Tạp chí Khoa học Đại học Huế: Kỹ thuật và Công nghệ; pISSN 2588-1175 | eISSN 2615-9732

Tập 134, Số 2A, 2025, Tr. 109–125; DOI: 10.26459/hueunijtt.v134i2A.8158

ĐÁNH GIÁ CÁC GIAO THỨC CÂY TRUY VẤN DỰA TRÊN TRI THỨC VỚI CÁC PHÂN BỐ THẺ RFID KHÁC NHAU

Nguyễn Đức Nhật Quang[1*], Nguyễn Phan Nguyên Bảo[1], Nguyễn Ngọc Thủy[1], Nguyễn Thanh Bình[1], Phạm Thị Thúy Sang[2]

[1] Trường Đại học Khoa học, Đại học Huế, Huế, Việt Nam
[2] Trung tâm Công nghệ thông tin Thành phố Huế, Huế, Việt Nam

Tóm tắt. Truy vấn thẻ trong hệ thống RFID là bước then chốt để nhận dạng các thẻ xuất hiện trong vùng đọc. Nhiều giao thức truy vấn thẻ đã được đề xuất, tuy nhiên việc giảm số khe xung đột thường kéo theo sự gia tăng số khe nhàn rỗi, ảnh hưởng đến hiệu suất truy vấn. Trong một số kịch bản triển khai, số lượng thẻ, miền định danh thẻ và phân bố thẻ có thể được biết trước; các thông tin này tạo thành "tri thức" và đã được khai thác để xây dựng các giao thức truy vấn dựa trên tri thức. Bài báo này đánh giá các giao thức cây truy vấn dựa trên tri thức dưới hai kiểu phân bố thẻ: đồng nhất và không đồng nhất. Kết quả cài đặt cho thấy các giao thức cây truy vấn dựa trên tri thức luôn đạt hiệu suất truy vấn cao hơn so với giao thức truy vấn cây truyền thống, trong đó QTKS và QTKS_TE cho kết quả tốt nhất. Phân tích và thực nghiệm cũng chỉ ra rằng, mức độ khai thác tri thức càng cao thì hiệu suất truy vấn thẻ càng được cải thiện.

Từ khóa: Hệ thống RFID, cây truy vấn dựa trên tri thức, phân bố thẻ RFID, hiệu suất truy vấn

Evaluation of knowledge-based query tree protocols with different RFID tag distributions

Nguyen Duc Nhat Quang[1*], Nguyen Phan Nguyen Bao[1], Nguyen Ngoc Thuy[1], Nguyen Thanh Binh[1], Pham Thi Thuy Sang[2]

[1] University of Sciences, Hue University, Hue, Vietnam
[2] Hue Center of Information and Technology, Hue, Vietnam

Abstract. Tag query in RFID systems is a key operation for identifying tags that appear in the reader's interrogation zone. Many tag query protocols have been proposed; however, reducing the number of collision slots often comes at the cost of increasing the number of idle slots. In certain deployment scenarios, the number of tags, the tag ID space, and the tag distribution can be known in advance; this information constitutes *knowledge* and has been exploited in several knowledge-based query algorithms. This paper evaluates knowledge-based query tree protocols under different tag distributions: uniform and non-uniform. Implementation results show that knowledge-based query tree protocols consistently achieve

** Liên hệ: ndnquang@hueuni.edu.vn*

Ngày nhận bài: 18–11–2025; Ngày nhận đăng: 10–12–2025

better query efficiency than the traditional query tree protocol, with QTKS and QTKS_TE providing the best performance. The analytical and implementation results also indicate that the more knowledge is exploited in the protocols, the higher the achievable query efficiency.

Keywords: RFID systems, knowledge-based query tree, tag distributions, query efficiency

1 Giới thiệu

Nhận dạng bằng tần số vô tuyến (Radio Frequency Identification – RFID) là công nghệ nhận dạng tự động sử dụng sóng vô tuyến để trao đổi dữ liệu hai chiều, không tiếp xúc giữa thẻ (tag) và đầu đọc (reader) [1]. Một hệ thống RFID gồm ba thành phần chính: thẻ, đầu đọc và hệ thống (máy tính) quản lý/xử lý dữ liệu. Thẻ (gồm chip và ăng-ten) được gắn lên đối tượng để lưu trữ thông tin và truyền không dây đến đầu đọc; hệ thống quản lý/xử lý dữ liệu tiếp nhận và xử lý thông tin này. Thẻ RFID được phân loại thành thẻ thụ động, bán thụ động và chủ động; tuy nhiên, do bài báo chỉ tập trung vào giao thức truyền thông chống xung đột, các loại thẻ này được xem là tương đương trong phạm vi nghiên cứu.

Khi đầu đọc phát bản tin truy vấn đến toàn bộ vùng đọc, nhiều thẻ có thể phản hồi đồng thời, gây xung đột, buộc đầu đọc phải truy vấn lặp lại, làm tăng chi phí liên lạc và thời gian nhận dạng. Do đó, cần các giao thức chống xung đột hiệu quả, đặc biệt trong các hệ thống IoT có số lượng thẻ lớn. Các giao thức chống xung đột RFID chủ yếu gồm hai nhóm: dựa trên Aloha [1-3] và dựa trên cây [4-6]. Các giao thức Aloha (như Slotted Aloha, Framed Slotted Aloha – FSA) ước lượng số lượng thẻ để gán số khe thời gian thích hợp, giảm xác suất xung đột. Các giao thức dựa trên cây, như Cây nhị phân (Binary Tree – BT) [7, 8] và Cây truy vấn (Query Tree – QT) [9, 10], chia dần tập thẻ xung đột thành các tập con cho đến khi mỗi tập con chỉ còn một thẻ được nhận dạng; BT dùng số ngẫu nhiên nhị phân, trong khi QT dựa trên ID thẻ. Giao thức QT có yêu cầu thấp hơn về năng lực thẻ, nhưng hiệu suất bị ảnh hưởng đáng kể bởi phân bố ID thẻ.

