

Analysis of Tree Algorithms for RFID Arbitration

Don R. Hush* and Cliff Wood**

*EECE Dept, Univ.of New Mexico, Albuquerque, NM 87131

**Micron Communications, 3176 Denver Way, Boise, ID 83707

Abstract

Radio Frequency Identification (RFID) systems have emerged as a viable and affordable alternative for tagging and/or labeling small to large quantities of items. The arbitration problem is that of identifying all ‘tags’ present in the current environment (i.e reading the labels). This is a multiaccess communication problem in which contention for the communication medium is transient in nature. Important measures of performance include the time required to identify the tags, and the power consumed by the tags. The first of these is proportional to the number of time slots needed to complete the arbitration process, and the second is proportional to the total number of times the tags are required to transmit during this process. We analyze a family of *Tree Search* algorithms with respect to these two measures. Previous analysis on the average number of time slots [1, 4, 5] reveals a linear dependence on the number of tags, m . We extend this work by developing expressions for the decomposition of these slots into three basic types: zero, one and multiple reply slots. Original expressions are also developed for the total number of tag replies. We show that on average this number is $\Theta(m \log m)$, and that this number can be reduced to $\Theta(m)$ when prior knowledge of m is available.

1 Introduction

A Radio Frequency Identification (RFID) system consists of a master device called an *interrogator*, and a number of slave devices called *tags*. The interrogator communicates with tags via an RF link, so that all interrogator transmissions are “heard” (simultaneously) by all tags. To target a command at a specific tag (i.e. to initiate point-on-point communication) the interrogator must send a Tag Identification (TagID) along with the command. At start-up (or in a new or changing environment), the TagIDs are not known. The *arbitration problem* is to identify all tags in

the current environment (i.e. determine their TagIDs). Once this is accomplished, point-to-point communication can proceed as dictated by the interrogator.

In a broader context, RFID systems are a type of *multiaccess* communication system with the following characteristics.

1. The distance between the interrogator and the tags is relatively short (generally several meters), so the total packet transmission time is dominated by the packet size and the baud rate. Propagation delays are negligible.
2. RFID environments may include a large number of tags ('transmitting units'), and are often *dynamic* (i.e. tags are swapped out frequently). This essentially mandates the use of *random* access methods for contention resolution.
3. A tag never communicates without being prompted by the interrogator. This type of master/slave relationship is somewhat unique to RFID. Furthermore, contention for the communication medium is short-lived, since once the tags have been identified, the interrogator can communicate with them in a point-to-point fashion. This is in contrast to a typical multiaccess system where the transmitting units operate more independently and contention is an ongoing problem. Thus, RFID arbitration is a *transient* process as opposed to the *steady-state* process more commonly encountered in multiaccess systems.
4. The capability of a tag is severely limited by practical restrictions on size, power and cost. The lifetime of a tag can often be measured in terms of the total number of transmissions it can perform before battery power is lost.

The two most important measures of system performance in RFID arbitration are the *total time* required to arbitrate a complete set of tags, and the *total power* consumed by the tags during the process. This is in contrast to the measures of *throughput* and *packet delay* for typical multiaccess systems.

The *Tree Search* (or *splitting*) method introduced by Capetanakis for use in conventional multiaccess systems [1] can also be applied to RFID arbitration. This paper performs a transient analysis of this, and related methods. We develop expressions for the average number of time slots required to complete the arbitration process, and decompose it into expressions for the number of collisions, single replies and zero replies. We also develop expressions for the total number of tag replies (which is proportional to the total power consumption). It is assumed throughout that the interrogator can distinguish between the following three events without error: Z = zero tag transmissions, S = a single tag transmission, and C = a collision (two or more tags are transmitting).

