứng của hai module con, hoặc giữa cổng của module con với cổng của module kết hợp.



Hình 3.2. Các kết nối

Tương ứng với cấu trúc phân cấp của một mô hình, các message thường di chuyển qua một loạt các kết nối với điểm bắt đầu và kết thúc là các module đơn giản. Tập các kết nối đi từ một module đơn giản và đến một module đơn giản được gọi là route. Các module kết hợp hoạt động giống như các "cardboard box" trong mô hình, "trong suốt" trong việc chuyển tiếp các message giữa các thành phần bên trong và thế giới bên ngoài.

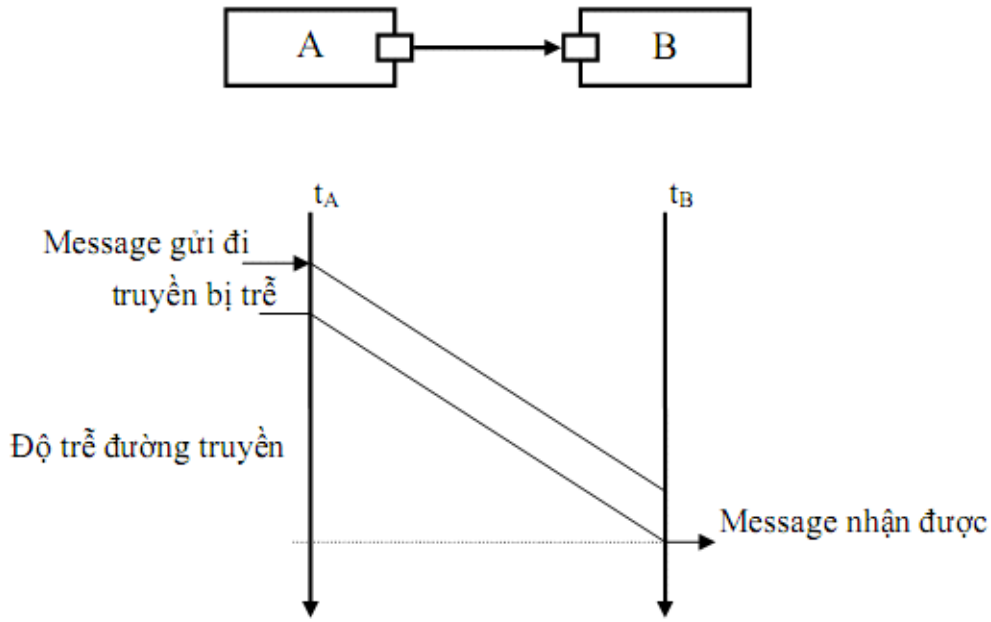
3.2.4. Mô hình truyền gói tin

Một kết nối có thể có ba tham số đặc trưng. Những tham số này rất thuận tiện cho các mô hình mô phỏng mạng thông tin nhưng không hữu dụng lắm cho các kiểu mô hình khác. Ba tham số này bao gồm:

- Độ trễ đường truyền (propagation delay) tính bằng giây.
- Tỉ số lỗi bit, được tính bằng số lỗi/bit.
- Tỉ số dữ liệu, được tính bằng số bit/s.

Các tham số này là tuỳ chọn. Giá trị của các tham số này là khác nhau trên từng kết nối, phụ thuộc vào kiểu của liên kết (hay còn gọi là kiểu của kênh truyền - channel type).

Độ trễ đường truyền là tổng thời gian đến của message bị trễ đi khi truyền qua kênh.



Hình 3.3. Truyền message

Tỉ số lỗi bit ảnh hưởng đến quá trình truyền message qua kênh. Tỉ số này là xác suất các bit bị truyền sai. Do đó xác suất để một message độ dài n bit truyền đi chính xác là:

P(message gửi đi được nhận chính xác) = (1 - BER)n

trong đó BER là tỉ số lỗi bit và n là số bit của message.

Các message truyền đi đều có một cờ lỗi, cờ này sẽ được thiết lập khi việc truyền message có lỗi.

Tỉ số dữ liệu được tính theo đơn vị bit/s, và nó được sử dụng để tính thời gian để truyền một gói tin. Khi tỉ số này được sử dụng, quá trình gửi message đi trong mô hình sẽ tương ứng với việc truyền bit đầu tiên và message được tính là đến nơi sau khi bên nhận đã nhận được bit cuối cùng.

3.2.5. Tham số

Các module có thể các tham số.Các tham số này có thể được đặt giá trị trong các file NED hoặc các file cấu hình ompnetpp.ini.

Các tham số này có thể được dùng để thay đổi các thuộc tính của các module đơn giản hoặc dùng để biểu diễn cho topology của mô hình. Các tham số có thể có kiểu là chuỗi, số học, giá trị logic hoặc cũng có thể chứa cây dữ liệu XML (XML data tree). Các biến kiểu số trong các biểu thức có thể nhận giá trị từ các tham số khác, gọi hàm, sử dụng các biến ngẫu nhiên từ các nguồn phân tán hoặc nhận giá trị trực tiếp được nhập vào bởi người sử dụng.

Các tham số có kiểu số có thể được dùng để cấu hình topology rất dễ dàng. Nằm trong các module kết hợp, các tham số này có thể được dùng để chỉ ra số module con, số cổng giao tiếp và cách các kết nối nội bộ được tạo ra.

3.3. Sử dụng OMNET++

3.3.1. Xây dựng và chạy thử các mô hình mô phỏng

Một mô hình OMNeT++ bao gồm những phần sau:

- Ngôn ngữ mô tả topology - NED (file có phần mở rộng .ned): mô tả cấu trúc của module với các tham số, các cổng... Các file .ned có thể được viết bằng bất kỳ bộ soạn thảo hoặc sử dụng chương trình GNED có trong OMNeT++.
- Định nghĩa cấu trúc của các message (các file có phần mở rộng .msg): Người sử dụng có thể định nghĩa rất nhiều kiểu messsage và thêm các trường dữ liệu cho chúng. OMNeT++ sẽ dịch những định nghĩa này sang các lớp C++ đầy đủ.
- Mã nguồn của các module đơn giản. Đây là các file C++ với phần mở rộng là .h hoặc .cc.