Một số hệ thống RFID được trang bị cơ sở dữ liệu, nhờ đó đầu đọc có thể biết trước số lượng thẻ cần nhận dạng [11], miền giá trị ID thẻ có thể xuất hiện [12] hoặc kiểu phân bố của các thẻ [11, 13]; tập thông tin này được gọi chung là "tri thức" (knowledge). Trong một số kịch bản, đầu đọc thực sự biết toàn bộ miền ID của các thẻ tiềm năng. Chẳng hạn, hệ thống RFID trong siêu thị thường có cơ sở dữ liệu lưu trữ ID thẻ của tất cả hàng hóa được nhập để bán; tương tự, hệ thống RFID trong nhà máy thường quản lý cơ sở dữ liệu chứa ID thẻ của toàn bộ sản phẩm được sản xuất tại đó. Tuy nhiên, do các thẻ này có thể xuất hiện (appearing tag) hoặc không xuất hiện trong từng phiên làm việc cụ thể, nên hệ thống thường phải định kỳ thực hiện quá trình nhận dạng. Ví dụ, trong siêu thị, mỗi khách hàng chỉ mua một tập con hàng hóa (tương ứng với tập con thẻ xuất hiện) và các thẻ này cần được nhận dạng để phục vụ thanh toán. Một cách tiếp cận trực quan là lần lượt truy vấn từng thẻ tiềm năng (potential), song hiệu quả nhận dạng sẽ rất thấp khi số thẻ thực sự xuất hiện chỉ chiếm một phần nhỏ trong tập thẻ tiềm năng. Hiện nay đã có một số nghiên cứu về giao thức chống xung đột thẻ khai thác loại tri thức này. Trong một số giao thức,

đầu đọc liên tục nhận dạng các thẻ xuất hiện trong hệ thống; vì vậy, giao thức có tri thức về các thẻ đã được nhận dạng trước đó và có khả năng phát hiện việc các thẻ này còn hiện diện hay đã rời khỏi vùng phủ sóng. Ngược lại, một số giao thức giả định đầu đọc có tri thức về toàn bộ ID thẻ có thể xuất hiện. Do đó, nếu giao thức chống xung đột RFID khai thác hiệu quả các dạng tri thức này, quá trình nhận dạng thẻ có thể được tăng tốc đáng kể.

Gần đây, một số nghiên cứu đã đề xuất các giao thức chống xung đột dựa trên tri thức, tiêu biểu gồm Cây truy vấn tri thức (Knowledge Query Tree – KQT) [1], Cây truy vấn heuristic (Heuristic Query Tree – HQT) [12] và Cây truy vấn với phân tách dựa trên tri thức (Query Tree with Knowledge-based Splitting – QTKS) [13]. KQT phân chia miền ID thẻ đã biết thành nhiều khoảng, đầu đọc lần lượt truy vấn các thẻ trong từng khoảng ID, qua đó giới hạn số lượng thẻ phản hồi đồng thời. HQT xây dựng cây quyết định cho tập thẻ đã biết và liên tục chọn bit truy vấn "hiệu quả nhất" cho mỗi bước dựa trên việc duyệt cây quyết định. QTKS sử dụng cơ sở dữ liệu tri thức để xây dựng cây truy vấn và áp dụng các kỹ thuật rút gọn, phân giải trong quá trình nhận dạng.

Bài báo này tập trung phân tích các giao thức cây truy vấn dựa trên tri thức trong hai kịch bản phân bố thẻ: phân bố đồng nhất (ngẫu nhiên, đều) và phân bố không đồng nhất (phân bố Gauss). Các đóng góp chính gồm: Phân tích và so sánh lý thuyết các giao thức cây truy vấn dựa trên tri thức hiện có; Cài đặt và đánh giá hiệu suất các giao thức này với hai mô hình phân bố thẻ đồng nhất và không đồng nhất.

Phần còn lại của bài báo được tổ chức như sau. Phần 2 trình bày các nghiên cứu liên quan, bao gồm giao thức QT và các biến thể dựa trên tri thức của nó, cụ thể là KQT, HQT, QTKS và QTKS có ước lượng số thẻ (QTKS_TE), kèm theo phân tích và so sánh lý thuyết. Phần 3 mô tả chi tiết việc cài đặt và đánh giá hiệu suất các giao thức cây truy vấn dựa trên tri thức trong hai trường hợp phân bố thẻ. Cuối cùng, Phần 4 đưa ra kết luận và định hướng nghiên cứu tiếp theo.

2 Giao thức QT và các biến thể dựa trên tri thức của QT

Trong bài báo này, chúng tôi tập trung nghiên cứu giao thức cây truy vấn và các biến thể giao thức cây truy vấn dựa trên tri thức. Do cách tiếp cận chủ yếu là phân tích thuật toán của giao thức, các thuật ngữ "giao thức cây truy vấn" và "thuật toán cây truy vấn" được sử dụng thay thế lẫn nhau trong toàn bài. Phần dưới đây trình bày chi tiết phân tích giao thức cây truy vấn (QT) và các biến thể dựa trên tri thức của nó.

2.1 Giao thức QT

Giao thức cây truy vấn (giao thức QT), hiện là giao thức chống xung đột phổ biến nhất. Câu lệnh truy vấn của giao thức QT có chuỗi tiền tố có thể thiết lập động. Để phản hồi đầu đọc, chỉ những thẻ có định danh (ID) khớp với chuỗi tiền tố (prefix) mới cung cấp phần ID còn lại

(remainder) của nó sau khi loại bỏ chuỗi tiền tố. Các thẻ xung đột sẽ được đưa vào một nhóm mới để được truy vấn lại, vì vậy quá trình này được lặp lại cho đến khi chỉ còn một thẻ trả lời. Nếu đầu đọc phát hiện tín hiệu xung đột thì số "0" và "1" sẽ được bổ sung trong chuỗi tiền tố. Thuật toán QT được trình bày ở Thuật toán 1.