2 Splitting Methods: Tree Search Algorithms

Tree Search algorithms work by recursively splitting groups of colliding tags into B (disjoint) subsets. These subsets get smaller and smaller until they contain at most one tag. In practice this is accomplished by having each colliding tag flip a B -sided coin to determine its subset. The first subset of tags replies immediately in the next time slot, while the others wait until the current subset is completely resolved. Once a subset is completely resolved, waiting subsets are resolved in a ‘first-in last-out’ order.

Algorithms of this type can be viewed as a *tree search*. Each split moves the algorithm one level deeper in the tree. In Capetanakis’ original splitting algorithm a *binary tree* was used (i.e. $B = 2$) [1, 2, 6]. An example is shown in Figure 1 for the case where $m = 4$. Nodes in the tree are labeled according to their activities: W = Wait, C = Collision, S = Single Reply and Z = Zero Reply. The algorithm starts at the root node. Solid branches show the portion of the tree spawned by the splitting process up to the point where the first successful transmission is received (in the lower right leaf). Dashed branches show the additional portion of the tree that is explored after the first tag is found. Activities shown in parenthesis are the result of the action taken once the node’s waiting period has expired (i.e. once the search returns to that position in the tree). Successful transmissions (S replies) occur in order from right to left in the shaded nodes in this example. The sequence of coin flips made by the tags in this example are:

First Tag Identified: 1,1,1
 Second Tag Identified: 1,1,0
 Third Tag Identified: 0,1
 Fourth Tag Identified: 0,0

The complete arbitration process uses a total of 9 time slots (one for each node in the tree).

3 Time Slot Analysis

In this section we develop expressions for the following quantities:

$$\begin{aligned} \bar{t}_{TS}(m) &= \text{Average (total) number of Time Slots} \\ \bar{c}_{TS}(m) &= \text{Average number of Collisions} \\ \bar{z}_{TS}(m) &= \text{Average number of Zero replies} \end{aligned}$$

The first of these can be decomposed into the remaining two as follows,

$$\bar{t}_{TS}(m) = \bar{c}_{TS}(m) + \bar{z}_{TS}(m) + m \tag{1}$$

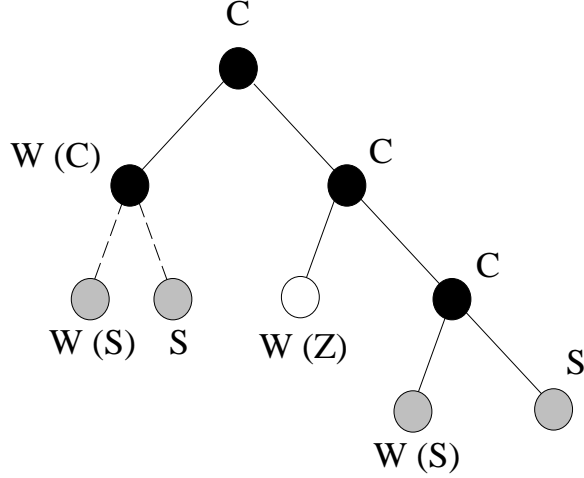


Figure 1: Example of Splitting algorithm with $m = 4$.

where the ‘ $+m$ ’ accounts for the m time slots corresponding to Single replies.

Suppose we interpret the sequence of bits flipped by a tag during the arbitration process as a fractional binary number in the interval $[0, 1)$. In the previous example these numbers would be .111, .110, .010 and .000. For purposes of analysis we can pretend that these numbers are computed ahead of time (i.e. prior to the arbitration process). With this view the arbitration process can be thought of as a *search* for a set of m randomly generated numbers in the interval $[0, 1)$. To identify one of the numbers, it is sufficient to probe an interval in which it alone resides. The individual steps (time slots) in a B -ary Tree Search algorithm can be viewed as probes over subintervals of size B^{-L} (where L is the level of the tree in which the probe is made; the root node is considered to reside at level 0). The interrogator makes at most B^L probes at level L (i.e. the probe intervals are nonoverlapping). If the random numbers are uniformly distributed, then the probability that exactly k out of m of them fall within a particular subinterval of size B^{-L} is given by the binomial distribution