Hệ thống mô phỏng cung cấp cho ta các thành phần sau:

• Phần nhân mô phỏng. Phần này chứa code để quản lý quá trình mô phỏng và các thư viện lớp mô phỏng. Nó được viết bằng C++, được biên dịch và được đặt cùng dạng với các file thư viện (các file có phần mở rộng là .a hoặc .lib).

• Giao diện người sử dụng. Giao diện này được sử dụng khi thực hiện quá trình mô phỏng, tạo sự dễ dàng cho quá trình sửa lỗi, biểu diễn (demonstration) hoặc khi thực hiện mô phỏng theo từng khối (batch execution of simulations).

Có một vài kiểu giao diện trong OMNeT++, tất cả đều được viết bằng C++, được biên dịch và đặt cùng nhau trong các thư viện (các file có phần mở rộng là .a hoặc .lib).

a, Thực hiện mô phỏng và phân tích kết quả

Các chương trình thực hiện mô phỏng (the simulation executable) là các chương trình độc lập, tức là nó có thể chạy trên các máy khác không cài đặt OMNeT++ hay các file mô hình tương ứng. Khi chương trình khởi động, nó bắt đầu đọc file cấu hình (thông thường là file omnetpp.ini). File này chứa các thiết lập để điều khiển quá trình mô phỏng thực hiện, các biến cho các tham số của mô hình... File cấu hình cũng có thể được sử dụng để điều khiển nhiều quá trình mô phỏng, trong trường hợp đơn giản nhất là các quá trình mô phỏng này sẽ được thực hiện lần lượt bởi một chương trình mô phỏng (simulation program).

Đầu ra của quá trình mô phỏng là các file dữ liệu. Các file này có thể là các file vector, các file vô hướng hoặc các file của người sử dụng. OMNeT++ cung cấp một công cụ đồ hoạ Plove để xem và vẽ ra nội dung của các file vector. Tuy nhiên chúng ta cũng nên hiểu rằng khó mà có thể xử lý đầy đủ các file kết quả mà chỉ dùng riêng OMNeT++; các file này đều là các file có định dạng để có thể đọc được bởi các gói xử lý toán học của các chương trình như Matlab hay Octave, hoặc có thể được đưa vào bảng tính của các chương trình như OpenOffice Calc, Gnumeric hay Microsoft Excel. Tất cả các chương trình này đều có chức năng chuyên dụng trong việc phân tích số hoá, vẽ biểu diễn (visualization) vượt qua khả năng của OMNeT++. Các file vô hướng cũng có

thể được biểu diễn bằng công cụ Scalar. Nó có thể vẽ được các biểu đồ, các đồ thị dựa vào tập hợp các toạ độ (x, y) và có thể xuất dữ liệu vào clipboard để có thể sử dụng trong các chương trình khác nhằm đưa những phân tích chi tiết hơn.

b, Giao diện người sử dụng

Mục đích chính của giao diện người sử dụng là che những phần phức tạp bên trong cấu trúc của các mô hình đối với người sử dụng, dễ dàng điều khiển quá trình mô phỏng, và cho phép người sử dụng có khả năng thay đổi các biến hay các đối tượng bên trong của mô hình. Điều này là rất quan trọng đối với pha phát triển và sửa lỗi trong dự án. Giao diện đồ hoạ cũng có thể được sử dụng để trình diễn hoạt động của mô hình. Cùng một mô hình người sử dụng có thể trên nhiều giao diện khác nhau mà không cần phải thay đổi gì trong các file mô hình. Người sử dụng có thể kiểm thử và sửa lỗi rất dễ dàng qua giao diện đồ hoạ, cuối cùng có thể chạy nó dựa trên một giao diện đơn giản và nhanh chóng có hỗ trợ thực hiện theo khối (batch execution).

c, Các thư viện thành phần

Các kiểu module có thể được lưu tại những vị trí độc lập với chỗ mà chúng thực sự được sử dụng. Đặc điểm này cung cấp cho người sử dụng khả năng nhóm các kiểu module lại với nhau và tạo ra các thư viện thành phần.

d, Các chương trình mô phỏng độc lập

Các chương trình thực hiện quá trình mô phỏng có thể được lưu nhiều lần, không phụ thuộc vào các mô hình, sử dụng cùng một thiết lập cho các module đơn giản. Người sử dụng có thể chỉ ra trong file cấu hình mô hình nào sẽ được chạy. Điều này tạo khả năng cho người sử dụng có thể xây dựng những chương trình thực hiện lớn bao gồm nhiều quá trình mô phỏng, và phân phối nó như một công cụ mô phỏng độc lập. Khả năng linh hoạt của ngôn ngữ mô tả topology cũng hỗ trợ cho hướng tiếp cận này.

3.3.2. Hệ thống file

Sau khi cài đặt OMNet++, thư mục omnetpp trên hệ thống máy của bạn nên chứa các thư mục con dưới đây.

