

# A Class of Algorithms for Batch Conflict Resolution with Multiplicity Estimation

Petar Popovski, Frank H.P. Fitzek and Ramjee Prasad  
Center for TeleInFrastructure (CTIF), Aalborg University  
Niels Jernes Vej 12, DK-9220 Aalborg, Denmark  
Email: {petarp|ff|prasad}@kom.aau.dk

## Abstract

The wireless connectivity, essential for pervasive computing, has ephemeral character and can be used for creating ad hoc networks, sensor networks, connection with RFID tags etc. The communication tasks in such wireless networks often involve an inquiry over a shared channel, which can be invoked for: discovery of neighboring devices in ad hoc networks, counting the number of active sensors in sensor networks, estimating the mean value contained in a group of sensors etc. Such inquiry solicits replies from possibly large number of terminals  $n$ . This necessitates the usage of algorithms for resolving batch conflicts with unknown conflict multiplicity  $n$ . In this paper we present a novel approach to the batch conflict resolution. We show how the conventional tree algorithms for conflict resolution can be used to obtain progressively accurate estimation of the multiplicity. We use the estimation to propose a more efficient binary tree algorithm, termed Estimating Binary Tree (EBT) algorithm. We extend the approach to design the Interval Estimation Conflict Resolution (IECR) algorithm. For  $n \rightarrow \infty$  we prove that the efficiency achieved by IECR for batch arrivals is identical with the efficiency that Gallager's FCFS algorithm achieves for Poisson packet arrivals. For finite  $n$ , the simulation results show that IECR is, to the best of our knowledge, the most efficient batch resolution algorithm reported to date.

## I. INTRODUCTION

### A. Motivation

The wireless connectivity is seen as essential part of the pervasive computing. Such connectivity often has ephemeral character and can be used for creating ad hoc networks [1], sensor networks [2], connection with RFID tags [3] etc. The seamless support for the interaction patterns in the pervasive computing poses myriad of challenges, many of which are related to the multiple access problems.

The communication tasks in wireless networks often involve an inquiry sent to the set of surrounding terminals over a shared channel. A device that we call *interrogator* starts an inquiry and it initially sends a probe message to solicit replies from the terminals within its radio range. Such terminals can be wireless sensors, RFID tags, wireless devices etc. All  $n$  devices that receive the solicitation probe transmit replies to the interrogator. Due to the considered short radio range, the replies arrive almost simultaneously at the interrogator and the interrogator perceives *conflict* or *collision*. In other words, there is a *batch arrival of the replies* and a batch conflict or collision of batch size  $n$ , where  $n$  is unknown to the interrogator. Here the value  $n$  is referred to as conflict *multiplicity*. The *collision resolution algorithm* is performed by the interrogator and the terminals in order to successfully transmit (resolve) the packets of the initial collision (batch). The interrogator transmits an acknowledgement to a terminal from which it receives the reply message successfully. In general, the interrogator does not need to receive a reply from all  $n$  terminals. The resolution can be partial (only fraction of terminals is resolved) or complete (all replies from the initial batch should be transmitted). In partial resolution, the interrogator may stop the procedure when enough replies are obtained. The criterion for *enough replies* depends on the application for which the considered inquiry is invoked. Here are some examples:

- **Approximate counting.** The interrogator needs to make estimation of number of active sensors  $n$  in a region. The estimated value  $\hat{n}$  should have predefined accuracy, e.g. limited variance. This requires partial resolution.

- **Mean value extraction.** Partial resolution can be also applicable here. The interrogator needs to find the mean value of a parameter, measured by a group of  $n$  sensors. If  $n$  is large, it may be not feasible to get reply from each sensor and then estimate the mean value. Rather, the interrogator calculates the mean value from the replies received so far and it can stop the procedure when the standard deviation of the estimate is within some limits.
- **Device discovery.** In this case the interrogator needs to receive reply from each of the  $n$  terminals, such that the collision resolution should be complete.

Note that the contention in all these cases has a transient nature, as opposed to the *standard* MAC protocols in which the contention is a steady-state phenomenon [3]. That is, here we consider how to resolve a single conflict of unknown multiplicity  $n$ , while in the MAC protocols we are interested in how to resolve the conflicts which continuously occur at the channel and thus represent a stationary state.

The performance of a collision resolution algorithm is evaluated through the time and message complexity. For complete resolution, the time complexity is expressed through the average duration of the *Batch Resolution Interval (BRI)*. We will use  $T_n$  to denote the duration BRI and it is defined as the average time elapsed from the point when the algorithm starts until the point at which the interrogator is sure that all  $n$  terminals are resolved. An efficient collision resolution algorithm should resolve collision of large (and unknown) multiplicity  $n$  using a short BRI. A convenient measure for time complexity is the *efficiency* of the algorithm [4], defined as  $\eta_n = \frac{n}{T_n}$ . In fact, the efficiency is directly related to the maximal achievable throughput when the collision resolution algorithm is applied in a MAC protocol. Concerning the message complexity, denote by  $M_n$  the average of the total number of messages used until all terminals are resolved. We are interested in the *average number of messages per terminal* defined as  $\mu_n = \frac{M_n}{n}$ . The terminals are usually battery-powered and thus energy-constrained, such that each transmission is precious and the minimization of the number of messages is of crucial importance. For the partial resolution, the performance measures also depend on the criterion for stopping the collision resolution. For example, it can be the average messages per terminal until the desired accuracy of the estimate of  $n$  is achieved.

### B. The Context of This Work

The collision resolution algorithms based on splitting tree have been introduced through the work of Hayes [5], Capetanakis [6] and Tsybakov and Mikhailov [7]. Besides the fact that the estimation of the multiplicity  $n$  can be important on its own right, the estimation of  $n$  can also be used to speed up the collision resolution process. Capetanakis in [6] observed that binary tree algorithms are most efficient for conflicts of small multiplicity and applied this observation to devise a *dynamic tree algorithm*. The dynamic tree algorithm is tailored to the Poisson arrivals of messages. In fact, the fastest collision resolution methods have been designed to resolve collision among messages with Poisson arrivals. Such is the Gallager's FCFS algorithm [8], which achieves efficiency 0.487. However, the collision resolution algorithms that are finely tuned to Poisson arrivals have poor performance for non-Poisson arrivals, example of which is the batch arrival [4].

The multiplicity estimation for batch resolution has been applied in [9] and [4]. In both works, the proposal is a hybrid algorithm that consists of two phases. The first phase is devoted to the estimation of multiplicity. After obtaining  $\hat{n}$ , the second phase starts. The unresolved terminals are randomly split into approximately  $\hat{n}$  groups and each group is resolved by using the basic tree algorithms. In particular, the partial resolution algorithm from [4] can asymptotically ( $n \rightarrow \infty$ ) achieve the same efficiency as the FCFS algorithm and has been the fastest known batch resolution algorithm to date. However, both [9] and [4] have an explicit phase for estimation of  $\hat{n}$  and the accuracy of such estimate depends only on the parameters that are chosen in advance. Comprehensive overviews of the collision resolution algorithms can be found in [10], [14], [15] and [17].

When CSMA channels are considered, the packet duration is longer than the time a terminal needs to sense idle channel, which accordingly changes the design of a conflict resolution algorithm. In [14] it is elaborated that the splitting tree paradigm is not efficient when the duration of the idle slot is *significantly* smaller than the packet duration, since the splitting tree is good in resolving collision and with short idle slots the collision may rarely occur, while not losing in time-efficiency. The *Sift* protocol, proposed in [16], addresses the problem of minimizing the collisions to obtain replies from a population of terminals over a CSMA channel. All  $n$  terminals start to simultaneously contend to send reply, such that they in effect create a batch conflict of unknown size  $n$ . The collisions are minimized by taking advantage of the small duration of the idle slot — the terminals start transmission with very small probability and increase it as the sequence of idle slots progresses. Sift is optimized to obtain the first reply (partial resolution 1 out of  $n$ ), but it does not adapt during the operation when  $k > 1$  out of  $n$  possible replies are needed.

The context of the work presented here can be summarized as follows. The collision resolution algorithms have been applied to the problem of multiple access and the most efficient algorithms assume Poisson arrivals of the packets. There have also been works where a collision resolution for batch packet arrival where the batch size (conflict multiplicity)  $n$  is unknown. In these works the authors use *one-shot estimation*  $\hat{n}$  of the multiplicity  $\hat{n}$  and use that estimate to speed up the collision resolution. Here we will propose a class of batch collision resolution algorithms in which the value  $\hat{n}$  is continuously estimated such that it becomes progressively accurate as more terminals are resolved. Therefore, our class of collision resolution algorithms has two generic applications:

- 1) To resolve all  $n$  terminals
- 2) To partially resolve only  $k$  out of  $n$  terminals until desired accuracy is achieved.