$$P(k/m, L) = \binom{m}{k} p^k (1-p)^{m-k} \quad (2)$$

where $p = B^{-L}$. With this we obtain the following expressions for the probabilities of events Z , S and C resulting from an independent probe at level L in the tree:

$$Z : \quad P(0/m, L) = (1-p)^m = (1-B^{-L})^m \quad (3)$$

$$S : \quad P(1/m, L) = mp(1-p)^{m-1} = mB^{-L}(1-B^{-L})^{m-1} \quad (4)$$

$$\begin{aligned} C : \quad P(k > 1/m, L) &= 1 - P(0/m, L) - P(1/m, L) \\ &= 1 - (1-p)^m - mp(1-p)^{m-1} \\ &= 1 - (1-B^{-L})^m - mB^{-L}(1-B^{-L})^{m-1} \end{aligned} \quad (5)$$

Now, let $q_{Li/m}$ be the probability that the i^{th} node at level L is visited by the Tree Search algorithm. Clearly, for $L = 0$, $q_{0i/m} = q_{00/m} = 1$ regardless of m (i.e. the root node is *always* visited). In subsequent levels of the tree a node is visited only when its parent experiences a collision. Let $\beta_{Li/m}$ be the probability of collision in the i^{th} node at level L . When the random numbers are uniformly distributed, the probability of collision is the same for all nodes at the same level, and is given by (5),

$$\beta_{Li/m} = \beta_{L/m} = P(k > 1/m, L) = 1 - (1 - B^{-L})^m - mB^{-L}(1 - B^{-L})^{m-1} \quad (6)$$

With this we have

$$q_{Li/m} = q_{L/m} = \begin{cases} 1 & L = 0 \\ \beta_{L-1/m} & L > 0 \end{cases} \quad (7)$$

Now the average (expected) number of nodes visited by the Tree Search algorithm is determined by summing $q_{Li/m}$ over all nodes in all levels,

$$\begin{aligned} \bar{t}_{TS}(m) &= \sum_{L=0}^{\infty} \sum_{i=0}^{B^L-1} q_{Li/m} \\ &= 1 + \sum_{L=1}^{\infty} B^L q_{L/m} \\ &= 1 + \sum_{L=1}^{\infty} B^L \beta_{L-1/m} \\ &= 1 + B \sum_{L=0}^{\infty} B^L \beta_{L/m} \end{aligned} \quad (8)$$

Substituting from (6) gives

$$\bar{t}_{TS}(m) = 1 + B \sum_{L=0}^{\infty} B^L \left[1 - (1 - B^{-L})^m - mB^{-L}(1 - B^{-L})^{m-1} \right] \quad (9)$$

The expression for the average number of collisions, $\bar{c}_{TS}(m)$, is only slightly different, and is obtained by summing $\beta_{Li/m}$ over all nodes at all levels of the tree,

$$\begin{aligned} \bar{c}_{TS}(m) &= \sum_{L=0}^{\infty} \sum_{i=0}^{B^L-1} \beta_{Li/m} \\ &= \sum_{L=0}^{\infty} B^L \beta_{L/m} \end{aligned} \quad (10)$$

Comparing this with the expression for $\bar{t}_{TS}(m)$ in (8) gives

$$\bar{c}_{TS}(m) = \left(\frac{1}{B} \right) (\bar{t}_{TS}(m) - 1) \quad (11)$$

Thus, approximately $1/B$ of the nodes visited in a B -ary Tree Search are collisions.