Hệ thống mô phỏng:

omnetpp/ thư mục gốc của OMNeT++

bin/ các công cụ trong OMNeT++ (GNED, nedtool...)

include/ các file header cho mô hình mô phỏng

lib/ các file thư viện

bitmaps/ các biểu tượng đồ hoạ

doc/ các file hướng dẫn, readme...

manual/ file hướng dẫn dạng HTML

tictoc-tutorial/ giới thiệu sử dụng OMNeT++

api/ API tham chiếu dạng HTML

nedxml-api/ API tham chiếu cho thư viện NEDXML

src/ mã nguồn của tài liệu

src/ mã nguồn của OMNeT++

nedc/ nedtool, trình biên dịch message

sim/ phần nhân mô phỏng

parsim/ các file dành cho việc thực hiện phân tán

netbuilder/ các file dành cho việc đọc động các file NED

envir/ mã nguồn cho giao diện người sử dụng

cmdenv/ giao diện người dùng dòng lệnh

tkenv/ giao diện người sử dụng dựa trên Tcl/tk

gned/ công cụ soạn thảo file NED

plove/ công cụ vẽ và phân tích đầu ra dạng vector

scalars/ công cụ vẽ và phân tích đầu ra dạng vô hướng

nedxml/ thư viện NEDXML

utils/ các tiện ích khác...

test/ bộ kiểm thử lùi

core/ bộ kiểm thử lùi cho thư viện mô phỏng

distrib/ bộ kiểm thử lùi

samples/ thư mục chứa các mô hình mô phỏng mẫu

aloha/ mô hình của giao thức Aloha

cqn/ Closed Queue Network

Thư mục contrib chứa các chương trình có thể kết hợp với OMNeT++

contrib/

octave/ script của Octave dùng để xử lý kết quả

emacs/ bộ đánh dấu cú pháp NED cho Emacs

3.4. Ngôn ngữ NED

3.4.1. Các chỉ dẫn import

Từ khoá import được sử dụng để thêm các khai báo trong các file mô tả khác. Sau khi đã import, người sử dụng có thể sử dụng tất cả các thành phần đã được định nghĩa trong file mô tả đó.

Chú ý khi thêm một file mô tả, chỉ có các thông tin khai báo được sử dụng. Cũng tương tự như vậy khi một file được thêm vào không có nghĩa là nó sẽ được dịch khi file chứa nó được dịch. Người sử dụng sẽ phải dịch tất cả các file chứ không phải chỉ là file ở mức cao nhất.

Có thể xác định một file thêm vào mà có hoặc không viết phần mở rộng.

Ví dụ:

import “ethenet”; //import ethernet.ned

Cũng có thể sử dụng đường dẫn trong khi sử dụng từ khoá import hoặc tốt hơn là sử dụng trình biên dịch của NED với tham số -I để đặt tên cho thư mục chứa các file muốn import.

3.4.2. Khai báo các kênh

Một định nghĩa kênh được dùng để xác định kiểu kết nối. Tên của kênh có thể được sử dụng sau đó trong file để tạo các liên kết với các tham số khác.

Cú pháp:

channel Tên kênh

//...

endchannel

Ba tham số có thể được gán giá trị trong phần thân của đoạn mã khai báo kênh, tất cả các tham số này đều là các tuỳ chọn: độ trễ, lỗi và tốc độ dữ liệu (datarate). Độ trễ là thời gian trễ trên đường truyền được tính bằng giây. Lỗi là tham số đặc trưng cho xác suất truyền sai một bit trên đường truyền. Tốc độ dữ liệu là tham số được tính bằng độ rộng băng thông của kênh truyền, được tính bằng bit/s và được dùng để tính thời gian truyền của một gói tin. Các thuộc tính có thể xuất hiện theo bất kỳ thứ tự nào trong khai báo.

Giá trị của các tham số (thuộc tính) nên là các hằng số.

Ví dụ:

```
channel LeasedLine
delay 0.0018 // sec
error 1e-8
datarate 128000 // bit/sec
endchannel
```

3.4.3. Khai báo các module đơn giản

Các module đơn giản là các khối chương trình được xây dựng sẵn cho các module khác (có thể là các module kết hợp). Các module được khai báo bằng tên và theo quy ước tên của các module này được đặt tên bắt đầu bằng chữ cái in hoa.

Các module đơn giản được khai báo thông qua các cổng và các tham số.

Cú pháp:

```
simple SimpleModuleName
parameters:
//...
gates:
//...
endsimple
```

3.4.3.1. Các tham số của module đơn giản

Các tham số là các biến phụ thuộc vào từng mô hình. Tham số của các module đơn giản được sử dụng bởi các hàm (hay còn được gọi là các thuật toán của module) khai báo trong chính module. Theo quy ước các tham số sẽ được đặt tên bắt đầu bằng chữ cái thường.

Các tham số được khai báo bằng cách liệt kê tên sau từ khoá parameters. Kiểu của các tham số có thể là kiểu số (numeric), hằng số (numeric const hay viết gọn là const), giá trị logic (bool), kiểu chuỗi (string) hoặc xml. Khi tham số không khai báo rõ kiểu thì mặc định kiểu của tham số đó là numeric.

Ví dụ:

```
simple TrafficGen
parameters:
interarrivalTime,
numOfMessages : const,
address : string;
gates: //...
endsimple
```

Các tham số có thể được gán giá trị từ NED (khi các module được sử dụng như các khối dựng sẵn của một khối kết hợp lớn hơn) hoặc từ file cấu hình omnetpp.ini.

3.4.3.2. Các cổng của module đơn giản

Cổng là các điểm kết nối của module. Điểm bắt đầu và kết thúc một kết nối giữa hai module chính là các cổng. OMNeT++ hỗ trợ kiểu kết nối một chiều (đơn công) do đó có hai loại cổng là cổng vào và cổng ra. Các message được gửi đi từ cổng ra và được nhận vào từ cổng vào. Theo quy ước, các cổng được đặt tên bắt đầu bằng chữ cái thường.

Cổng được khai báo bằng cách khai báo tên sau từ khoá gates. Cặp dấu [] thể hiện một vector cổng. Các thành phần của một vector cổng được đánh số bắt đầu từ 0.