Thuật toán 1. QT

```
Input:   Các thẻ trong vùng đọc;
Output:  ID_list;                    // Danh sách ID các thẻ đọc thành công
         idle, success, collision;   // Các biến đếm số khe nhàn rỗi, thành công và xung đột
idle ← 0; success ← 0; collision ← 0;
ID_list ← ∅;
L ← ∅;                               // Danh sách các thẻ trả lời tại mỗi vòng truy vấn
Push(Q,1); Push(Q,0);                // Initialize Q
while (Q != ∅)
    q ← Pop(Q);                      // Pop up chuỗi truy vấn q từ ngăn xếp S
    L ← Broadcast(q);                // Broadcast prefix q đến các thẻ;
    if |L| == 0 then                 // Không có thẻ nào trả lời (idle)
        idle ← idle + 1;
    elseif |L| == 1 then             // Chỉ một thẻ trả lời (success)
        success ← success + 1;
        ID_list.append(q+L[0]);      // Thêm thẻ đọc thành công (q+L[0]) vào ID_list
      else
          collision ← collision + 1;
          Push(Q, q1);               // Tách nhánh bằng cách push chuỗi
          Push(Q, q0);               // Truy vấn q1 và q0 vào ngăn xếp S
      end
    end
end
```

Độ dài và phân bố ID thẻ ảnh hưởng đáng kể đến hiệu suất của giao thức QT. Khi nhiều thẻ chia sẻ tiền tố dài, đầu đọc cần thêm các tín hiệu phân tách để chia tập thẻ thành các nhóm nhỏ hơn, do đó nhiều cải tiến cho QT đã được đề xuất. Nhìn chung, các giao thức chống xung đột dạng cây có thời gian xử lý và cấu trúc cây tương đối phức tạp, nhưng bù lại đạt thông lượng cao.

2.2 Giao thức cây truy vấn dựa trên tri thức

Giao thức KQT

Giao thức KQT là biến thể của giao thức QT, khai thác tri thức về số lượng thẻ cần nhận dạng và phân bố ID của chúng [1]. Ý tưởng chính của KQT là hạn chế số thẻ phản hồi đồng thời để xử lý xung đột hiệu quả hơn. Ban đầu, đầu đọc sử dụng số lượng thẻ đã biết, ký hiệu N_{exp}, để chia miền ID thẻ thành N_{exp} khoảng. Mỗi khoảng thứ i được ký hiệu $\mathrm{I}_i = [SP_{\mathrm{i}}, EP_{\mathrm{i}}]$, trong đó SP_{i} và EP_{i} lần lượt là điểm đầu và điểm cuối của khoảng ID; về mặt lý thuyết, mỗi khoảng có thể chỉ chứa một thẻ.

Sau đó, đầu đọc tiến hành nhận dạng trong N_{exp} chu kỳ, mỗi chu kỳ tương ứng với một khoảng I_i. Trong chu kỳ thứ i, đầu đọc truy vấn cặp giá trị $(SP_{\mathrm{i}}, EP_{\mathrm{i}})$; tất cả các thẻ có ID thuộc khoảng I_i sẽ phản hồi ID của mình. Nếu chỉ có một thẻ phản hồi, thẻ đó được nhận dạng thành công và đầu đọc chuyển sang chu kỳ tiếp theo. Nếu xảy ra xung đột, đầu đọc áp dụng quy trình QT cơ bản trên tập thẻ trong khoảng đó. Quá trình này được lặp lại cho đến khi xử lý xong tất cả các khoảng. Do tri thức sử dụng trong KQT dựa trên một tập ID thẻ cụ thể, giao thức này cho hiệu suất cao hơn QT trong trường hợp các ID thẻ phân bố tập trung trong một vùng nhất định. Thuật toán KQT được minh họa trong Thuật toán 2.

Thuật toán 2. KQT

reader procedure

```
Input:   N_exp;                          // Số lượng thẻ ước tính
         g(.);                           // Hàm giả định xác suất rời rạc của phân phối ID thẻ
Output:  ID_list;                        // Danh sách ID các thẻ đọc thành công
         idle, success, collision;       // Các biến đếm số khe nhàn rỗi, thành công và xung đột
1.  makeIntervals(g(.), N_exp, L, R);    // Chia N_exp khoảng từ L đến R theo hàm g(.)
2.  idle ← 0; success ← 0; collision ← 0;
3.  ID_list ← ∅;                         // Danh sách các thẻ trả lời tại mỗi vòng truy vấn
4.  for (i = 0; i < N_exp; i++)
5.      broadcast(act, sp[i], ep[i]);    // Broadcast lệnh activation đến các thẻ trong (sp[i], ep[i])
6.      receiveAnswer;
7.      if (channelStatus == 1) then     // Chỉ một thẻ trả lời (success)
8.          success ← success + 1;
9.          ID_list.append(receivedID);  // Thêm thẻ đọc thành công (receivedID) vào ID_list
10.     elseif (channelStatus > 1) then  // Có nhiều hơn 1 thẻ trả lời (collision)
11.         collision ← collision + 1;
12.         broadcast(sub, sp[i], h);    // Broadcast lệnh subtract đến các thẻ trong (sp[i], h)
```

```
13.         QTProcedure();                  // Thực hiện giao thức QT
14.     else                                // Không có thẻ nào trả lời (idle)
15.         idle ← idle + 1;
16.     end
17. end
```

tag procedure

```
1.  identified ← false;
2.  while (not identified)
3.      receive(command, sp, ep);
4.      if (command == act) ∧ (sp <= myID <= ep) then     // Các thẻ trong (sp, ep) nhận lệnh activation
5.          send myID;
6.          if (receivedAck) then
7.              identified ← true;
8.          end
9.      end
10.     if (command == sub) ∧ (sp <= myID <= ep) then     // Các thẻ trong (sp, ep) nhận lệnh subtract
11.         tmpMyID ← last h bits of myID - sp;
12.         receive(prefix);
13.         offset ← prefix.length;
14.         if (prefix == tmpMyID[0..offset-1]) then
15.             send tmpMyID;
16.         end
17.         if (receivedAck) then
18.             identified ← true;
19.         end
20.     end
21. end
```

Giao thức HQT

Giao thức HQT mở rộng giao thức QT nhằm khai thác tập ID thẻ đã biết để nhận dạng các thẻ xuất hiện [2]. HQT xây dựng một cây quyết định dựa trên tri thức và tại mỗi bước sẽ lựa chọn bit truy vấn "hiệu quả nhất" cho lần truy vấn kế tiếp. Đóng góp thứ nhất của HQT là cơ chế heuristic: giao thức sử dụng tập thẻ đã biết như các ứng viên, từ đó lựa chọn bit truy vấn tối ưu

và truy vấn thẻ một cách linh hoạt. Đóng góp thứ hai là tính thực tiễn: nhiều thẻ đã biết có thể đồng thời được sử dụng làm ứng viên cho các thẻ xuất hiện trong vùng đọc của đầu đọc. Đóng góp thứ ba là tính hiệu quả: HQT có khả năng mở rộng tốt với độ phức tạp thời gian O(log n). Thuật toán HQT được minh họa trong Thuật toán 3.