Given expressions for $\bar{t}_{TS}(m)$ and $\bar{c}_{TS}(m)$, and the fact that m time slots are used for single replies, the number of zero replies is simply

$$\bar{z}_{TS}(m) = \left(\frac{B-1}{B} \right) \bar{t}_{TS}(m) - m + \frac{1}{B} \quad (12)$$

Expressions similar to (9) for $\bar{t}_{TS}(m)$ have been developed by numerous authors [1, 4, 5]. It is well-known that $\bar{t}_{TS}(m)$ is, for all practical purposes, linear in m

[5]. In reality it has been shown that $\bar{t}_{TS}(m)$ has small oscillations around the linear approximation [4]. These oscillations are so small however, that they can usually be ignored. For example, with $B = 2$ the oscillations in $\bar{t}_{TS}(m)/m$ can be detected only if we evaluate the expression out to the seventh decimal place [4]. Consequently, we treat $\bar{t}_{TS}(m)$ as a linear function of m . Since $\bar{t}_{TS}(m)$ is linear, equations (11) and (12) tell us that $\bar{c}_{TS}(m)$ and $\bar{z}_{TS}(m)$ are also linear.

A summary of the linear approximations for $\bar{t}_{TS}(m)$ and its decompositions are shown in the table below for the first four values of B

B	$\bar{t}_{TS}(m)$	$\bar{c}_{TS}(m)$	$\bar{z}_{TS}(m)$
2	$2.885m$	$1.443m$	$0.442m$
3	$2.730m$	$0.910m$	$0.820m$
4	$2.882m$	$0.720m$	$1.162m$
5	$3.113m$	$0.622m$	$1.491m$

The numbers in this table can be obtained using the rigorous mathematical methods in [5, 4], or by curve-fitting data produced from a numerical evaluation of the expressions in (9), (11) and (12).

Interestingly enough, the best performance (fewest time slots) is achieved when $B = 3$. In addition, the performance for $B = 4$ is nearly *identical* to that for $B = 2$. In fact, from the decomposition we see that the decrease in the number of C nodes is countered by an (almost equal) increase in the number of Z nodes as we move from $B = 2$ to $B = 4$. This can be predicted from the expressions in (11) and (12). In general these expressions tell us that as B is increased, the number of C nodes becomes a smaller fraction of the total (roughly $1/B$) while the number of Z nodes becomes a much larger fraction, approaching $(B - 1)/B$ for large B .

Even though $\bar{t}_{TS}(m)$ is the same for $B = 2$ and 4, the reduction in the number of collisions in the $B = 4$ case results in fewer total tags replies, which translates into lower overall power consumption on the part of the tags. We expand on this point in the next section.

4 The Total Number of Tag Replies

The total number of tag replies, $\bar{r}_{TS}(m)$, is proportional to the total power consumed by the tags during the arbitration process. Unlike the quantities in the previous section however, it is not a linear function of m , rather it is $\Theta(m \log m)$. First we give an intuitive explanation for this result, and then provide a proof. Intuitively, the Tree Search algorithm is most likely to resolve a majority of the tags at level $L^* \approx \log_B(m)$ in the tree. So the number of tag replies in each of the $(\log_B(m) - 1)$ levels leading

up to L^* will be approximately m . Thus, the total number of replies should be on the order of $m \log_B m$.

We start by developing an expression for $\bar{s}_{TS}(m, n)$, the average number of tags resolved by the Tree Search algorithm, *up to and including* level n . Let $u_{ni/m} = P(1/m, n)$ be the probability of a Single reply at node i in level n of the tree. Then $\bar{s}_{TS}(m, n)$ is given by

$$\bar{s}_{TS}(m, n) = \sum_{i=0}^{B^n-1} u_{ni/m} = B^n u_{n/m} = m(1 - B^{-n})^{m-1} \quad (13)$$

This is simply the sum of $u_{ni/m}$ over all B^n nodes at level n . The justification for this expression is simple. If a tag is resolved at a level higher than n in the tree, then it will also be resolved at level n . By probing all nodes at level n and counting the Single replies we are accounting for all S nodes visited by the Tree Search algorithm up to and including those at level n . Note that for large n , $\bar{s}_{TS}(m, n)$ is very close to m (as expected).