The paper is organized as follows. The system model is given in Section II. In Section III we will review the binary tree algorithm for collision resolution, point out the method to obtain multiplicity estimation and formulate the *Estimating Binary Tree (EBT)* algorithm. In Section IV we formulate the *Interval Estimation Conflict Resolution (IECR)* algorithm for slotted (non-CSMA) channels and CSMA channels. We show how IECR can be combined with the Sift protocol from [16] to produce adaptive collision resolution protocol. The results are presented and discussed in Section V. The final section summarizes the paper and gives directions for future work.

## II. SYSTEM MODEL

There are two scenarios depending on the role of the interrogator in the conflict resolution procedure. In the *probing* scenario, a terminal can transmit a packet only after receiving a probe packet from the interrogator. In fact, in the probing scenario the interrogator completely governs the conflict resolution. In the *non-probing* scenario, after initiating the conflict resolution, the interrogator role is to only to send acknowledgement after successfully receiving reply from a terminal. In addition, the interrogator can send a message to terminate the conflict resolution procedure.

We will consider two channel models: *slotted channel* and *carrier sense multiple access (CSMA)* channel [14]. Let  $T_P$  denote the duration of a packet and  $T_S$  denote the duration of a slot.

We first consider the slotted channel. A packet transmitted by a terminal has a duration of a single slot, i.e.  $T_P = T_S$ . Each packet transmission starts at the slot start. When  $k$  terminals transmit in the same slot  $t$ , then the interrogator perceives the channel in slot  $t$  as:

- *Idle slot (I)* if  $k = 0$  i.e. no terminal transmits.
- *Successful reception (S)* or *resolution* if  $k = 1$  i.e. only one terminal transmits.
- *Collision (C)* if  $k \geq 2$ . In this case, the messages of the terminals interfere destructively at the interrogator and error is detected.

In the model of slotted channel, the slot duration is the same regardless of whether the channel state is  $I, S$ , or  $C$ . Prior to the next slot ( $t + 1$ ), all nodes receive the feedback  $I, S$ , or  $C$ , indicating the state perceived by the interrogator in slot  $t$ . This is known as a ternary feedback model [10]. We neglect other sources of error besides the collision. For the channel state  $S$ , the feedback must be explicit i.e. an acknowledgement from the interrogator. For idle or collision, the feedback is implicit - if  $k = 0$  no terminal expects feedback and if  $k \geq 2$  the feedback is the absence of acknowledgement. The slot duration is enough to accommodate one packet sent by a terminal and the acknowledgement from the interrogator.

In the CSMA channel model the duration of the idle slots  $T_S$  is less than the duration of single packet or collision, both of which have duration  $T_P$ . An important parameter in the CSMA channel is  $\beta = \frac{T_S}{T_P} < 1$ .

The terminals should generate random bits to be used in the conflict resolution process. In some cases (e.g. when a terminal is RFID tag), the terminal cannot generate random bits. In such case the conflict resolution relies on the unique identity of each terminal. However, we will assume that the terminals are capable of generating random bits, but we will show the applicability of the proposed method to the cases where the resolution should rely on terminal identity.

## III. BINARY TREE ALGORITHMS AND MULTIPLICITY ESTIMATION

### A. Overview of Binary Tree Algorithms

We first introduce the basic variant of the recursive binary tree (BT) algorithm from [6]. There are  $n > 1$  terminals that attempt to transmit at slot  $t$ . After receiving feedback that a collision occurred in slot  $t$ , each of the conflicting

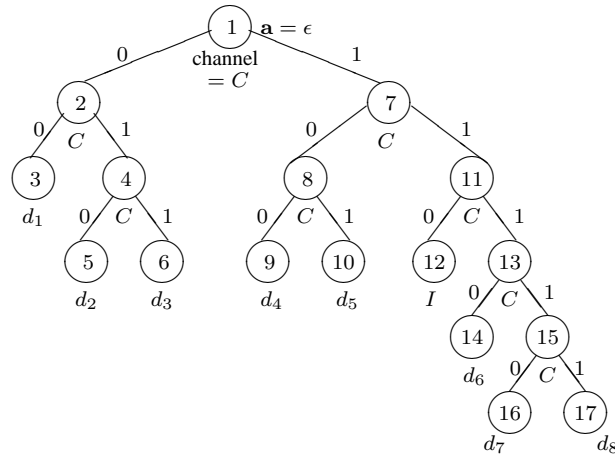


Fig. 1. An instance of the binary tree algorithm for  $n = 8$  terminals. The number denotes the slot in which a node is enabled. Below each node is the channel state in that slot. For channel state “single”,  $d_i$  denotes the resolved terminal.

terminals tosses a binary coin with possible outcomes 0 and 1. Let  $N(0)$  and  $N(1) = n - N(0)$  be the number of devices that tossed 0 and 1, respectively. The terminals that tossed 0 are allowed to transmit in  $(t + 1)$ . The terminals that tossed 1 should wait until all terminals that tossed 0 transmit successfully their packets. If  $N(0) = 0$  or  $N(0) = 1$ , then the slot  $(t + 1)$  is in state idle or single, respectively. The  $N(1)$  terminals transmit in  $(t + 2)$ . If  $N(0) > 2$ , then this procedure is invoked recursively for the terminals that tossed 0. A terminal that has transmitted successfully is denoted as *resolved*.

Fig. 1 depicts one possible evolution of the tree algorithm for initial conflict of multiplicity  $n = 8$ . The *level* of a tree node is the path length from that node to the root of the tree. Each tree node is uniquely associated with a string called *address*. The address of a tree node is determined by the tossing outcomes for the terminals belonging to that node. For example,  $d_2$  and  $d_3$  belong to the tree node with address 01, enabled in slot 4. The root has address  $\epsilon$  (empty string) and is at level 0. In fact, the initial conflict among the terminals occurs upon they receive probe with address  $\epsilon$  from the interrogator. The number of nodes with address  $\mathbf{a}$  is denoted by  $N(\mathbf{a})$ , while its level is denoted  $l(\mathbf{a})$ . Using the tree representation, we can say that an address (a node) is *enabled* in slot  $t$  if the terminals that belong to that node are allowed to transmit in  $t$ . In the basic variant, the tree nodes are enabled in a pre-order fashion.

Massey in [10] and Tsybakov and Mikhailov in [7] proposed a simple way to improve the basic variant for the binary tree algorithm. They noticed that there are some tree nodes that certainly contain more than 1 terminal and thus produce *certain collision* if the node is enabled. These nodes should be skipped during the traversal of the tree. For example, in Fig. 1, after the collision in slot 11 and an idle slot 12, it is clear that the enabling of the address 11 results in certain collision. Hence, after the idle slot 12, the terminals belonging to node 111 toss coin immediately and in slot 13 the enabled node is 1110. This algorithm will be referred to as Modified Binary Tree (MBT) algorithm.

The Clipped Binary Tree (CBT) algorithm has been independently introduced by several authors. It is identical to the MBT algorithm except that it is stopped (the tree is clipped) whenever two consecutive successful transmissions follow a conflict. CBT is a partial conflict resolution algorithm since not necessarily all nodes of an initial batch are resolved during the execution of CBT. The CBT algorithm was originally designed to deal with Poisson arrival process and it has been embedded into the First Come First Serve (FCFS) tree algorithm in [8]. The slight modifications of FCFS from [11] and [12] are the fastest known conflict resolution algorithms for Poisson arrivals. However, their performance for batch conflicts is poor and decreases as  $n$  increases. We will postpone the discussion of the FCFS algorithm until Section IV, where we will show how to design batch resolution algorithm which asymptotically achieves the same efficiency as FCFS.

The basic tree algorithms for complete resolution offer a natural way to get an multiplicity estimate  $\hat{n}$  with accuracy that improves as the number of resolved terminals increases. In the next subsection we show how to derive such estimation and we will further use that estimation method to speed up the conflict resolution.