Thuật toán 3. HQT

Input: Các thẻ có thể trong cơ sở dữ liệu; Các thẻ xuất hiện trong vùng đọc;

Output: Cây quyết định T; Tập các ID thẻ được nhận dạng trong vùng đọc;

reader procedure

tree construction procedure

```
1.  T = null; dif_t = null
2.  do InsertNode (T, tags) for all possible tags in database        // Chèn mọi thẻ trong CSDL
3.  procedure InsertNode(T, tags):
4.      if T is null then
5.          T = create_new_node(tags)                                // Tạo node mới với tags
6.          directly end procedure
7.      else
8.          entropy ← calculate_entropy(tags)                        // Tính entropy của node gốc
9.          if entropy > 0 then
10.             best_attribute ← find_best_attribute(tags)           // Chọn thuộc tính tốt nhất
11.             left_tags, right_tags = split_tags(tags, best_attribute)  // Chia tags
12.             T.left ← InsertNode(T.left, left_tags)               // Đệ quy nhánh trái
13.             T.right ← InsertNode(T.right, right_tags)            // Đệ quy nhánh phải
14.         end
15.     end
16. end
```

identification procedure

```
1.  procedure TraceTree(n: the node, p: prefix, d: the depth)        // Thủ tục TraceTree
2.      decide the most efficient bit to query the tags              // Chọn bit hiệu quả nhất
3.      send query based on the most efficient bit
4.      receive response from tags                                   // Nhận phản hồi thẻ
5.      if collision then                                            // Có nhiều hơn 1 thẻ trả lời
6.          TraceTree(left sub-tree of n, q, d+1)                    // Đệ quy cây con trái
7.          TraceTree(right sub-tree of n, q, d+1)                   // Đệ quy cây con phải
```

8. **elseif** readable **then** // Chỉ một thẻ trả lời
9. Retrieve the tag ID
10. **end**

Trên thực tế, HQT mô hình hóa bài toán chống xung đột dựa trên tri thức như một bài toán phân loại dữ liệu. Giao thức sử dụng entropy và thuật toán ID3 để xây dựng cây quyết định, trong đó bit truy vấn ở mỗi nút được chọn dựa trên độ lợi thông tin lớn nhất. Cây truy vấn heuristic được phát triển cho đến khi mỗi nhánh chỉ còn một thẻ. Trong quá trình nhận dạng, đầu đọc luôn chọn bit có entropy lớn nhất để truy vấn và lặp lại cho đến khi nhận dạng hết các thẻ xuất hiện.

Giao thức QTKS

Giao thức QTKS giả định đầu đọc có cơ sở dữ liệu chứa ID của toàn bộ các thẻ có thể có [13]. Trước tiên, dựa trên tập ID này, đầu đọc xây dựng một cây truy vấn dựa trên tri thức, gọi là k-tree. Đây là cây nhị phân mà mỗi nút luôn có đúng hai nút con, trong đó mỗi nút lá tương ứng với một thẻ tiềm năng nhằm tránh các xung đột không cần thiết và giảm các khe nhàn rỗi. Trong giai đoạn nhận dạng, đầu đọc sử dụng k-tree để sinh ra các truy vấn dạng chuỗi bit thích hợp, từ đó xác định tất cả các thẻ xuất hiện trong vùng đọc. QTKS áp dụng thêm hai kỹ thuật: (i) rút gọn để lược bỏ các truy vấn dư thừa trên k-tree, và (ii) phân giải cặp đôi, cho phép truyền đồng thời hai ID trong cùng một khe và bỏ qua các truy vấn dư thừa khi chỉ còn hai thẻ có thể. Nhờ vậy, QTKS rút ngắn đáng kể thời gian nhận dạng so với KQT và HQT. Thuật toán QTKS được minh họa trong Thuật toán 4; ngoài ra trong phiên bản mở rộng được sử dụng trong nghiên cứu này, thuật toán QTKS được bổ sung thêm một bước ước lượng số thẻ (Tag Estimation) tại Dòng 7–9 của quá trình nhận dạng. Bước ước lượng số thẻ dựa trên số khe xung đột/nhàn rỗi trong các truy vấn gần nhất để ước lượng nhanh số thẻ đang hoạt động. Thông tin ước lượng này được dùng để điều chỉnh mức phân tách hoặc chiến lược chia nhánh, giúp giảm các truy vấn không cần thiết khi số thẻ thực tế nhỏ hơn tập thẻ tiềm năng. Giao thức QTKS có tích hợp bước ước lượng số thẻ này được ký hiệu là QTKS_TE (QTKS with Tag Estimation). Đây là một cải tiến nhỏ nhưng mang tính thực tiễn, nhằm nâng cao hiệu suất trong các môi trường có số thẻ biến động hoặc phân bố không đồng nhất.

Thuật toán 4. QTKS và QTKS_TE

Input: Các thẻ có thể trong cơ sở dữ liệu; Các thẻ xuất hiện trong vùng đọc;

Output: Cây k-tree T đã được xây dựng; Tập các ID thẻ được nhận dạng trong vùng đọc;