Ví dụ:

```
simple NetworkInterface
parameters: //...
gates:
in: fromPort, fromHigherLayer;
out: toPort, toHigherLayer;
endsimple
simple RoutingUnit
parameters: //...
gates:
in: output[];
out: input[];
endsimple
```

Kích thước của một vector cổng có thể được xác định sau do đó mỗi đối tượng cụ thể của một mô hình có thể có các vector cổng có kích thước khác nhau.

3.4.4. Khai báo các module kết hợp

Module kết hợp là các module có thể chứa một hoặc nhiều các module con. Bất kỳ kiểu module nào (đơn giản hay kết hợp) đều có thể được dùng như là một module con.

Cũng giống như các module đơn giản, các module kết hợp cũng có các cổng, các tham số và chúng có thể được sử dụng ở bất kỳ chỗ nào mà các module đơn giản có thể được sử dụng.

Theo quy ước, tên của các module (bao gồm cả kiểu module kết hợp) đều được bắt đầu bằng chữ hoa. Các module con có thể sử dụng các tham số của module cha. Các module con này có thể kết nối với nhau hoặc/và kết nối với module kết hợp chứa chúng.

Việc khai báo các module kết hợp cũng tương tự như khai báo các module đơn giản. Phần khai báo cũng bao gồm các từ khoá parameters và gates, ngoài ra nó còn sử dụng thêm hai từ khoá là submodules và connections.

Cú pháp:

```
module Tên_module
parameters:
//...
gates:
//...
submodules:
//...
connections:
//...
endmodule
```

3.4.5. Khai báo mạng

Để thực sự tạo một mô hình mô phỏng chạy được thì người sử dụng phải khai báo mạng. Việc khai báo mạng sẽ tạo ra một mô hình mô phỏng như là một đối tượng cụ thể của một kiểu module đã định nghĩa trước đó. Kiểu module ở đây thường là một module kết hợp, tuy nhiên cũng có thể tạo ra một mạng chỉ là một module đơn giản độc lập.

Có thể khai báo nhiều mạng trong một hoặc nhiều file NED. Chương trình mô phỏng sử dụng các file NED đó sẽ có thể chạy bất cứ một mạng nào. Nếu bạn muốn cụ thể một mạng nào đó được thực hiện bạn có thể chỉ rõ trong file cấu hình (omnetpp.ini).

Cú pháp khai báo mạng cũng tương tự như khai báo các module con:

```
network wirelessLAN: WirelessLAN
parameters:
numUsers=10,
httpTraffic=true,
```

```
ftpTraffic=true,
distanceFromHub=truncnormal(100,60);
endnetwork
```

Ở đây WirelessLAN là tên của một kiểu module kết hợp đã định nghĩa từ trước, trong đó có thể chứa các kiểu module kết hợp khác như WirelessHost, WirelessHub... Một cách tự nhiên, chỉ các kiểu module không có cổng mới có thể được dùng trong các khai báo mạng.

Chương 4 - MÔ PHỎNG VÀ ĐÁNH GIÁ HIỆU QUẢ NĂNG LƯỢNG CỦA CSMA, S-MAC, T-MAC

4.1. Thiết lập mô hình mô phỏng

Các giao thức CSMA, S-MAC, T-MAC được mô phỏng trên cơ sở hoạt động của nút cảm biến EYES.

Hình 4.1. Nút cảm biến EYES

Nút cảm biến EYES được trang bị một bộ xử lý Texas Instruments MSP430F149 với 2KB RAM và 60 KB bộ nhớ Flash; bộ vi xử lý 16 bít có thể chạy ở nhiều xung nhịp, cực đại là 5MHz. Nút cảm biến EYES truyền thông sử dụng sóng vô tuyến 115kbps (RFM TR1001, 868.35 MHz, hybrid transceiver), trang bị với một bộ nhớ 2Mb EEPROM (AST 25P20V6). Nút cảm biến EYES có nhiều giao diện tương tác với thế giới bên ngoài bao gồm JTAG, RS232, 2 LEDs, … Năng lượng cung cấp cho nút cảm biến là 02 pin AA hiệu điện thế 3V chiếm hầu hết thể tích của nút.

CPU	
active (5 MHz)	2.1 mA
sleep	1.6 μA
Radio	
transmit	10 mA
receive	4 mA
sleep	20 μA

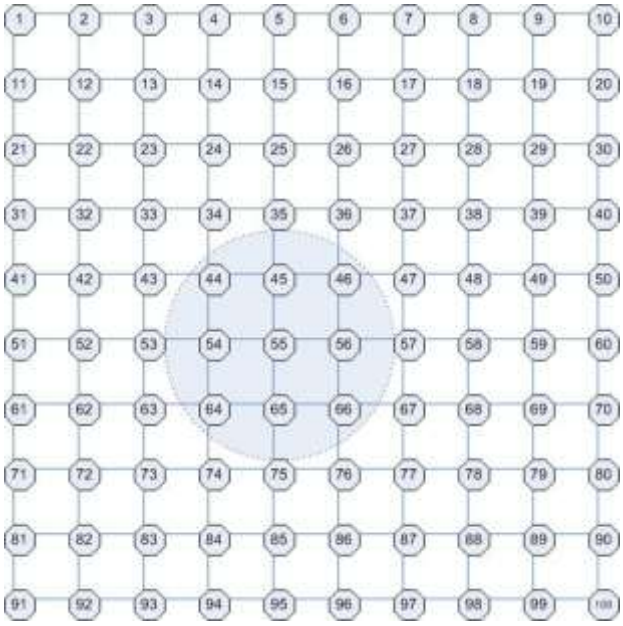
Bảng 4.1. Thông số tiêu thụ điện của nút cảm biến EYES

Dung lượng và khả năng và tiêu thụ điện của nút EYES giống với những nút cảm biến nguyên mẫu khác. Bộ nhớ RAM 2KB và năng lượng cung cấp, là

hai dạng tài nguyên khan hiếm. Do vậy một giao thức MAC được thiết kế sao cho việc sử dụng những tài nguyên đó là ít nhất có thể.