### B. Real-Time Multiplicity Estimation with the Binary Tree Algorithms

Let the terminals use the following randomization for the conflict resolution. Before the initial attempt to transmit, each terminal generates a random real number, uniformly distributed in the interval  $[0, 1)$ . This random number is referred to as *token* and let  $r_i$  denote the token generated by the terminal  $d_i$ . Let  $\mathbf{b}_i = (b_{i1}b_{i2}b_{i3} \dots)$  be the binary representation of fractional part of  $r_i$ . Then each  $\mathbf{b}_i$  is an infinite string of 0s and 1s and:

$$r_i = r(\mathbf{b}_i) = \sum_{j=1}^{\infty} \frac{b_{ij}}{2^j} \quad (1)$$

The token  $r_i$  can be understood as an infinite reservoir for generating fair coin tosses, since each  $b_{ij}$  gets value 0 or 1 with probability 0.5. The terminal  $i$  belongs to the node with address  $\mathbf{a} = (a_1a_2a_3 \dots a_L)$ , where  $L = l(\mathbf{a})$ , if and only if  $b_{ij} = a_j$  for all  $j = 1 \dots L$ . In such case the string  $\mathbf{a}$  is a prefix of the string  $\mathbf{b}_i$ . We define the following mapping of the finite string  $\mathbf{a}$  to the interval  $[0, 1)$ :

$$r(\mathbf{a}) = \sum_{j=1}^L \frac{a_j}{2^j} \quad (2)$$

By definition we take  $r(\epsilon) = 0$ . To each address  $\mathbf{a}$  we associate unique interval  $[r(\mathbf{a}), p(\mathbf{a})) \in [0, 1)$ , where  $p(\mathbf{a})$  is determined from:

$$p(\mathbf{a}) = r(\mathbf{a}) + \frac{1}{2^{l(\mathbf{a})}} \quad (3)$$

When address  $\mathbf{a}$  is enabled, we can equivalently state that the interval  $[r(\mathbf{a}), p(\mathbf{a}))$  is enabled. Hence, having the address  $\mathbf{a}$  enabled, a terminal  $i$  is allowed to transmit if and only if:

$$r(\mathbf{a}) \leq r_i < p(\mathbf{a}) \quad (4)$$

Fig. 2 represents the example from Fig. 1 through the sequence of enabled intervals.

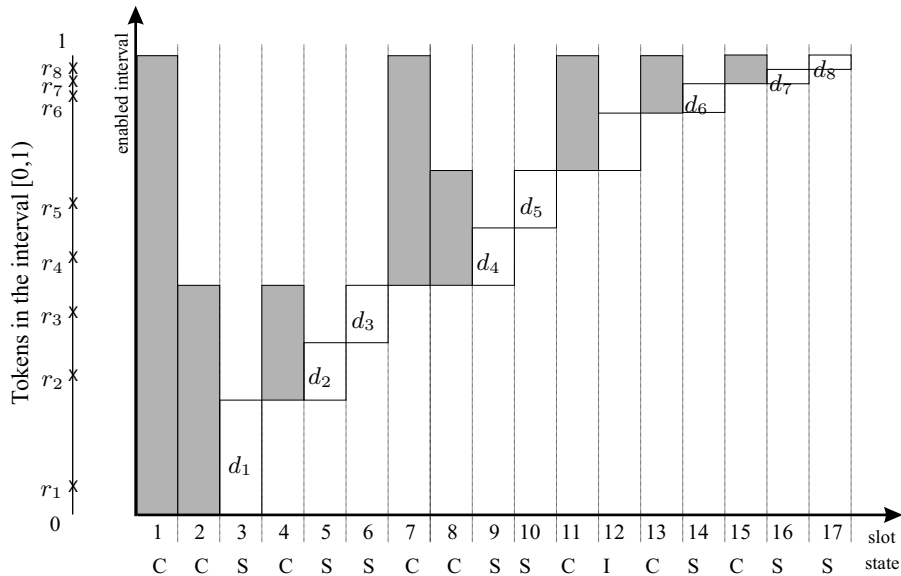


Fig. 2. Representation of the example of binary tree from Fig. 1 by using tokens and sequence of enabled intervals.

Now we show how the basic Binary Tree (BT) algorithm can be used to estimate the conflict multiplicity  $n$  with accuracy that increases as more terminals are resolved. The operation of the BT algorithm can be restated in terms of enabled intervals. Let the interval  $[r(\mathbf{a}), p(\mathbf{a})]$  be enabled, then the next enabled interval is  $[r(\mathbf{c}), p(\mathbf{c})]$  where:

- If  $N(\mathbf{a}) = 0$  or  $N(\mathbf{a}) = 1$ , then  $\mathbf{c}$  is the unique finite binary string that satisfies  $r(\mathbf{c}) = p(\mathbf{a})$ .
- If  $N(\mathbf{a}) > 1$ , then  $\mathbf{c}$  is obtained by appending 0 to  $\mathbf{a}$ , i.e.  $\mathbf{c} = (a_1 a_2 \dots a_L 0)$  and  $r(\mathbf{c}) = r(\mathbf{a})$ .

For the example on Figures 1 and 2, the interval  $[0.25, 0.5)$  is enabled in slot 4 and  $[0.25, 0.375)$  in slot 5. From such operation of the BT algorithm, it can be shown that when the node  $\mathbf{a}$  is enabled, all terminals with tokens  $r_i < r(\mathbf{a})$  must be already resolved. In slot 6 on Fig. 2, both  $d_1$  and  $d_2$  must be already resolved in slot 6, since address 011 is enabled in slot 6 with  $r(011) = 0.375$  and  $r_1 < r_2 < r(011)$ .

The address  $\mathbf{a}$  is resolved if  $N(\mathbf{a}) = 0$  or  $N(\mathbf{a}) = 1$ . If all terminals that belong to the subtree rooted at  $\mathbf{a}'$  are resolved, then we say that the subtree  $\mathbf{a}'$  is resolved. If the resolved address is  $\mathbf{a} = (a_1 a_2 \dots a_{L-1} 0)$ , then only the trivial subtree  $\mathbf{a}$  is resolved - a subtree that consists of node  $\mathbf{a}$ . If the resolved address is  $\mathbf{a} = (a_1 a_2 \dots a_{L-1} 1)$ , then there can be several resolved subtrees. A subtree  $\mathbf{a}'$  is resolved when the address  $\mathbf{a} = (a_1 a_2 \dots a_{L-1} 1)$  is resolved if and only if  $\mathbf{a}' = (a_1 a_2 \dots a_{L_1})$  and  $a_j = 1$  for all  $j > L_1$ . In slot 5 on Fig. 1 only the node with address 010 is resolved, while in slot 6 the resolved address is 011 and the resolved subtrees are 011, 01 and 0. All subtrees  $\mathbf{a}^1, \mathbf{a}^2 \dots$  that are resolved when the address  $\mathbf{a}$  is resolved have identical upper bound in the equivalent enabled interval  $p(\mathbf{a}^1) = p(\mathbf{a}^2) = \dots$

Let the address  $\mathbf{a}$  be resolved and let  $k(\mathbf{a})$  denote the number of tokens in the interval  $[0, p(\mathbf{a}))$ . In this case we can say that the interval of length  $p(\mathbf{a})$  is already resolved. Then the  $n - k(\mathbf{a})$  terminals that are still unresolved can observe the following: *From  $n$  tokens that are uniformly picked from the interval  $[0, 1)$ , where  $n$  is unknown,  $k(\mathbf{a})$*

tokens have been picked from the interval  $[0, p(\mathbf{a})]$ . With that observation, the terminals can make an estimation of the unknown multiplicity  $n$ . The probability that  $k(\mathbf{a}) = k$  tokens are picked from the interval  $[0, p]$ , conditioned that the total number of terminals is  $n$ , is given by:

$$P(N(\mathbf{a}) = k|n) = \binom{n}{k} p^k (1-p)^{n-k} \quad (5)$$

where  $p = p(\mathbf{a})$ . We assume that no prior distribution of  $n$  is known. Then the best that the terminals can do is to calculate a *maximum-likelihood* (ML) estimate of  $n$ , estimated when  $\mathbf{a}$  is resolved. The ML estimation  $\hat{n}$  is computed from:

$$\hat{n} = \arg \max_n P(N(\mathbf{a}) = k|n) \quad (6)$$

Let us denote:

$$\hat{n} = \frac{k(\mathbf{a})}{p(\mathbf{a})} = \frac{k}{p} \quad (7)$$

Since  $n$  is integer, the ML estimation of  $n$  from Eq. (6) is found to be:

$$\hat{n}_{ML} = \left\lfloor \frac{k}{p} \right\rfloor \quad (8)$$

Some properties of the estimate (7) are stated through the following lemma.

*Lemma 1:* Let there be  $k$  tokens in the interval  $[0, p]$  and let  $\hat{n} = \frac{k}{p}$ . Given the conflict multiplicity  $n$ , for any  $\delta > 0$ :

$$E[\hat{n}|n] = n \quad (9)$$

$$\text{Var}[\hat{n}|n] = \frac{n}{p} - n \quad (10)$$

$$P\left(\left|\frac{\hat{n}}{n} - 1\right| \geq \delta|n\right) \leq \frac{1-p}{\delta^2 np} \quad (11)$$

The equality in Eq. (9) follows from the fact that for given  $n$  we have  $E[k|n] = np$ . Also, the variance of  $k$ , conditioned on  $n$ , is  $\text{Var}[k|n] = np(1-p)$ . Hence:

$$\text{Var}[\hat{n}|n] = \text{Var}\left[\frac{k}{p}|n\right] = \frac{\text{Var}[k|n]}{p^2} = \frac{n}{p} - n \quad (12)$$

The inequality in Eq. (11) of the lemma follows from the Chebyshev inequality, see [13].