reader procedure

k-tree construction procedure

1. T = null; dif_t = null

```
2.  do InsertNode (T, tid) for each tid of all possible tags in database   // Chèn từng thẻ trong CSDL
3.  procedure InsertNode(T: tree, tid: tag ID)
4.      if T is null then
5.          create new n as the root node of T with nS = tid              // Tạo nút gốc T
6.          directly end procedure
7.      else
8.          n ← the root node of T; l ← the length of nS                  // n: gốc T, l: |nS|
9.          k ← the length of common prefix nS and tid                    // Độ dài tiền tố chung
10.         if k < l then
11.             m ← new node values and sub-nodes of n                    // m: nút mới từ n
12.             mS ← different part of nS against tid                     // mS là phần khác nhau
13.             nS ← common prefix nS against tid                         // nS là tiền tố chung
14.             nV ← nV + 1 if n is leaf node                             // Tăng nV nếu n là lá
15.             clear the child nodes of n                                // Xóa các nút con của n
16.             if first bit of mS is '0' then                            // Bit đầu của mS = '0'
17.                 set m as left sub-node of n                           // Đặt m là nút con trái của n
18.             elseif first bit of mS is '1' then                        // Bit đầu của mS = '1'
19.                 set m as right sub-node of n                          // Đặt m là nút con phải của n
20.             end
21.         end
22.         nV ← nV + 1
23.         dif_t ← different part of tid against nS                      // dif_t là phần khác nhau
24.         if first bit of dif_t is '0' then                             // Bit đầu dif_t = '0'
25.             InsertNode(left sub-tree of n, dif_t)                     // Đệ quy cây con trái
26.         elseif first bit of dif_t is '1' then                         // Bit đầu dif_t = '1'
27.             InsertNode(right sub-tree of n, dif_t)                    // Đệ quy cây con phải
28.         end
29.     end
30. end
```

identification procedure

```
1.  level ← -1                                                            // Khởi tạo level
2.  Transmit the command starting a frame                                 // Bắt đầu khung
```

```
3.  do TraceTree(root, '', 0)                                  // Duyệt cây từ gốc
4.  Transmit the command terminating a frame                   // Kết thúc khung
5.  procedure TraceTree(n: the node, p: prefix, d: the depth)  // Thủ tục TraceTree
6.      q ← p + nS
7.      if d < level and n has sub-tree then                   // d < level và có cây con
8.          TraceTree(left sub-tree of n, q, d+1)              // Nhánh trái
9.          TraceTree(right sub-tree of n, q, d+1)             // Nhánh phải
10.     else
11.         Transmit q
12.         Receive tag response and detect the status         // Nhận phản hồi thẻ
13.         if collision and n has sub-tree then               // Nếu có xung đột & có cây con
14.             TraceTree(left sub-tree of n, q, d+1)          // Nhánh trái
15.             TraceTree(right sub-tree of n, q, d+1)         // Nhánh phải
16.         elseif readable then                               // Chỉ một thẻ trả lời
17.             Retrieve the tag ID
18.         end
19.         level ← d if (no collision and level < 0)          // Cập nhật level lần đầu
20.     end
21. end
```

tag procedure

```
1.  Receive message m from the reader                          // Nhận m từ reader
2.  while r != the command terminating a frame do              // Chưa kết thúc khung
3.      if r is a query then                                   // r là truy vấn
4.          if prefix(ID) == r then Transmit ID                // Nếu prefix(ID)=r thì gửi ID
5.      end
6.      Receive message r from the reader                      // Nhận r mới
7.  end
```

Một đánh giá tổng quan các thuật toán cây truy vấn dựa trên tri thức được tóm tắt trong Bảng 1.

Bảng 1. Đánh giá các thuật toán dựa trên tri thức KQT, HQT và QTKS

Thuật toán	Giả thiết tri thức	Cách thực hiện	Đánh giá hiệu quả
KQT	- Biết số thẻ N_{exp} và chia miền ID thẻ thành N_{exp} khoảng; - Biết phân bố thẻ (đồng nhất hoặc không đồng nhất) để việc chia khoảng tương ứng là đều hoặc theo hàm phân bố.	- Áp dụng thuật toán QT độc lập trên từng khoảng đã chia.	- Nếu số thẻ được biết/ước lượng chính xác và việc chia khoảng hợp lý (mỗi khoảng xấp xỉ một thẻ), KQT đạt hiệu quả nhận dạng rất cao.
HQT	- Biết phân bố thẻ và miền ID của tập thẻ ứng viên.	- Xây dựng cây quyết định dựa trên phân bố các thẻ đã biết; - Chọn bit truy vấn "hiệu quả nhất" theo heuristic; - Sử dụng thuật toán ID3 với độ lợi thông tin làm tiêu chí chọn bit.	- Việc sử dụng entropy để chọn bit tốt nhất giúp hầu như loại bỏ các khe nhàn rỗi.
QTKS và QTKS_TE	- Biết trước miền phân bố ID trong cơ sở dữ liệu thẻ để xây dựng cây truy vấn; việc truy vấn một tập con thẻ sau đó dựa hoàn toàn trên cây này.	- Xây dựng cây truy vấn dựa trên cơ sở dữ liệu ID thẻ đã biết; - Thực hiện truy vấn một tập con thẻ dựa trên cấu trúc cây truy vấn đã xây.	- Quá trình truy vấn thẻ được rút ngắn và hiệu quả hơn; - Khi có thể ước lượng được số thẻ cần truy vấn, hiệu quả truy vấn được cải thiện đáng kể.

Sau đây là phần đánh giá thực nghiệm đối với các giao thức (thuật toán) cây truy vấn dựa trên tri thức.

3 Mô phỏng và phân tích kết quả

Các thuật toán QT, KQT, HQT, QTKS và QTKS_TE được cài đặt và mô phỏng trên máy tính cá nhân sử dụng bộ xử lý Intel(R) Core(TM) i7-12700 @ 2.10 GHz, RAM 24 GB, với ngôn ngữ lập trình Python 3.11.4. Để đánh giá hiệu suất nhận dạng thẻ của các giao thức dựa trên cây, chúng tôi sử dụng các tiêu chí sau:

Số khe xung đột (collision slots) và số khe nhàn rỗi (idle slots);

Hiệu suất truy vấn (Query efficiency), được định nghĩa là tỷ lệ giữa số khe thành công và tổng số khe truy vấn:

$$Query_{efficiency} = \frac{S}{S + C + I} \tag{1}$$

trong đó S là số khe thành công (success slots), C là số khe xung đột (collision slots) và I là số khe nhàn rỗi (idle slots).

Theo mô hình năng lượng trong [14], tổng năng lượng cho một vòng nhận dạng là $E = E_s + E_c + E_i$; với $E_s = S \times e_s$, $E_c = C \times e_c$, $E_i = I \times e_i$. Hiệu suất năng lượng được định nghĩa là

$$EE = \frac{E_s}{E_s + E_c + E_i} = \frac{S \times e_s}{S \times e_s + C \times e_c + I \times e_i} \tag{2}$$

Khi các năng lượng trung bình xấp xỉ nhau ($e_s \approx e_c \approx e_i$), ta có $EE \approx \frac{S}{S+C+I}$, tức $EE \approx$ Query_efficiency. Do đó, tối ưu hiệu suất truy vấn đồng thời tối ưu hiệu suất năng lượng.