Để minh họa những giao thức điều khiển truy nhập MAC được giới thiệu ở trên, thực hiện xây dựng mô hình mô phỏng hoạt động thực tế của nút cảm biến EYES trên công cụ mô phỏng OMNET++. Các giao thức MAC được cài đặt cho nút EYES để so sánh và đánh giá gồm: S-MAC, T-MAC và CSMA. Sở dĩ cài đặt và mô phỏng cả CSMA bởi vì ở đây xem xét đến trường hợp tồi nhất của giao thức điều khiển truy nhập đối với mạng cảm biến không dây. Tồi nhất trên khía cạnh CSMA không có đặc tính tiết kiệm năng lượng.

Để mô phỏng tiến hành xây dựng một ma trận các nút cảm biến gồm 100 nút phân bố trong một mạng lưới đều nhau 10x10 như trên hình 4.2. Trong ma trận đó, chọn công suất phát sóng của mỗi nút sao cho nếu nút ở trung tâm ma trận thì nó chỉ có 8 nút lân cận. Ví dụ, trong hình 4.2, nút 55 có các lân cận là: 44, 45, 46, 54, 56, 64, 65, 66.



Hình 4.2. Ma trận 100 nút cảm biến phân bố đều nhau

Đối với nút cảm biến EYES, mức tiêu thụ điện đo được trong thực tế: 20μA trong khi ngủ, 4mA trong khi nhận và 10mA trong khi truyền. Vì nguồn điện cung cấp cho mỗi nút cảm biến hoạt động có hiệu điện thế không đổi là 3V, thời gian mô phỏng là xác định, do vậy có thể dễ dàng tính được năng lượng tiêu thụ trung bình khi xác định được dòng điện tiêu thụ trung bình. Để thuận tiện trong tính toán, trong mô phỏng sử dụng đại lượng dòng điện tiêu thụ trung bình thay cho năng lượng tiêu thụ trung bình.

Trong mô phỏng, sử dụng nút cảm biến EYES có xung nhịp là 32768 xung trong một giây, thực hiện mô phỏng giao thức S-MAC với khung thời gian có độ dài 1 giây tương ứng với 32768 xung (tick), thời gian tích cực thay đổi theo mục đích mô phỏng. Với giao thức T-MAC, sử dụng cố định độ dài khung là 610ms (20000 xung nhịp), độ dài khoảng cách TA là 15ms (500 xung nhịp). Với giao thức T-MAC, sử dụng kỹ thuật tránh nghe thừa (overhearing avoidance), kỹ thuật gửi RTS sớm. Thời gian thực hiện mô phỏng là 30 giây.

Trong mô phỏng có một thông số quan trọng để theo dõi hiệu suất giao thức đó là thông lượng. Ở đây thay đổi thông lượng thông qua thay đổi tham số mô phỏng khác là: tốc độ phát sinh gói tin của mỗi nút cảm biến.

4.2. Kết quả mô phỏng và đánh giá

Tiến hành chạy mô phỏng với giao thức CSMA, quá trình hoạt động của mạng như sau:

```
Initializing...
0.0000000  ( 0.00s) net.nodes[0].mac: mac=Csma
0.0000000  ( 0.00s) net.nodes[0].mac: mac1=0.000000 mac2=0.000000
mac3=0.000000 mac4=0.000000
0.0000000  ( 0.00s) net.scenario: msglen = 20
0.0000000  ( 0.00s) net.scenario: msginterval = 10.000000
0.0000000  ( 0.00s) net.prop: node 0 is at (0.000000,0.000000)

…

Running simulation...
0.2076919670 (207ms) net.nodes[7].idle1: generate
0.2076919670 (207ms) net.nodes[7].appsel: Queuing new msg, pattern = 0

…

30.557766  (30.55s) net.nodes[15].radio: stats: sleep=0.0000 tx=0.0105
rx=0.1389 tx_lb=0.0008 rx_lb=30.4075 collision=0.0000
30.557766  (30.55s) net.nodes[15].idle1: stats: tx=3 rx=3 delay=0.003749
```

```
Initializing...
0.0000000  ( 0.00s) net.nodes[0].mac: mac=Csma
0.0000000  ( 0.00s) net.nodes[0].mac: mac1=0.000000 mac2=0.000000
mac3=0.000000 mac4=0.000000
0.0000000  ( 0.00s) net.scenario: msglen = 20
0.0000000  ( 0.00s) net.scenario: msginterval = 10.000000
0.0000000  ( 0.00s) net.prop: node 0 is at (0.000000,0.000000)
…

Running simulation...
0.2076919670 (207ms) net.nodes[7].idle1: generate
0.2076919670 (207ms) net.nodes[7].appsel: Queuing new msg, pattern = 0
…
```

Bảng 4.2. Tiến trình mô phỏng giao thức CSMA

Sau khi chạy file kd.pl, kết quả mô phỏng trên được tổng hợp như sau:

time=30.557766 nodes=16 app_tx=50 app_rx=50 rt_tx=50 rt_rx=50 rt_tx_drop=0 mac_tx=50 mac_rx=50 radio_sleep=0 radio_tx=0.1885 radio_rx=488.735 radio_collision=0 in_queue=0 mac_rx_data=0.1867 mac_rx_overhead=0 mac_rx_overhear=2.527 mac_tx_data=0.1867 mac_tx_overhead=0 own_sched=0 mac=Csma msglen=20 msginterval=10 mac1=0.000000 mac2=0.000000 mac3=0.000000 mac4=0.000000

```
time=30.557766 nodes=16 app_tx=50 app_rx=50 rt_tx=50 rt_rx=50
rt_tx_drop=0 mac_tx=50 mac_rx=50 radio_sleep=0 radio_tx=0.1885
radio_rx=488.735 radio_collision=0 in_queue=0 mac_rx_data=0.1867
mac_rx_overhead=0 mac_rx_overhear=2.527 mac_tx_data=0.1867
mac_tx_overhead=0 own_sched=0 mac=Csma msglen=20
```