As the number of resolved terminals increases, the value of  $p$  also increases, and according to (11) the estimation  $\hat{n}$  becomes more accurate. To summarize, *we can say that without any modification, the BT (or the MBT) algorithm offers a way to estimate the unknown conflict multiplicity.* The relation (11) implies that in that estimation method we can trade off the accuracy with the consumption of resources – time and messages. An application of such approach can be e. g. an approximate counting.

The presented method of estimating  $n$  can be further utilized to design the *Estimating Binary Tree (EBT)* algorithm, in which the highly probable collisions are avoided.

### C. Estimating Binary Tree (EBT) algorithm

In formulating the EBT algorithm, we are adopting the heuristics similar to the one used in [9] and [4]. Having an estimate  $\hat{n}$ , the algorithms in [9] and [4] divide all terminals into  $\hat{n}$  groups, where each terminal chooses randomly and uniformly to which group it belongs. When  $n$  is perfectly known, such strategy maximizes the probability that a group will contain single device. After the division, the groups are resolved sequentially.

In our setting with enabled intervals, an equivalent heuristics can be stated as follows: If we know that an interval  $[0, 1)$  contains  $n$  nodes, what should be the length of the enabled subinterval, such that it contains single token with highest probability? If the length of the enabled subinterval is  $x$ , then the probability that it contains single token is:

$$P_1(x|n) = nx(1-x)^{n-1} \quad (13)$$

which is maximized when  $x = \frac{1}{n}$ . Since the enabled intervals have length  $2^{-L}$ , where  $L$  is an integer, the enabled node in the binary tree should be chosen at level:

$$L_{\max} = \lfloor \log_2 n \rfloor \quad (14)$$

Now let us assume that upon a termination of execution of the CBT algorithm, the interval  $[0, p(\mathbf{a}))$  have already been resolved and let it contain  $k$  nodes. The obtained estimation  $\hat{n}$  implies that there are  $\hat{n} - k$  tokens in the interval  $[p(\mathbf{a}), 1)$ , which has length  $1 - p$ . According to the heuristics described above, the enabled subinterval of  $[p(\mathbf{a}), 1)$  should be of length:

$$\frac{1-p}{\hat{n}-k} = \frac{1-p}{\frac{k}{p}-k} = \frac{p}{k} = \frac{1}{\hat{n}} \quad (15)$$

from which it follows:

$$L_{\max} = \lfloor \log_2 \hat{n} \rfloor \quad (16)$$

We use the ‘‘soft’’ estimate  $\hat{n}$  rather than  $\lfloor \hat{n} \rfloor$ , in order to avoid any information loss due to the  $\lfloor \cdot \rfloor$  operation. Enabling a node from level  $L_1 < L_{\max}$  results in collision with high probability. By skipping the nodes that lead to highly probable collision, the conflict resolution algorithm can reduce both the duration and the messages sent by the terminals.

The presented heuristics of choosing the length of the enabled interval is not optimized. The discussion of the optimization approaches is given in Section IV. An alternative heuristics can be obtained by asking the following question: For a given conflict multiplicity  $n$ , at which level  $L$  there is a highest probability that the BT algorithm resolves a terminal? The outcome of such heuristics is discussed in Appendix I.

Now we can formulate the EBT algorithm. It is based on the on the MBT algorithm, but it introduces the important changes, stated in Table I. The most straightforward interpretation of the EBT algorithm can be: (1) Run the CBT algorithm, (2) Find the level in the tree in which the resolution is most likely to occur, (3) Start a new CBT at the next node from level  $L_{\max}$ . Fig. 3 describes the EBT algorithm as a pseudocode, executed by the  $i$ -th terminal. The Fig. 4 depicts an instance of the EBT algorithm. After the initial conflict in slot 1, the CBT algorithm is started and it terminates in slot 6, when  $d_3$  is resolved. At that point the, estimate is  $\hat{n} = \frac{3}{0.5} = 6$ , such that the

```

a =  $\epsilon$ ; end= no; k = 0; Tx= no; collision_ind= no;
generate  $r_i$ ;
while(end== no)
  if( $r_i \in [r(\mathbf{a}), p(\mathbf{a})]$ );
    transmit; set Tx= yes;
  else
    set Tx= no;
  get feedback at the end of the slot;
  % current address is  $\mathbf{a} = (a_1 a_2 \dots a_L)$ 
  if(collision)
    set  $\mathbf{a} = (a_1 \dots a_L 0)$ ;
    collision_ind= yes;
  else
    if(single)
      k ++; collision_ind= no;
      if (Tx= yes) set end= yes; exit;
    if( $a_L = 0$ )
      set  $a_L = 1$ ;
      if(idle AND collision_ind= yes)  $a_{L+1} = 0$ ;
    else
       $\hat{n} = \frac{k}{p(\mathbf{a})}$ 
       $K = \log_2 \hat{n}$ ;
       $M = L$ ;
      while( $a_M == 1$ )  $M = M - 1$ ;
      for  $i = 1 \dots M - 1$  set  $c_i = a_i$ ;
      set  $c_M = 1$ ;
      if( $K < M$ )
        while( $c_K = 1$ )  $K++$ ;
        delete  $c_{K+1} \dots c_M$ ;
      if( $K > M$ )
        for  $i = M + 1 \dots K$ 
          set  $c_i = 0$ ;
      for  $i = 1 \dots K$   $a_i = c_i$ ;
      delete  $a_{K+1} \dots a_L$ ;

```

Fig. 3. The estimating binary tree (EBT) algorithm as run by the  $i$ -th node.

level of the next enabled node is  $\lfloor \log_2 6 \rfloor = 2$ . Hence, the node with  $\mathbf{a} = 1$  is skipped and the next enabled address in the tree is 10. Note that in this example the node 111 is not enabled since the mechanism of MBT is employed to avoid certain collision.

By skipping some nodes in the splitting tree, the EBT algorithm decreases both the average duration and the average number of messages. It may happen that for some  $n$  and some configuration of tokens  $\{r_i\}$ , the EBT algorithm takes more time slots than the MBT algorithm. However, these cases occur with very low probability and they are not affecting the improvement in the average duration.

In a single execution of the MBT algorithm each address in the tree can be enabled only once. This is not the case in the EBT algorithm. Refer to step 3 from Table I. If MBT is applied, then always the address  $\mathbf{c} = (c_1 c_2 \dots c_M)$  is enabled. If EBT is applied and in step 3 it is  $K < M$ , then EBT enables a node with address  $\mathbf{c}'$ , where  $\mathbf{c}'$  can be a prefix of  $\mathbf{c} = (c_1 c_2 \dots c_M)$ . Fig. 5 depicts the case when the address 010 is enabled two times in a single instance of EBT. In slot 5, the node 010 contains three terminals  $d_2, d_3, d_4$  and a collision occurs. Note from Fig. 3 that a resolved terminal leaves the EBT algorithm. After the resolution of  $d_3$  in slot 9, the string obtained in step 1 of Table I is  $\mathbf{c} = 01001$ . Step 2 of Table I outputs  $K = 3$  for the slot 10, such that the enabled node in slot 10 is again 010. However, at this moment 010 contains only the terminal  $d_4$ , since  $d_2$  and  $d_3$  are already resolved.

TABLE I  
THE RULES APPLIED IN THE EBT ALGORITHM

---

If the node  $\mathbf{a} = (a_1 a_2 \dots a_L)$  with  $a_L = 1$  is resolved, then the next enabled node is selected as follows:

- 1) Obtain the string  $\mathbf{c}$  as the unique finite string that satisfies  $r(\mathbf{c}) = p(\mathbf{a})$ . The string  $\mathbf{c} = (c_1 c_2 \dots c_M)$  satisfies  $l(\mathbf{c}) = M < L = l(\mathbf{a})$ .
  - 2) Estimate  $\hat{n} = \frac{k(\mathbf{a})}{p(\mathbf{a})}$  and set  $K = \log_2 \hat{n}$
  - 3) There are 3 cases here:
    - a) If  $K = M$  then the next enabled address is  $\mathbf{c} = (c_1 c_2 \dots c_M)$ .
    - b) If  $K < M$  then while  $c_K = 1$  and  $K \leq M$ , increase  $K$ . Next enabled address is  $\mathbf{c} = (c_1 c_2 \dots c_K)$
    - c) If  $K > M$ , then the next enabled address is obtained by padding string  $\mathbf{c}$  with 0s until getting string of length  $K$ , i.e.  $\mathbf{c}' = (c_1 c_2 \dots c_M 00 \dots 0)$ , where  $l(\mathbf{c}') = K$ .
- 

Another peculiarity of EBT is the rule for case (b) in step 3. Observe the EBT instance in Fig. 6. In slot 9 the address 011001 is resolved, the unresolved terminals observe 3 tokens in interval of length  $p(011001) = 0.40625$ , they estimate  $\hat{n} \doteq 7.4$  and  $K = 2$ . The step 1 of Table I outputs  $\mathbf{c} = 01101$  and having  $K = 2$ , the maximum likelihood choice would be to enable address 01. However, note that once 01101 is visited, it must be that the subtree 010 is already resolved. Therefore,  $K$  is increased while  $c_K = 1$  and the enabled address in slot 10 is 011.