Chúng tôi xem xét hai trường hợp phân bố thẻ: phân bố đồng nhất (ngẫu nhiên, đều) và phân bố không đồng nhất (Gaussian) được mô tả bởi:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2}{2\sigma^2}) \tag{3}$$

trong đó μ là giá trị trung bình, σ là độ lệch chuẩn, π là hằng số PI và exp(·) là hàm mũ.

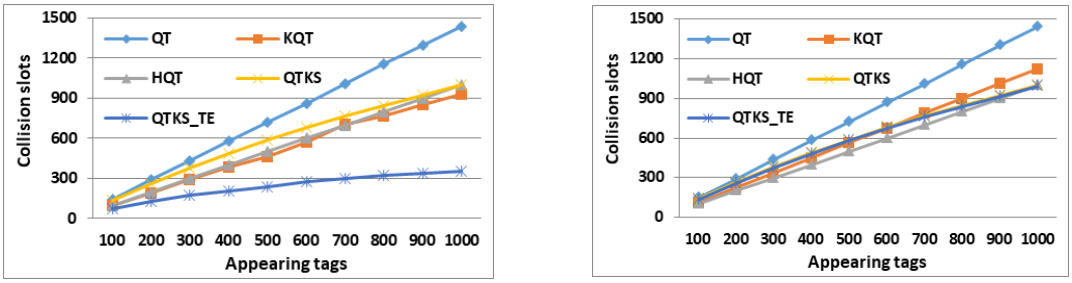
Các tham số mô phỏng được tóm tắt trong Bảng 2.

Bảng 2. Các tham số mô phỏng

STT	Tham số	Giá trị
1	Chiều dài ID thẻ	48 bit
2	Tổng thẻ có thể	1000
3	Số thẻ có thể xuất hiện	100-1000
4	Phân bố thẻ $g(\cdot)$	Đồng nhất và Gauss

3.1 So sánh số khe xung đột

Hình 1 so sánh số khe xung đột (collision slots) của các thuật toán QT, KQT, HQT, QTKS và QTKS_TE theo số lượng thẻ xuất hiện khác nhau trong vùng đọc, với hai dạng phân bố thẻ: đồng nhất (Hình 1(a)) và không đồng nhất (Hình 1(b)).



Hình 1. So sánh số khe xung đột của các thuật toán QT, KQT, HQT, QTKS và QTKS_TE: (a) phân bố đồng nhất; (b) phân bố không đồng nhất

Kết quả cho thấy các thuật toán cây truy vấn dựa trên tri thức luôn tạo ra số khe xung đột thấp hơn so với QT, nhờ khai thác "tri thức" về cơ sở dữ liệu thẻ, miền phân bố và số lượng thẻ mà đầu đọc có được, qua đó tránh được nhiều nút xung đột khi duyệt cây. Trong nhóm này, QTKS_TE đạt số khe xung đột thấp nhất, đặc biệt hiệu quả khi số thẻ xuất hiện xấp xỉ số thẻ trong cơ sở dữ liệu, do vừa xây dựng cây truy vấn trên cơ sở dữ liệu thẻ, vừa ước tính được số thẻ cần truy vấn; mức "tri thức" càng cao thì truy vấn càng hiệu quả.

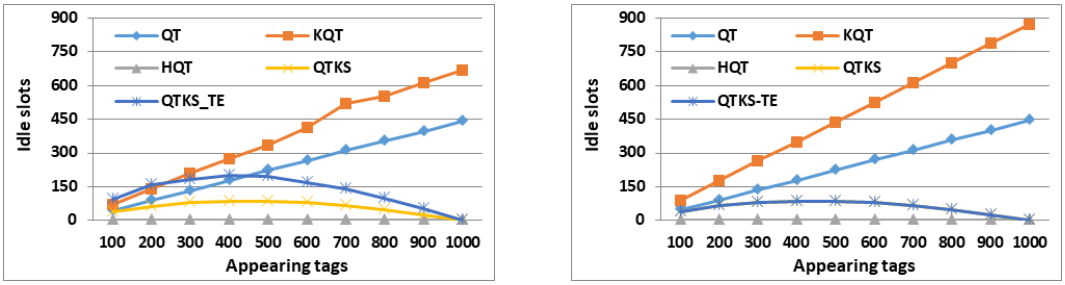
Tuy nhiên, khi so sánh hai dạng phân bố, QTKS_TE có thể tạo ra số khe xung đột khá cao trong một số trường hợp phân bố không đồng nhất. Hiệu suất của QTKS_TE phụ thuộc chặt chẽ vào cấu trúc cây được xây dựng và vị trí thực tế của các thẻ xuất hiện trong vùng đọc; nếu phân bố thẻ buộc thuật toán phải duyệt qua hầu hết các nút của cây để truy vấn hết thẻ, số khe xung đột của QTKS_TE sẽ khó được giảm xuống.

3.2 So sánh số khe nhàn rỗi

Đối với các cải tiến của thuật toán QT được công bố trước đây, việc giảm số khe xung đột thường đi kèm với việc tăng số khe nhàn rỗi (idle slots). Tuy nhiên, với các thuật toán dựa trên tri thức, số khe nhàn rỗi có xu hướng biến thiên khác nhau, như thể hiện ở Hình 2.

KQT có số khe nhàn rỗi cao hơn QT. Về bản chất, KQT thực hiện QT độc lập trên từng khoảng được chia. Nếu việc chia khoảng chính xác (mỗi khoảng xấp xỉ có một thẻ), KQT đạt hiệu quả tốt; ngược lại, khi một khoảng không chứa thẻ nào hoặc chứa nhiều hơn một thẻ, KQT lại hoạt động tương tự QT trong khoảng đó, dẫn đến xuất hiện thêm các khe nhàn rỗi so với QT.

Đối với HQT, do lựa chọn bit truy vấn "tốt nhất" dựa trên entropy, hầu như không xuất hiện khe nhàn rỗi.