We can develop an exact expression for $\bar{r}_{TS}(m)$ as follows. At the 0^{th} level of the tree (i.e. the root node) there are always m replies. The number of replies at level $L > 0$ is equal to m minus the number of tags resolved in levels 0 through $L - 1$. So the total number of replies is given by

$$\bar{r}_{TS}(m) = m + \sum_{L=1}^n (m - \bar{s}_{TS}(m, L - 1)) \quad (14)$$

where $\bar{s}_{TS}(m, L - 1)$ is the number of tags resolved up to and including level $L - 1$. Substituting for $\bar{s}_{TS}(m, L - 1)$ from (13) gives

$$\bar{r}_{TS}(m) = m + m \sum_{L=0}^{n-1} (1 - (1 - B^{-L})^{m-1}) \quad (15)$$

In an extended version of this paper we prove that [3]

$$m + m \log_B(m) \leq \bar{r}_{TS}(m) \leq \left(2 + \frac{1}{\ln(B)}\right) m + m \log_B(m - 1)$$

The table below shows curve-fitting approximations for $\bar{r}_{TS}(m)$ for the first four values of B .

B	$\bar{r}_{TS}(m)$
2	$m \log_2 m + 2.333m = 1.44m \ln m + 2.333m$
3	$m \log_3 m + 2.030m = 0.91m \ln m + 2.030m$
4	$m \log_4 m + 1.921m = 0.72m \ln m + 1.921m$
5	$m \log_5 m + 1.867m = 0.62m \ln m + 1.867m$

Note the substantial reduction in the number of tag replies as B is increased. For example, with $B = 4$ the total number of replies is nearly one-half that for $B = 2$ (assuming large m).

5 Optimal Tree Search

When the number of tags is known, the Tree Search algorithm can be optimized as follows. The interval $[0, 1)$ is divided into m nonoverlapping subintervals of equal size, and then a separate B -ary Tree Search is employed for each subinterval. This can be viewed as a *single* search applied to a tree whose root node has m children, and all subsequent nodes have B children. The root node is skipped in this search since it is known to produce a collision. The optimality of this algorithm is discussed in [1, 4].

In the analysis of this method we consider the m ‘root’ nodes for the individual tree searches to be at level 0. Children of these nodes are at level 1, and their children are at level 2, etc. Probabilities of events Z , S and C for probes in these searches are given by equations (3)-(5) with

$$p = \left(\frac{1}{m}\right) B^{-L} \quad (16)$$

Following the development in Section 3 the average number of nodes visited for one of the m searches is given by

$$1 + B \sum_{L=0}^{\infty} B^L \gamma_{L/m}$$

where

$$\begin{aligned} \gamma_{L/m} &= 1 - (1 - p)^m - mp(1 - p)^{m-1} \\ &= 1 - \left(1 - \left(\frac{1}{m}\right) B^{-L}\right)^m - B^{-L} \left(1 - \left(\frac{1}{m}\right) B^{-L}\right)^{m-1} \end{aligned} \quad (17)$$

The average number of nodes visited is the same for each search, so the total for all m searches is

$$\bar{t}_{OP}(m) = m + mB \sum_{L=0}^{\infty} B^L \gamma_{L/m} \quad (18)$$

It can be shown that, like $\bar{t}_{TS}(m)$, this quantity is essentially linear in m [1, 4].

Expressions for the decomposition of $\bar{t}_{OP}(m)$ are relatively straightforward. The average number of collisions is given by

$$\begin{aligned} \bar{c}_{OP}(m) &= m \left[\sum_{L=0}^{\infty} B^L \gamma_{L/m} \right] \\ &= \left(\frac{1}{B}\right) (\bar{t}_{OP}(m) - m) \end{aligned} \quad (19)$$

With this, the average number of zero replies is given by

$$\bar{z}_{OP}(m) = \left(\frac{B-1}{B}\right)(\bar{t}_{OP}(m) - m) \quad (20)$$

A summary of the linear approximations for $\bar{t}_{OP}(m)$ and its decompositions are shown in the table below for the first four values of B .