#### D. Practical details for the EBT algorithm

1) *Sequential bit tossing in EBT:* We have presented the EBT algorithm by assuming that each terminal  $d_i$  generates a real random number  $r_i \in [0, 1)$ . In fact, the random bits can be generated sequentially and on-demand, during the algorithm execution. Let terminal  $d_i$  have generated the first  $K$  bits  $\mathbf{b}_i = (b_{i1} b_{i2} \dots b_{iK})$  and let the address of the enabled node in the next slot be  $\mathbf{a} = (a_1 \dots a_L)$ . If the string  $\mathbf{b}_i$  is a prefix of the string  $\mathbf{a}$  and  $L > K$ , then  $d_i$  generates the random bits  $b_{i(K+1)}, b_{i(K+2)}, \dots, b_{iL}$ . To see how this works, note that each terminal  $d_i$  starts with an empty string  $\mathbf{b}_i = \epsilon$ . After the collision, the next enabled node is  $\mathbf{a} = 0$ . Since the empty string is prefix of any string, each terminal tosses a coin to generate the first random bit. Now let the subtree  $\mathbf{a} = 0$  be resolved and let EBT decide that the address to be enabled in the next slot is  $\mathbf{a} = 100$ . Since each unresolved terminals has  $\mathbf{b}_i = 1$ , all unresolved terminals generate two random bits and obtain sequences of type  $\mathbf{b}_i = 1b_{i2}b_{i3}$  and only the terminals with  $\mathbf{b}_i = 100$  transmit in that slot.

2) *Initiation of the conflict resolution:* We have already stated that the interrogator initiates the conflict resolution by sending probe packet with address  $\epsilon$ , which calls for reply from all neighboring terminals. Hence, a lot of

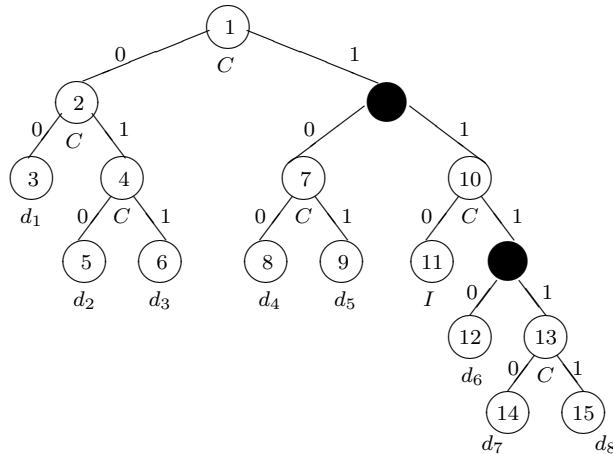


Fig. 4. Instance of the EBT algorithm for  $n = 8$ . The black circle denotes that a node is not enabled during the algorithm execution. The terminals have the same tokens as in the example on Fig. 1

slot no.	1	2	3	4	5	6	7	8	9	10	11	12
<b>a</b>	$\epsilon$	0	00	01	010	0100	01000	010000	010001	010	011	...
channel	C	C	$d_1$	C	C	C	C	$d_2$	$d_3$	$d_4$	$d_5$	...

Fig. 5. An instance of the EBT that repeats the enabling of 010

slot no.	1	2	3	4	5	6	7	8	9	10	11	12
<b>a</b>	$\epsilon$	0	00	01	010	0110	01100	011000	011001	011	100	...
channel	C	C	$d_1$	C	I	C	C	$d_2$	$d_3$	$d_4$	C	...

Fig. 6. An instance of the EBT algorithm to illustrate the rule 3b from Table I

messages are transmitted in the first few slots. If the interrogator somehow knows certain bounds of the number  $n$ , the unnecessary message transmission can be avoided. Let  $N_{\min}$  be the lower bound on  $n$ . Then the EBT algorithm is modified as follows. The first probe packet sent by the interrogator enables the node with address  $\mathbf{a} = 00 \dots 0$ , where the address length is  $l(\mathbf{a}) = \lceil \log_2 N_{\min} \rceil$ . Upon receiving this probe, the terminals generate random tokens or, equivalently, each terminal tosses  $l(\mathbf{a})$  fair coins. As long as there are idle responds to the probes, the interrogator will send probe packets with addresses obtained by traversing the tree as in the very basic BT algorithm. After the first slot with state single or collision, the interrogator continues to run the EBT algorithm as if it has started with  $\epsilon$ -probe. For example, let the initiating probe packet enable the address 0000. If from the response it follows that  $N(0000) = 0$ , the next enabled address is 0001. If again  $N(0001) = 0$ , the next enabled address is 001. If e.g.  $N(0001) = 1$ , then the subtrees 0001 and 000 are resolved and the EBT algorithm proceeds according to the rules in Table I.

3) *EBT for Identity-based Conflict Resolution:* Each responder has a unique device address  $\mathbf{b}_i = (b_{i1}b_{i2} \dots b_{iM})$ .  $M$  is a fixed, predefined number and clearly  $2^M > n$ . There is one problem with the straightforward application of the EBT to the probing. The main strength in EBT is the prediction of highly probable collisions and this

prediction is made by assuming that the terminal tokens are uniformly distributed in  $[0, 1)$ . Equivalently, in the probing scenarios we require that the  $n$  device addresses are uniformly picked from the address space of  $2^L$  integers  $0, 1, 2, \dots, (2^L - 1)$ . However, it may happen that the responders are highly correlated over some part of the device address. For example, the responders can be RFID tags attached to products of certain kind. Then, a part of the device address may be bounded to the product type and thus be identical for each device. In this case, the interrogator can modify the EBT suitably to circumvent the usage of the common part in the device address. As a general method to deal with the problem of non-uniformity, the interrogator may issue a random permutation of length equal to the size of the device address. This permutation can be used to scramble the addresses prior to the start of the EBT algorithm.

#### IV. INTERVAL ESTIMATION COLLISION RESOLUTION (IECR) ALGORITHM

The performance of the EBT algorithm is limited by the fact that the enabled intervals obtain values of type  $2^{-L}$ , where  $L$  is an integer. This limitation cannot be overcome if in the probing scenarios the interrogator has to include, in each probe packet, the binary value of the currently enabled address. Such can be the case of RFID tags. However, when the terminals are capable to calculate the enabled address (i.e. the enabled interval), the performance of EBT can be further improved. In this section we will present the *Interval Estimation Conflict Resolution (IECR)* algorithm. The IECR resolves batch conflicts with efficiency which approximates the efficiency by which the FCFS resolves conflicts of Poisson arrivals.

##### A. FCFS Algorithm for Packets with Poisson Arrivals

Before we proceed to the formulation of the IECR algorithm, it is important to introduce the basic ideas of the FCFS algorithm [14], proposed for Poisson arrivals of packets.

The scenario for operation of FCFS is defined as follows. Let there be a set with infinite number of terminals that contend for the channel, such that each message comes from a different terminal [14] [15]. Let the arrival of packets from this set of terminals be Poisson with rate  $\lambda$ , where  $\lambda$  is **known to the terminals a priori**. The conflict is resolved based on the packet arrival times. When a time epoch of length  $\tau$  is enabled, all packets that arrived within that epoch are transmitted. An epoch is resolved when all packets that arrived within that epoch are resolved. If there is a collision, then the colliding terminals start to run the CBT algorithm and the next enabled epoch is the left half of the current epoch with length  $\frac{\tau}{2}$ . If there is again collision, the CBT continues in the left half, while the right half of the original epoch  $\tau$  is “returned” to the arrival axis. Since it is a key feature of the FCFS algorithm we elaborate why such “return” of the right half is justifiable. Consider a newly enabled epoch of length  $\tau$  and denote the number of packet arrivals in its left and right half by  $k_l$  and  $k_r$ , respectively. The a priori distribution of the packet arrivals in both halves is Poisson, such that the probability of having  $k$  arrivals in each of them is given by:

$$P(k_l = k) = P(k_r = k) = \frac{\left(\frac{\lambda\tau}{2}\right)^k}{k!} e^{-\frac{\lambda\tau}{2}} \quad (17)$$

After having collision in the interval of length  $\tau$ , the a posterior distribution of  $k_l$  and  $k_r$  is not Poisson. However, after the left half is enabled and there is again a collision, the distribution of  $k_l$  is again Poisson as given by (17). Namely, the collision that there is collision in the interval of length  $\tau$  is represented by the condition  $k_r + k_l \geq 2$ , while the collision in the left half by  $k_l \geq 2$ , from which it follows:

$$\begin{aligned} & P_n (k_r = k | k_l \geq 2, k_r + k_l \geq 2) \stackrel{(a)}{=} \\ & = P_n (k_r = k | k_l \geq 2, k_r \geq 0) \stackrel{(b)}{=} \\ & = P (k_r = k | k_r \geq 0) \stackrel{(c)}{=} P(k_r = k) \end{aligned} \quad (18)$$

where  $P(k_r = k)$  is given by (17). In (18), the equality (a) follows from the non-negativity of  $k_l$  and  $k_r$ . The equality (b) follows from the independence among the tokens generated by different terminals. The equality (c) follows again from non-negativity of  $k_r$ . To summarize, after the collision in the left half, the right half becomes statistically identical to the parts of the time axis that has never been enabled.