Bảng 4.3. Kết quả mô phỏng giao thức CSMA

Tiến hành chạy mô phỏng với giao thức S-MAC, quá trình hoạt động của mạng như sau:

```
Initializing...
0.0000000  ( 0.00s) net.nodes[0].mac: mac=SMac
0.0000000  ( 0.00s) net.nodes[0].mac: mac1=2000.000000
mac2=20000.000000 mac3=0.000000 mac4=0.000000
0.0000000  ( 0.00s) net.scenario: msglen = 20
0.0000000  ( 0.00s) net.scenario: msginterval = 10.000000
0.0000000  ( 0.00s) net.prop: node 0 is at (0.000000,0.000000)
…
Running simulation...
0.2076919670 (207ms) net.nodes[7].idle1: generate
0.2076919670 (207ms) net.nodes[7].appsel: Queuing new msg, pattern = 0
…
30.200204  (30.20s) net.nodes[15].radio: stats: sleep=26.6760 tx=0.0233
rx=0.0946 tx_lb=0.0034 rx_lb=3.4029 collision=0.0000
30.200204  (30.20s) net.nodes[15].idle1: stats: tx=3 rx=3 delay=0.398374
```


```
Initializing...
0.0000000  ( 0.00s) net.nodes[0].mac: mac=SMac
0.0000000  ( 0.00s) net.nodes[0].mac: mac1=2000.000000
mac2=20000.000000 mac3=0.000000 mac4=0.000000
0.0000000  ( 0.00s) net.scenario: msglen = 20
0.0000000  ( 0.00s) net.scenario: msginterval = 10.000000
0.0000000  ( 0.00s) net.prop: node 0 is at (0.000000,0.000000)
…
Running simulation...
0.2076919670 (207ms) net.nodes[7].idle1: generate
0.2076919670 (207ms) net.nodes[7].appsel: Queuing new msg, pattern = 0
…
30.200204  (30.20s) net.nodes[15].radio: stats: sleep=26.6760 tx=0.0233
rx=0.0946 tx_lb=0.0034 rx_lb=3.4029 collision=0.0000
30.200204  (30.20s) net.nodes[15].idle1: stats: tx=3 rx=3 delay=0.398374
```

```
time=30.200204 nodes=16 app_tx=50 app_rx=50 rt_tx=50 rt_rx=50
rt_tx_drop=0 mac_tx=50 mac_rx=50 radio_sleep=426.8038
radio_tx=0.4508 radio_rx=55.9487 radio_collision=0 in_queue=0
mac_rx_data=0.1999 mac_rx_overhead=0.6686
mac_rx_overhear=0.9722 mac_tx_data=0.2 mac_tx_overhead=0.2426
own_sched=1 mac=SMac msglen=20 msginterval=10
mac1=2000.000000 mac2=20000.000000 mac3=0.000000
mac4=0.000000
```

Bảng 4.4. Tiến trình mô phỏng giao thức S-MAC

Sau khi chạy file kd.pl, kết quả mô phỏng trên được tổng hợp như sau:

```
time=30.200204 nodes=16 app_tx=50 app_rx=50 rt_tx=50 rt_rx=50
rt_tx_drop=0 mac_tx=50 mac_rx=50 radio_sleep=426.8038
radio_tx=0.4508 radio_rx=55.9487 radio_collision=0 in_queue=0
mac_rx_data=0.1999 mac_rx_overhead=0.6686 mac_rx_overhear=0.9722
mac_tx_data=0.2 mac_tx_overhead=0.2426 own_sched=1 mac=SMac
msglen=20 msginterval=10 mac1=2000.000000 mac2=20000.000000
mac3=0.000000 mac4=0.000000
```

Bảng 4.5. Kết quả mô phỏng giao thức S-MAC

Tiến hành chạy mô phỏng với giao thức T-MAC, quá trình hoạt động của mạng như sau:

```
Initializing...
0.0000000  ( 0.00s) net.nodes[0].mac: mac=TMac
0.0000000  ( 0.00s) net.nodes[0].mac: mac1=500.000000
mac2=20000.000000 mac3=1.000000 mac4=0.000000
0.0000000  ( 0.00s) net.scenario: msglen = 20

…

Running simulation...
0.2076919670 (207ms) net.nodes[7].idle1: generate
0.2076919670 (207ms) net.nodes[7].appsel: Queuing new msg, pattern =
0
```

```
Initializing...
0.0000000  ( 0.00s) net.nodes[0].mac: mac=TMac
0.0000000  ( 0.00s) net.nodes[0].mac: mac1=500.000000
mac2=20000.000000 mac3=1.000000 mac4=0.000000
0.0000000  ( 0.00s) net.scenario: msglen = 20
…
Running simulation...
0.2076919670 (207ms) net.nodes[7].idle1: generate
0.2076919670 (207ms) net.nodes[7].appsel: Queuing new msg, pattern = 0
…
30.260765  (30.26s) net.nodes[15].idle1: stats: tx=3 rx=3 delay=0.461452
```