Hình 2. So sánh số khe nhàn rỗi của các thuật toán QT, KQT, HQT, QTKS và QTKS_TE: (a) phân bố đồng nhất; (b) phân bố không đồng nhất

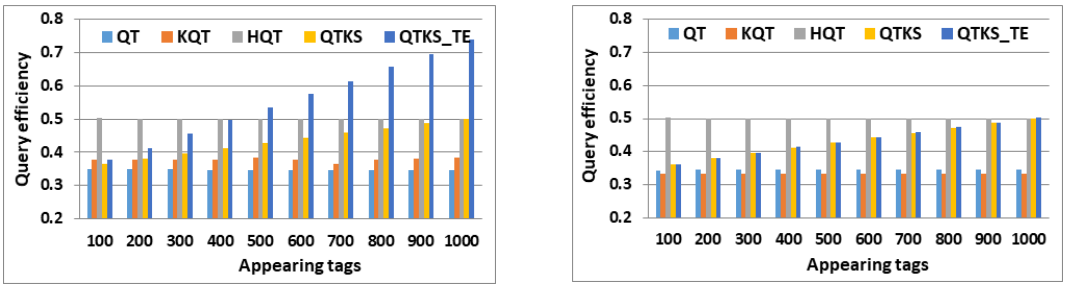
Đối với QTKS và QTKS_TE, khi số thẻ xuất hiện trong vùng đọc tăng, số khe nhàn rỗi giảm. Nguyên nhân là cây truy vấn được xây dựng sẵn từ cơ sở dữ liệu thẻ đã biết; nếu số thẻ cần truy vấn ít, quá trình duyệt cây có thể đi qua nhiều nút mà không gặp thẻ, tạo ra khe nhàn rỗi. Khi số thẻ xuất hiện tăng lên (tiệm cận số thẻ trong cơ sở dữ liệu dùng để xây cây), xác suất mỗi nhánh dẫn tới một thẻ được nhận dạng thành công cao hơn, do đó số khe nhàn rỗi giảm, như trong Hình 2.

Trong trường hợp phân bố thẻ không đồng nhất, QTKS và QTKS_TE cho số khe nhàn rỗi xấp xỉ nhau. Việc QTKS_TE ước lượng được số thẻ thực tế không giúp giảm đáng kể số khe nhàn rỗi, bởi với dạng phân bố không đồng nhất được xem xét, thuật toán vẫn phải duyệt gần như toàn bộ cây để truy vấn hết toàn bộ thẻ.

3.3 So sánh hiệu suất truy vấn

Hiệu suất truy vấn được xác định theo (1), tức là tỉ số giữa số khe thành công và tổng số khe cần thiết để truy vấn hết các thẻ trong vùng đọc. Hình 3 trình bày so sánh hiệu suất của các thuật toán QT, KQT, HQT, QTKS và QTKS_TE đối với hai dạng phân bố thẻ: đồng nhất (Hình 3(a)) và không đồng nhất (Hình 3(b)).

Đối với QT, KQT và HQT, hiệu suất hầu như không thay đổi đáng kể khi số thẻ xuất hiện trong vùng đọc tăng. Nguyên nhân là KQT về bản chất chỉ thực hiện QT trên từng khoảng đã chia, nên hiệu suất của KQT chủ yếu phụ thuộc vào độ chính xác của việc chia khoảng, ít bị ảnh hưởng bởi số thẻ cần truy vấn; do đó, hiệu suất của QT và KQT gần như không biến động khi số thẻ tăng. Với HQT, số khe nhàn rỗi gần như bằng 0, nên theo (1), việc số khe thành công và số khe xung đột cùng tăng khi số thẻ xuất hiện tăng cũng không làm cải thiện đáng kể hiệu suất truy vấn.



(a) (b)

Hình 3. So sánh hiệu suất truy vấn của các thuật toán QT, KQT, HQT, QTKS và QTKS_TE: (a) phân bố đồng nhất; (b) phân bố không đồng nhất

Ngược lại, hiệu suất của QTKS và QTKS_TE phụ thuộc rõ rệt vào số thẻ trong vùng đọc. Như đã phân tích, khi số thẻ xuất hiện tăng, số khe nhàn rỗi trong QTKS và QTKS_TE giảm, làm giảm mẫu số trong (1) và do đó làm tăng hiệu suất truy vấn. Riêng với QTKS_TE, nhờ số khe xung đột cũng giảm khi số thẻ xuất hiện tăng (Hình 1), hiệu suất truy vấn của QTKS_TE tăng vượt trội so với các thuật toán còn lại.

Ở trường hợp phân bố thẻ không đồng nhất, hiệu suất của QTKS và QTKS_TE không được cải thiện đáng kể so với HQT. Lý do là trong cấu hình phân bố này, QTKS và QTKS_TE không giảm được nhiều cả số khe xung đột lẫn số khe nhàn rỗi, nên theo (1), hiệu suất truy vấn của chúng khó có thể tăng thêm.

4 Kết luận và hướng phát triển

Hiệu suất truy vấn thẻ có ảnh hưởng trực tiếp đến hiệu quả hoạt động của hệ thống RFID, trong đó đầu đọc đóng vai trò then chốt. Khi đầu đọc không có tri thức về số lượng thẻ, miền ID thẻ và phân bố thẻ trong vùng đọc, số khe xung đột và khe nhàn rỗi thường cao, làm suy giảm hiệu suất truy vấn. Trong nhiều ứng dụng thực tế, miền ID và phân bố thẻ có thể được biết trước và được xem như "tri thức" của hệ thống. Dựa trên tri thức này, các giao thức truy vấn thẻ truyền thống có thể được cải tiến để nâng cao hiệu suất.

Bài báo đã phân tích và đánh giá các giao thức cây truy vấn dựa trên tri thức thông qua mô phỏng với hai mô hình phân bố thẻ: đồng nhất (uniform) và không đồng nhất (non-uniform). Mặc dù giả thiết tri thức và cách khai thác tri thức của từng giao thức là khác nhau, kết quả cho thấy hiệu suất truy vấn phụ thuộc rõ rệt vào lượng tri thức sẵn có: tri thức càng nhiều, hiệu suất truy vấn càng cao. Tuy nhiên, các giao thức cây truy vấn dựa trên tri thức cũng bị ảnh hưởng mạnh bởi phân bố thẻ, đặc biệt trong trường hợp phân bố không đều. Điều này cho thấy bản thân phân bố thẻ cũng là một dạng tri thức mà nếu được tích hợp vào thiết kế giao thức, hiệu suất truy vấn có thể tiếp tục được cải thiện cho từng dạng phân bố cụ thể.