The manner in which the epochs are enabled ensures that the packets are transmitted in a FCFS manner. After the current CBT is terminated, a new epoch of length  $\tau$  should be enabled. An optimized choice  $\tau_{opt}$  is numerically obtained to be [15]:

$$\tau = \frac{1.26}{\lambda} \quad (19)$$

such that there are on average 1.26 arrivals in a freshly enabled interval. The analysis used in obtaining the value 1.26 is given in the Appendix II. The maximal achievable throughput for this algorithm is  $\lambda = 0.487$  and hence it is often referred to as 0.487 FCFS algorithm.

### B. Formulation of the IECR Algorithm

The ideas from the 0.487 FCFS can be translated into a conflict resolution algorithm for batch arrival. Let the interrogator send  $\epsilon$ -probe to solicit replies from the  $n$  terminals. The terminal  $d_i$  generates token  $r_i$  from the uniform distribution over the interval  $[0, 1)$ . The conflict resolution proceeds by enabling subintervals of  $[0, 1)$ .

Let us first assume that there  $n$  is known and it is known that there are  $k$  tokens in some interval  $[0, x)$ . Then it is also known that there are  $n - k$  tokens uniformly distributed within the interval  $[x, 1)$ . Let the next enabled interval be  $s = [x, x + x_0)$  and let there be collision. Subsequently the interval  $s_l = [x, x + \frac{x_0}{2})$  is enabled and let there also be collision; then the interval  $s_r = [x + \frac{x_0}{2}, x + x_0)$  is returned to the interval  $[x, 1)$ . Similarly to the FCFS case, the information about the distribution of tokens  $s_r$  which remains after perceiving the collision in  $s_l$  is that the number of tokens in  $s_r$  is non-negative, which is actually a priori information. Note that in FCFS the terminals know the value  $\lambda$  of the arrival rate and therefore they can choose the optimized length of the new enabled interval. The analogy in case of batch conflict is that, with known number  $n$  of tokens in some interval, the optimized length of the enabled subinterval can be chosen.

When  $n$  is not known, then it can be estimated as the batch conflict resolution proceeds, as explained in Section III-B. In fact, the conclusions stated by (18) are implicitly used in the EBT algorithm. Recall that, after finishing the

```

 $p_{low} = 0; p_{up} = 1; p_{lim} = 1;$ 
 $end = no; k = 0; Tx = no; state = right;$ 
generate  $r_i$ ;
while( $end \neq no$ )
  if( $r_i \in [p_{low}, p_{up}]$ );
    transmit; set  $Tx = yes$ ;
  else
    set  $Tx = no$ ;
  get feedback at the end of the slot;
  if(collision)
    state = left;
     $p_{up} = \frac{p_{low} + p_{lim}}{2}$ ;
  else
    if(single)
       $k++$ ;
      if ( $Tx = yes$ ) set  $end = yes$ ; exit;
    if( $state = left$ )
       $p_{low} = p_{up}$ ;
      if(idle)
         $p_{up} = \frac{p_{low} + p_{lim}}{2}$ ;
      else
         $p_{up} = p_{lim}$ ;
        state = right;
    else
       $\Delta = \frac{1.26 p_{lim}}{k}$ 
       $p_{low} = p_{lim}$ ;
       $p_{lim} = \min(1, p_{low} + \Delta)$ ;
       $p_{up} = p_{lim}$ ;
      state = right;

```

Fig. 7. The interval estimation conflict resolution (IECR) algorithm as run by the  $i$ -th node.

current execution of the CBT, the EBT algorithm does not make use of the information about previous collisions. It rather uses the total number of resolved tokens, while it forgets the collisions that possibly involved the tokens that are still not resolved. Assume that in EBT the interval  $[0, p)$  is already resolved and there are  $k > 0$  tokens in  $[0, p)$ . The next enabled interval in EBT is  $[p, p + x)$ , where  $x = 2^{-\lceil \log_2 \hat{n} \rceil}$  and  $\hat{n}$  is given by (7). When  $n$  becomes large, the distribution of the number of tokens  $k$  in  $[p, p + x)$  becomes approximately Poisson:

$$P(k) \doteq \frac{\Delta^k}{k!} e^{-\Delta} \quad (20)$$

where  $\Delta = nx$ . Then, the enabling of the interval  $[p, p + x)$  is equivalent to enabling epoch in the FCFS algorithm of length  $\tau$ , where:

$$\tau = \frac{\Delta}{\lambda} \quad (21)$$

From (19) it follows that for large  $n$  it should be  $\Delta = 1.26$ , while in the EBT algorithm we have  $\Delta = n2^{-\lceil \log_2 \hat{n} \rceil}$ . This observation gives a hint how to improve the EBT algorithm: The terminals that are still unresolved should calculate the length of the next enabled subinterval to be

$$x = \frac{1.26}{\hat{n}} = \frac{1.26p}{k} \quad (22)$$

where  $k$  is the number of resolved tokens within  $[0, p)$ . Fig. 7 shows the pseudocode for the IECR algorithm. It is seen that, after a terminated CBT, the length of the next enabled interval is chosen according to (22)<sup>1</sup>.

<sup>1</sup>Clearly, this also hints that in EBT algorithm the next enabled level should be chosen as  $L_{\max} = \lceil \log_2 \frac{n}{1.26} \rceil$  instead of according to (14).

The choice (22) is suboptimal. In general, the value of  $\Delta$  is not constant, but  $\Delta = f(k, p)$ , a function of the resolved tokens  $k$  and the interval  $p$ . Also, as it has been shown in [12], the binary splitting of the intervals upon collision is also suboptimal. In order to find the optimal value for  $\Delta$  and the splitting, one can observe IECR as a Markov Decision Process [18] and attempt to optimize it further. Nonetheless, such complete optimization is out of the scope of this paper. Instead, in Appendix III we show how the newly enabled intervals should be chosen when the multiplicity  $n$  is known a priori.

The following theorem is related to the asymptotic efficiency ( $n \rightarrow \infty$ ) of the IECR algorithm:

*Theorem 1:* The time efficiency of the IECR algorithm satisfies

$$\lim_{n \rightarrow \infty} \tau_n = \lim_{n \rightarrow \infty} \frac{n}{T_n} = 0.487$$

*Proof:* Let us assume that  $k$  tokens have been resolved within the interval  $[0, p)$ . Let the new CBT start by enabling new interval  $[p, p + x)$  and in addition it holds that  $p + x < 1$ . If the newly enabled interval initially contains  $m$  tokens then the average duration of the CBT is denoted by  $B_m$ , while the average number of tokens resolved upon the termination of the CBT is denoted by  $U_m$ . The relations to obtain  $B_m$  and  $U_m$  for any  $m > 1$  are given in the Appendix II. By using the bounding techniques presented in [10] it can be shown that  $B_m$  grows sub-linearly  $B_m = o(m)$ , while  $U_m$  is upperbounded by a constant ( $\approx 2.6$ ) for any  $m$ .