Bảng 4.6. Tiến trình mô phỏng giao thức T-MAC

Sau khi chạy file kd.pl, kết quả mô phỏng trên được tổng hợp như sau:

```
time=30.260765 nodes=16 app_tx=50 app_rx=50 rt_tx=50 rt_rx=50
rt_tx_drop=0 mac_tx=50 mac_rx=50 radio_sleep=450.4184
radio_tx=0.4578 radio_rx=33.2963 radio_collision=0.0039 in_queue=0
mac_rx_data=0.1999 mac_rx_overhead=0.6502
mac_rx_overhear=0.9694 mac_tx_data=0.2 mac_tx_overhead=0.2481
own_sched=1 mac=TMac msglen=20 msginterval=10 mac1=500.000000
mac2=20000.000000 mac3=1.000000 mac4=0.000000
```

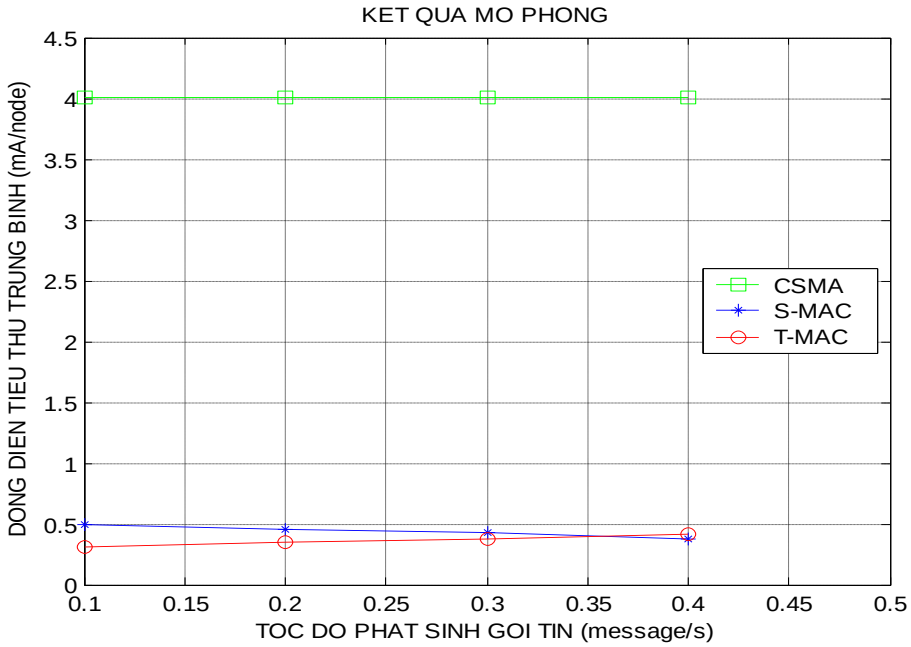

```
time=30.260765 nodes=16 app_tx=50 app_rx=50 rt_tx=50 rt_rx=50
rt_tx_drop=0 mac_tx=50 mac_rx=50 radio_sleep=450.4184
radio_tx=0.4578 radio_rx=33.2963 radio_collision=0.0039 in_queue=0
mac_rx_data=0.1999 mac_rx_overhead=0.6502
mac_rx_overhear=0.9694 mac_tx_data=0.2 mac_tx_overhead=0.2481
own_sched=1 mac=TMac msglen=20 msginterval=10 mac1=500.000000
mac2=20000.000000 mac3=1.000000 mac4=0.000000
```

Bảng 4.7. Kết quả mô phỏng giao thức T-MAC

Chạy file nl2.pl để phân tích và tổng hợp kết quả mô phỏng có bảng sau:

Tốc độ phát sinh gói tin (message/s)	Dòng điện tiêu thụ trung bình (mA/node)			
	CSMA	S-MAC	T-MAC-oa	T-MAC-oa-frts
0,1	4,0023	0,4901	0,3031	0,3072
0,2	4,0047	0,4554	0,3462	0,3669
0,3	4,0069	0,4236	0,3801	0,4292
0,4	4,0091	0,3793	0,4126	0,4591

Bảng 4.8. Tổng hợp kết quả mô phỏng



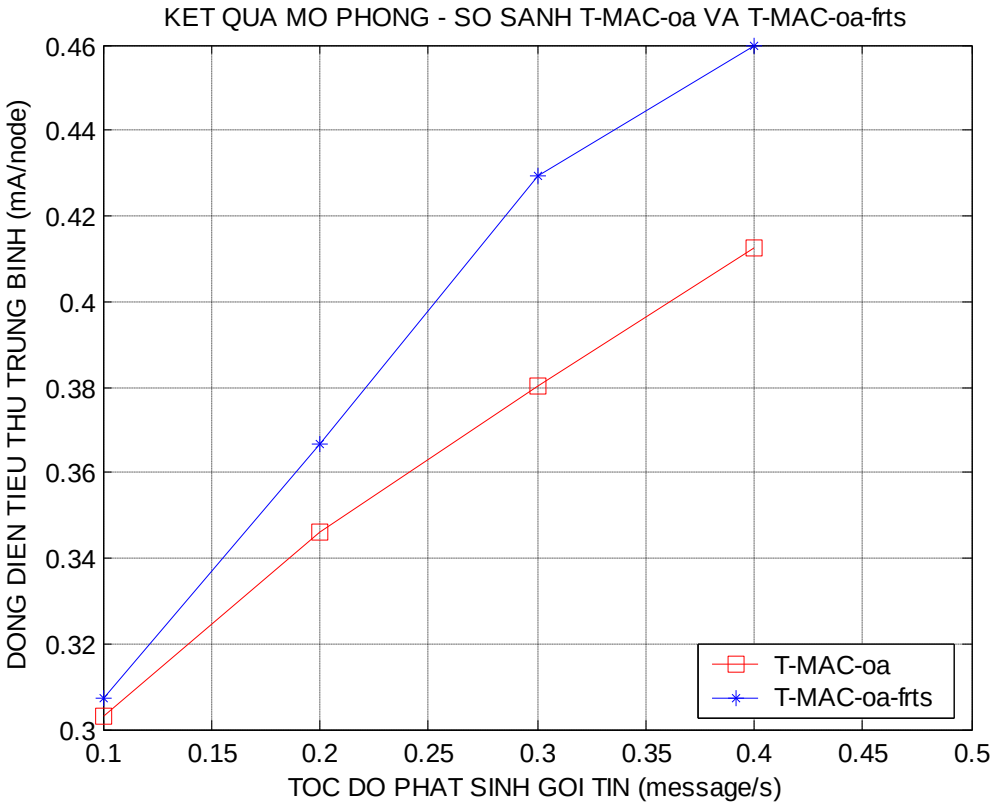
Hình 4.3. Dòng điện tiêu thụ trung bình ứng với từng giao thức thay đổi theo tốc độ phát sinh gói tin