Hướng phát triển tiếp theo của nhánh nghiên cứu này là xác định và mô hình hóa thêm các dạng tri thức khác có thể xuất hiện trong thực tế (chẳng hạn tri thức theo thời gian, theo khu vực, theo nhóm ứng dụng) và khai thác chúng để thiết kế hoặc cải tiến các giao thức truy vấn thẻ. Việc thẻ RFID thụ động ngày càng được triển khai rộng rãi trong nhiều lĩnh vực ứng dụng mở ra thêm nhiều kịch bản thực tế, qua đó tạo cơ hội tiếp tục phát triển và hoàn thiện các giao thức cây truy vấn dựa trên tri thức.

Lời cảm ơn

Nghiên cứu này được thực hiện trong khuôn khổ đề tài khoa học cấp Đại học Huế, mã số DHH2024-01-220.

Nguyễn Đức Nhật Quang được tài trợ bởi Chương trình học bổng đào tạo thạc sĩ, tiến sĩ trong nước của Quỹ Đổi mới sáng tạo Vingroup (VINIF), mã số VINIF.2024.TS.072.

Tài liệu tham khảo

1. Y. Zhang and D. Zhao, "A New Dynamic Frame Slotted ALOHA-Algorithm for Anti-Collision in Radio Frequency Identification Systems," Proceedings of the 2008 China–Japan Joint Microwave Conference, 2008, pp. 502–504, doi: 10.1109/CJMW.2008.4772479.
2. J.-B. Eom and T.-J. Lee, "Accurate Tag Estimation for Dynamic Framed-Slotted ALOHA in Radio Frequency Identification Systems," Communications Letters, Institute of Electrical and Electronics Engineers, vol. 14, no. 1, pp. 60–62, Jan. 2010, doi: 10.1109/LCOMM.2010.01.091378.
3. J. Park, M. Y. Chung, and T. J. Lee, "Identification of Radio Frequency Identification Tags in Framed-Slotted ALOHA with Tag Estimation and Binary Splitting," Proceedings of the HUT-ICCE 2006 First International Conference on Communications and Electronics, PART 1, no. 5, 2006, pp. 368–372, doi: 10.1109/CCE.2006.350795.
4. C.-N. Yang, Y.-C. Kun, C.-Y. Chiu, and Y.-Y. Chu, "A New Adaptive Query Tree on Resolving Radio Frequency Identification Tag Collision," in Proceedings of the 2010 Institute of Electrical and Electronics Engineers International Conference on Radio Frequency Identification Technology and Applications, June 2010, pp. 153–158, doi: 10.1109/RFID-TA.2010.5529878.
5. F. Yang, L. Zhao, H. Chen, and S. Hao, "A Novel Query Tree Anti-Collision Algorithm for Radio Frequency Identification," Applied Computational Electromagnetics Society Journal, vol. 34, no. 3, pp. 490–496, 2019.
6. N. Zhao, L. Zhang, L. Lei, and S. Cai, "Dynamic Query Tree Anti-Collision Protocol for Radio Frequency Identification Systems," Proceedings of the International Conference on Parallel and Distributed Systems, December 2019, pp. 778–781, doi: 10.1109/ICPADS47876.2019.00115.
7. W. Chen, "Performance Comparison of Binary Search Tree and Framed ALOHA Algorithms for Radio Frequency Identification Anti-Collision," Transactions on Communications, Institute of Electronics, Information and Communication Engineers, vol. E91-B, no. 4, pp. 1168–1171, April 2008, doi: 10.1093/ietcom/e91-b.4.1168.
8. Y. C. Lai, L. Y. Hsiao, and B. S. Lin, "Optimal Slot Assignment for Binary Tracking Tree Protocol in Radio Frequency Identification Tag Identification," Transactions on Networking, Institute of Electrical

and Electronics Engineers and Association for Computing Machinery, vol. 23, no. 1, pp. 255–268, 2015, doi: 10.1109/TNET.2013.2295839.

9. F. Yang, Y. Yang, H. Chen, S. Ren, and L. Zhao, "A Low Complexity Anti-Collision Algorithm for Radio Frequency Identification Using Query Tree," in Proceedings of the 2018 Cross Strait Quad-Regional Radio Science and Wireless Technology Conference, 2018, vol. 10110001, pp. 6–7, doi: 10.1109/CSQRWC.2018.8455646.
10. M. M. Samsami and N. Yasrebi, "Novel Radio Frequency Identification Anti-Collision Algorithm Based on the Monte–Carlo Query Tree Search," Wireless Networks, vol. 27, no. 1, pp. 621–634, 2021, doi: 10.1007/s11276-020-02466-1.
11. M. A. Bonuccelli, F. Lonetti, and F. Martelli, "Exploiting Identification Knowledge for Tag Identification in Radio Frequency Identification Networks," in Proceedings of the Fourth Association for Computing Machinery Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor and Ubiquitous Networks, October 2007, pp. 70–77, doi: 10.1145/1298197.1298210.
12. J. Sung, D. Kim, T. Kim, and J. Choi, "Heuristic Query Tree Protocol: Use of Known Tags for Radio Frequency Identification Tag Anti-Collision," Transactions on Communications, Institute of Electronics, Information and Communication Engineers, vol. E95-B, no. 2, pp. 603–606, 2012, doi: 10.1587/transcom.E95.B.603.
13. Y.-C. Lai, R. Jayadi, S.-C. Huang, and Y.-H. Chen, "Query Tree with Knowledge-Based Splitting for Radio Frequency Identification Tag Identification," in Proceedings of the 2018 International Conference on Sensors, Signal and Image Processing, October 2018, pp. 24–28, doi: 10.1145/3290589.3290592.
14. V.-H. Le, D.-N.-Q. Nguyen, and V.-M.-N. Vo, "An Improved Q-Learning Algorithm Integrated into the Aloha Anti-Collision Protocol for Energy-Efficient Radio Frequency Identification Systems," The International Arab Journal of Information Technology, vol. 22, no. 1, pp. 51–59, January 2025, doi: 10.34028/iajit/22/1/5.