Let us first assume that  $n$  is known such that it is known to the terminals that there are  $N = n - k$  unresolved tokens in  $[p, 1)$ . In such case the length of the newly enabled interval is  $x = \frac{1.26(1-p)}{N}$ . The efficiency by which the CBT is executed at with this interval can be calculated as:

$$\lambda_N(x) = \frac{\sum_{m=0}^N U_m P_m(\frac{x}{1-p} | N)}{\sum_{m=0}^N B_m P_m(\frac{x}{1-p} | N)} \quad (23)$$

where

$$P_m(\frac{x}{1-p} | N) = \binom{N}{m} \left(\frac{1.26}{N}\right)^m \left(1 - \frac{1.26}{N}\right)^{N-m} \quad (24)$$

Due to the Poisson approximation of the binomial distribution we have that:

$$\lim_{N \rightarrow \infty} \lambda_N(\frac{x}{1-p}) = \lambda = \frac{\sum_{m=0}^{\infty} U_m \frac{1.26^m}{m!} e^{-1.26}}{\sum_{m=0}^{\infty} B_m \frac{1.26^m}{m!} e^{-1.26}} = 0.487 \quad (25)$$

Now let us assume that  $n$  is unknown, such that the value of  $\hat{n}$  is used and  $x$  is chosen according to the Eq. (22). Using the Tschebysheff inequality (11) and taking into account the sub-linear increase of  $B_m$  as  $m$  increases, it can be shown that the average duration  $\bar{B}$  of the CBT obtained for unknown  $n$  becomes arbitrarily close to the average duration  $\bar{B}$  when  $N = n - k$  is perfectly known. The same applies for the average number of resolved tokens  $\bar{U}$ . This is because, for fixed  $p$  the estimate of  $n$  becomes progressively accurate as  $n \rightarrow \infty$ . These facts can be used to show that the efficiency of a single CBT run approaches  $\lambda = 0.487$ , given by (25).

From the previous discussion we can conclude that the IECR algorithm moves across the interval  $[p_1, p_2)$ , where  $p_1 > 0, p_2 < 1$ , with the ‘‘speed’’ that approximates the speed by which the FCFS algorithm moves across the time axis. If the total number of tokens in  $[p_1, p_2)$  be  $n_1 < n$  then the time consumed to resolve these tokens is  $T_{n_1} = \frac{n_1}{0.487} = 2.0534 \cdot n_1$ .

To complete the proof, we need to consider the boundary cases i.e. what happens in the intervals  $[0, p_1)$  and  $[p_2, 1)$ . The discussion above is valid when  $p + x < 1$ . Since  $x \rightarrow 0$  as  $n \rightarrow \infty$ , the value  $p_2$  approaches 1 and the average number of tokens in  $[p_2, 1)$  remains upperbounded by a constant  $c_2$ .

Finally, recall the IECR includes an initial CBT algorithm before it starts to operate with the estimate  $\hat{n}$ . The average number of resolved tokens within the initial CBT algorithm  $U_n$  is upperbounded by  $c_1$ . Hence, the initial CBT algorithm resolves the interval  $[0, q_n)$ , where  $q_n \doteq \frac{c_1}{n} \rightarrow 0$  and thus we can conclude that  $p_1$  can be arbitrary close to 0. Recall that the average duration of the initial CBT algorithm  $B_n$  grows as  $o(n)$ . Consequently, the efficiency of the IECR algorithm is:

$$\eta_n = \frac{n}{T_n} = \frac{c_1 + n_1 + c_2}{o(n) + 2.0534n_1 + O(1)} \rightarrow \frac{n_1}{T_{n_1}} = 0.487 \quad (26)$$

over the complete interval  $[0, 1)$ . ■

Similarly to the EBT algorithm, if a lower bound  $N_{\min}$  on the number of terminals is known, the IECR algorithm can save some messages during the conflict resolution. The interrogator initially enables the interval  $\left[0, \frac{1}{N_{\min}}\right)$ . Refer to Fig. 7. To account for this case, the algorithm should be changed at the line in which  $\Delta = \frac{1.26p_{lim}}{k}$  is set. Instead,  $\Delta$  should be initialized at  $\Delta = \frac{1}{N_{\min}}$  and while  $k = 0$ ,  $\Delta$  is set  $\Delta = 2 \cdot \Delta$ . Otherwise, the algorithm stays the same.

To run the IECR algorithm, the terminals can again toss fair coin sequentially. Let us assume that the  $i$ -th terminal  $d_i$  has tossed  $L$  bits so far  $\mathbf{b}_i = (b_{i1}b_{i2}b_{i3} \dots b_{iL})$ . The string  $\mathbf{b}_i$  can be interpreted that the token  $r_i$  belongs to the interval  $[r(\mathbf{b}_i), p(\mathbf{b}_i))$ , where  $r(\mathbf{b}_i)$  and  $p(\mathbf{b}_i)$  are determined from (2) and (3). Having only  $L$  bits,  $r_i$  is still not fully determined. Let IECR enable the interval  $[x_1, x_2)$ . The terminal  $d_i$  starts to toss coins if and only if the interval  $[r(\mathbf{b}_i), p(\mathbf{b}_i))$  intersects with the interval  $[x_1, x_2)$  and  $[r(\mathbf{b}_i), p(\mathbf{b}_i))$  does not lie fully within  $[x_1, x_2)$ . The terminal tosses coins and until it obtains  $\mathbf{b}'_i$  which satisfies either  $[r(\mathbf{b}'_i), p(\mathbf{b}'_i)) \cap [x_1, x_2) = \emptyset$  (the terminal stays silent) or  $x_1 < r(\mathbf{b}'_i), p(\mathbf{b}'_i) < x_2$  (the terminal transmits).

### C. IECR Algorithm for CSMA Channel

The design of the IECR algorithm can be extended to CSMA channel model. Since the idle slot  $T_S \ll T_P$ , the idle response becomes “cheap” in terms of time, besides being “cheap” in messages. The CSMA design for IECR follows the same guidelines as the CSMA design for FCFS, discussed in [14]. An idle slot occurs at the end of each success or collision to provide time for feedback. Thus, a collision or a success has duration of  $(1 + \beta)T_P$ , while idle response has duration  $\beta T_P$ . Since the collisions waste much more slots than the idle slots the enabled interval in (22), used after the current CBT is terminated, should be shortened. Following the derivations in Section 4.4.4 of [14], the next enabled interval after  $[0, p)$  is resolved and  $C(0, p) = k$  should be  $[p, p + x)$  where:

$$x = \frac{p}{k} \sqrt{\frac{2\beta}{1 + \sqrt{\beta}}}; \quad (27)$$

On the other hand, let  $[p, p + x)$  be enabled and let there be collision. The next enabled interval should be chosen to be  $[p, p + y)$  where:

$$y = \sqrt{\beta + \beta^2} - \beta \quad (28)$$

The EBT algorithm can also be engineered to operate in a CSMA channels. However, in such case the lengths of the enabled intervals would only approximate the values prescribed by Equations (27) and (28), since it can use the intervals of length  $2^{-l}$ .

1) *The Combined Sift / IECR Protocol:* In order to evaluate the IECR approach for CSMA channel we first introduce the Sift protocol from [16]. The Sift protocol considers the scenario with  $n$  terminals that simultaneously attempt to transmit, while the interrogator needs the replies only the first  $k$  resolved terminals. Sift uses a small and fixed contention window of size  $w = 32$  slots. The terminals that are running Sift compete for any slot  $j \in [1, w]$  based on the shared belief of the current value of  $n$ . The terminal chooses to start transmission in slot  $j$  with probability:

$$p_j = \frac{(1 - \alpha)\alpha^w}{1 - \alpha^w} \cdot \alpha^{-j} \quad (29)$$

for  $j \in [1, w]$  and  $0 < \alpha < 1$  is the parameter of the distribution. In this range of  $\alpha$ ,  $p_j$  increases exponentially with  $j$ . Clearly  $\sum_{j=1}^w p_j = 1$ . To determine  $\alpha$ , an upper bound  $\bar{N}_{\max}$  on the number of nodes should be known. If  $n > \bar{N}_{\max}$ , the performance of the Sift protocols degrades gracefully.

The Sift protocol can also be represented within the framework of “enabled intervals”, used in describing the IECR approach. Before the start of the Sift, each of the  $n$  terminals generates a token uniformly within  $[0, 1)$ . In the slot  $m$  the interval  $[x_m, y_m)$  is enabled, where:

$$\begin{aligned} x_m &= \sum_{j=1}^{m-1} p_j \\ y_m &= \sum_{j=1}^m p_j \end{aligned} \quad (30)$$

In the first slot, the enabled interval is  $[0, p_1)$ . If a collision occurs, all terminals generate new tokens. The Sift protocol has been optimized for minimizing the collisions and time until the first resolved interval is obtained. If  $k > 1$  terminals need to be resolved, then the Sift protocol is started over by the nodes that have not been resolved, but in such restart no adaptation is done.

A key assumption adopted in the Sift protocol is that there is a CSMA channel with  $T_S \ll T_P$  i.e.  $\beta \ll 1$  and the collisions are minimized by taking advantage of the small  $\beta$ . Sift is optimized to obtain the first reply (partial resolution 1 out of  $n$ ), but it does not adapt during the operation when  $k > 1$  out of  $n$  possible replies are needed. We can combine the Sift protocol with the CSMA version of the IECR algorithm to produce an efficient conflict resolution algorithm for the CSMA channel as follows:

- Until the first non-idle slot occurs, the Sift algorithm is being run.
- Assume that the first non-idle slot is successful transmission started in slot  $m$ , where  $m = [1, w]$ . Then the IECR algorithm starts having  $[0, p)$  resolved and  $C(0, p) = 1$ , where  $p = \sum_{j=1}^m p_j$ .