B	$\bar{t}_{OP}(m)$	$\bar{c}_{OP}(m)$	$\bar{z}_{OP}(m)$
2	$2.337m$	$0.669m$	$0.669m$
3	$2.431m$	$0.477m$	$0.954m$
4	$2.642m$	$0.411m$	$1.231m$
5	$2.881m$	$0.376m$	$1.505m$

Note that $B = 2$ provides the best performance here, in contrast to $B = 3$ for the Tree Search in Section 3. Note also that the $B = 2$ performance is a significant improvement over the performance for all values of B in Section 3. In general, larger values of B lead to fewer collisions, but this is overshadowed by the increase in zero replies. It is interesting to note that the number of collisions and zero replies is essentially identical for the $B = 2$ case. Also, in comparison with the Tree Search in Section 3, the number of collisions with optimized Tree Search is always significantly less (by roughly a factor of two), while the number of zero replies is always slightly greater.

The number of tags resolved up to and including level L in the optimized Tree Search is given by

$$\begin{aligned} \bar{s}_{OP}(m, L) &= m \left[B^L m p (1-p)^{m-1} \right] \\ &= m \left(1 - \frac{B^{-L}}{m} \right)^{m-1} \end{aligned} \quad (21)$$

With this, the total number of tag replies is given by

$$\begin{aligned} \bar{r}_{OP}(m) &= m + \sum_{L=1}^{\infty} [m - \bar{s}_{OP}(m, L-1)] \\ &= m + m \sum_{L=0}^{\infty} \left[1 - \left(1 - \frac{B^{-L}}{m} \right)^{m-1} \right] \end{aligned} \quad (22)$$

In an extended version of this paper we prove that [3]

$$(1+c)m \leq \bar{r}_{TS}(m) \leq (2+c)m$$

where c is a constant satisfying $0 < c < 1/\ln(B)$. This is a significant improvement over the $\Theta(m \log m)$ result in Section 4. The table below shows curve-fitting approximations for $\bar{r}_{OP}(m)$ for the first four values of B .

B	$\bar{r}_{TS}(m)$
2	$2.49m$
3	$2.08m$
4	$1.93m$
5	$1.86m$

Note that these results closely match those for the *linear terms* in the fits of $\bar{r}_{TS}(m)$ in Section 4.

6 Summary

Our results for $\bar{t}(m)$ tell us that without knowledge of m a 3-ary tree is optimal, and with knowledge of m a 2-ary tree is optimal. Furthermore, knowledge of m can improve the performance by as much as 19% with $B = 2$. In addition, the results for $\bar{r}(m)$ tell us that without knowledge of m a total of $\Theta(m \log m)$ tag replies are invoked by the algorithm, but with knowledge of m this is reduced to $\Theta(m)$. Also, generally speaking the total number of tag replies is less with larger values of B . This is true regardless of whether the algorithm is optimized for m or not.

References

- [1] J.I. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Transactions on Information Theory*, IT-25(5):505–515, 1979.
- [2] J.F. Hayes. An adaptive technique for local distribution. *IEEE Transactions on Communications*, COM-26:1178–1186, 1978.
- [3] D.R. Hush. A comparison of arbitration algorithms for rfid. Technical report, Micron Communications, Inc., Boise, ID, 1997.
- [4] M.A. Kaplan and E. Gulko. Analytic properties of multiple-access trees. *IEEE Transactions on Information Theory*, IT-31(2):255–263, 1985.
- [5] J.L. Massey. Collision resolution algorithms and random-access communications. In G. Longa, editor, *Multi-User Communication Systems*, pages 73–137. Springer-Verlag, New York, 1981.
- [6] B.S. Tsybakov and V.A. Mikhailov. Free synchronous packet access in a broadcast channel with feedback. *Problemy Peredachi Inform.*, (USSR), 14(4):32–59, 1978.