Từ bảng 4.8, tiến hành dựng đồ thị mối quan hệ giữa tốc độ phát sinh gói tin và dòng điện tiêu thụ trung bình của các giao thức, kết quả cho đồ thị hình 4.3.

Từ đồ thị cho thấy: với CSMA mức tiêu thụ năng lượng là rất cao và hầu như không đổi, đơn giản vì CSMA không có đặc tính hiệu năng. Với S-MAC và T-MAC mức tiêu thụ năng lượng khá nhỏ.

Đối với T-MAC cho thấy dòng điện tiêu thụ trung bình của T-MAC tuy thấp hơn so với S-MAC nhưng tăng tỉ lệ thuận với tốc độ phát sinh gói tin. Đó chính là sự khác biệt của T-MAC so với S-MAC, vì không như S-MAC, trong T-MAC các nút cảm biến vẫn duy trì trạng thái thức khi lân cận của nó trao còn đổi dữ liệu.

Đối với S-MAC dòng điện tiêu thụ trung bình lại giảm đi khi tăng tốc độ phát sinh gói tin. Điều này có thể giải thích: với S-MAC, các nút cảm biến sẽ tắt thành phần vô tuyến chuyển sang trạng thái ngủ khi lân cận của nó đang có sự trao đổi dữ liệu. Trong mô phỏng khi ta tăng tốc độ phát sinh gói tin (của toàn mạng) số lượng và thời các cuộc trao đổi dữ liệu giữa các nút sẽ tăng. Do đó số lượng và thời gian các nút phải duy trì trạng thái ngủ sẽ tăng, dòng điện tiêu thụ trung bình sẽ giảm.



Hình 4.4. So sánh T-MAC-oa với T-MAC-oa-frts

Tuy nhiên, do hiện tượng ngủ sớm *(early sleeping problem)* nên thông lượng cực đại của T-MAC bị giới hạn. Trong mô phỏng, thực hiện gia tăng tốc độ phát sinh gói tin thì thấy rằng sau một thời điểm tăng tỉ lệ thuận với tốc độ phát sinh gói tin, dòng điện tiêu thụ trung bình của mạng giảm và đi đến "bão hòa". Đây là thời điểm không gói tin nào được chuyển. Để khắc phục hiện tượng thông lượng cực đại bị giới hạn bởi hiện tượng ngủ sớm, một trong những giải

pháp của T-MAC là sử dụng kỹ thuật gửi sớm RTS (*Future Request to Send - FRTS*). Thực hiện mô phỏng hoạt động của T-MAC sử dụng kỹ thuật FRTS. Hình 4.4 thể hiện kết quả mô phỏng so sánh T-MAC có FRTS và T-MAC không có FRTS. Từ đồ thị cho thấy thông lượng cực đại của T-MAC có FRTS cao hơn so với T-MAC không FRTS, tuy nhiên dòng điện tiêu thụ trung bình cũng vì đó mà tăng lên tương ứng.

Kết quả mô phỏng cho thấy được mức độ tiêu thụ dòng điện trung bình của T-MAC rõ ràng là thấp hơn khá nhiều so với S-MAC và tất nhiên là thấp hơn rất nhiều so với CSMA.

Nhược điểm của T-MAC là thông lượng lớn nhất thấp hơn so với S-MAC do hiện tượng ngủ sớm. Tuy nhiên, với những ứng dụng mạng cảm biến vấn đề này không phải là vấn đề lớn.

KẾT LUẬN

Bản luận văn đã giới thiệu tổng quan về mạng cảm biến không dây, nghiên cứu đánh giá một số cơ chế điều khiển truy nhập môi trường (MAC) điển hình như CSMA, S-MAC, T-MAC thông qua mô phỏng bằng bộ phần mềm OMNET++.

Các kết quả mô phỏng cho thấy với CSMA mức tiêu thụ năng lượng là rất cao và hầu như không đổi. Trong khi đó với giao thức T-MAC và S-MAC mức tiêu thụ năng lượng là khá nhỏ.

Đối với T-MAC dòng điện tiêu thụ trung bình tuy thấp hơn S-MAC nhưng tăng tỉ lệ thuân với tốc độ phát sinh gói tin. Tuy nhiên, do hiện tượng ngủ sớm nên thông lượng cực đại của T-MAC bị giới hạn. Để khắc phục hiện tượng thông lượng cực đại bị giới hạn bởi hiện tượng ngủ sớm, một trong những giải pháp của T-MAC là sử dụng kỹ thuật gửi sớm RTS. Trong mô phỏng cho thấy thông lượng cực đại của T-MAC có FRTS cao hơn so với T-MAC không FRTS, tuy nhiên dòng điện tiêu thụ trung bình cũng vì đó mà tăng lên tương ứng.

Đối với S-MAC dòng điện tiêu thụ trung bình lại giảm khi tăng tốc độ phát inh gói tin.

Đồ án cũng đã giới thiệu một cách tổng quan về OMNET++, là một phần mềm dùng để mô phỏng mạng rất mạnh và hiệu quả