TABLE II  
TIME EFFICIENCY  $\eta_n$  OF VARIOUS CONFLICT RESOLUTION ALGORITHMS

$n$	MBT	Cidon & Sidi $p = 0.1, \beta = 8$	EBT	EBT with $N_{\min} = 64$	IECR	IECR with $N_{\min} = 50$	IECR with $N_{\min} = 500$	Optimized IECR with a priori known $n$
10	0.390	0.412	0.425	0.417	0.432	0.433	0.385	0.535
50	0.379	0.452	0.452	0.472	0.463	0.475	0.467	0.499
100	0.376	0.455	0.460	0.471	0.467	0.480	0.479	0.494
500	0.375	0.465	0.460	0.464	0.481	0.484	0.486	0.489
1000	0.375	0.469	0.463	0.465	0.483	0.485	0.486	0.488

TABLE III  
AVERAGE NUMBER OF MESSAGES PER TERMINAL  $\mu_n$  FOR VARIOUS CONFLICT RESOLUTION ALGORITHMS

$n$	MBT	Cidon & Sidi $p = 0.1, \beta = 8$	EBT	EBT with $N_{\min} = 64$	IECR	IECR with $N_{\min} = 50$	IECR with $N_{\min} = 500$	Optimized IECR with a priori known $n$
10	5.100	4.497	4.191	2.474	4.146	2.554	2.504	2.123
50	7.458	3.286	4.554	2.622	4.448	2.565	2.548	2.370
100	8.475	3.177	4.617	2.655	4.494	2.545	2.509	2.410
500	10.798	3.128	4.740	2.742	4.482	2.513	2.476	2.456
1000	11.800	3.156	4.770	2.806	4.470	2.513	2.473	2.461

- If the first non-idle slot is collision, the IECR algorithm starts to resolve the collision

We expect this combined algorithm (shortly called Sift/IECR) to be more time efficient than Sift when  $k > 1$  terminals should be resolved.

## V. RESULTS

In this section we will show two group of results. The first group of results evaluates the performance of EBT and IECR algorithms for slotted channel. The second group of results shows an example how the IECR approach can be successfully applied to the problem of partial resolution in CSMA channel.

### A. Batch Conflict Resolution in a Slotted Channel

In this section we show that both EBT and IECR have high efficiency, thus providing fast conflict resolution, while the number of messages spent in the process of conflict resolution remains low and grows only linearly with the multiplicity  $n$ . We compare the performance of EBT and IECR with the MBT algorithm, as well as with the batch resolution algorithm of Cidon and Sidi [4]. The algorithm from [4] that we have simulated for comparison purposes, has been so far known as the fastest method for resolving batch conflict with unknown multiplicity. Finally, we also provide the results for the efficiency of the optimized algorithm with a priori known multiplicity, derived in Appendix III.

Table II shows the speed of various conflict resolution algorithms, expressed through the efficiency  $\eta_n$ . Table III shows the average number  $\mu_n$  of messages transmitted by a terminal during the complete conflict resolution. To make fair comparison, we have simulated the algorithm of Cidon and Sidi from [4] with re-execution of the partial resolution, which is suggested as the most efficient method. Clearly, the IECR algorithm is the fastest method for resolving conflicts of finite multiplicity  $n$  and its efficiency approaches the 0.487 FCFS algorithm. For conflicts of relatively small multiplicity  $n = 10, 50, 100$ , the EBT algorithm also outperforms the algorithm of Cidon and Sidi. The authors in [4] recommend the usage of the values  $p = 0.1$  and  $\beta = 8$  heuristically. Here we can see that the introduction of  $N_{\min}$  can significantly decrease the average number of messages per terminal. The overestimation of  $N_{\min}$  may decrease the time efficiency (see  $\eta_{10}$  for  $N_{\min} = 500$ ), but it is not always the case (see  $\eta_{50}$  for  $N_{\min} = 500$ ). On the other hand, the overestimation of  $N_{\min} = 500$  can never increase the average number of messages per terminal. This is understandable, since for every  $N_{\min} > 1$  the first collision when all  $n$  terminals transmit is avoided. If the reduction of the number of transmitted messages is of high importance, small decrease in the efficiency may be acceptable. From tables II and III it can be inferred that, for the range of multiplicity  $n \leq 1000$ , picking  $N_{\min} = 50$  is satisfactory.

Regarding the performance of the optimized IECR algorithm when  $n$  is known in advance, it can be concluded that the efficiency  $\eta_n$  approaches the value 0.487, but contrary to the IERC with unknown multiplicity, in this case  $\eta_n$  monotonically decreases with  $n$  and approaches 0.487 from above. The a priori knowledge of  $n$  helps to speed up the conflict resolution only when  $n$  is small, while as  $n \rightarrow \infty$  the exact value of  $n$  does not contribute to the overall efficiency of the batch conflict resolution.

The Table IV shows the average number of slots and messages that are spent until the interval  $[0, p)$  is resolved. For these results, IECR is taken with  $N_{\min} = 50$  and multiplicity  $n = 1000$ . The value  $k = np$  denotes the average number of resolved tokens (terminals) within the interval of length  $p$ . For example, if the interrogator starts an inquiry by which it needs to get only 10 reply packets from a set of 1000 terminals, IECR provides these replies in 26 slots with total number of messages spent 64.5. From (10), the value  $p$  determines the accuracy by which the estimation of  $n$  is made. On the other hand, the multiplicity estimation in [9] and [4] attains only predefined accuracy, which cannot be changed during the algorithm execution. With appropriate value of  $N_{\min}$ , in IECR and EBT the amount of resources (slots and messages) spent grows roughly linearly with  $p$ .

Although EBT appears to have a more modest performance compared to IECR, it must be noted that EBT is still applicable when the probing scenario needs the state of the probing algorithm to be explicitly put into the probe packet. This is the case, for example, with the RFID tags [3]. Furthermore, the explicit state in the probe packet can help an error-resilient design of the probing procedure. Finally, the explicit state can help the terminals to employ some energy-efficient sleeping strategies while participating in the probing process.

### B. Partial Resolution in CSMA Channel

The results in Table V compare the average duration of the partial resolution of the combined Sift/IECR algorithm, described earlier, and the pure Sift protocol. The parameter in Eq. (29) is taken to be  $\alpha = 0.82$ , which is obtained

TABLE IV

AVERAGE NUMBER OF SLOTS AND MESSAGES UNTIL THE INTERVAL  $[0, p)$  IS RESOLVED FOR IECR WITH  $N_{\min} = 50$  AND  $n = 1000$ . THE VALUE  $k$  DENOTES THE AVERAGE NUMBER OF RESOLVED TERMINALS IN  $[0, p)$

$p$	$k$	slots	messages
0.01	10	26.07	64.53
0.04	40	86.60	137.54
0.1	100	209.39	284.42
0.4	400	826.46	1027.56

TABLE V

SIFT/IECR VS. SIFT. AVERAGE DURATION AND AVERAGE NUMBER OF MESSAGES PER TERMINAL UNTIL SUFFICIENT NUMBER OF  $k$  TERMINALS ARE RESOLVED. THE DURATION IS NORMALIZED WITH RESPECT TO THE PACKET DURATION  $T_P$  AND  $\beta = 1/16$ .

$n$	$k$	Time (in $T_P$ )		Messages	
		Sift/IECR	Sift	Sift/IECR	Sift
100	1	1.651	1.704	1.320	1.348
100	10	13.864	16.717	14.134	12.768
100	50	66.796	86.319	67.516	63.152
1000	1	1.539	1.473	1.942	1.742
1000	10	13.714	14.656	14.806	17.550
1000	50	66.943	73.047	68.964	86.742

by assuming  $N_{\max} = 512$  [16]. We show the results for  $n = 100$  and  $n = 1000$  so that we have both  $n < N_{\max}$  and  $n > N_{\max}$  cases. The value  $\beta = 1/16$  is used and such choice is motivated from the values in IEEE 802.11 [19], also used in [16], where  $T_S = 20$  [ $\mu\text{sec}$ ] and  $T_P = 320$  [ $\mu\text{sec}$ ] for the RTS packet used in the contention at 1 Mbps.

The simulation results show that Sift/IECR is faster than the Sift algorithm for all required values of  $k$ . The fact that Sift/IECR is faster than Sift even for  $k = 1$  can be explained by noting that if Sift encounters collision, it starts over the contention, while in the Sift/IECR the efficient collision resolution takes part. The larger gain in time efficiency for Sift/IECR over Sift for  $n = 100$  and  $k = 50$  can be explained as follows. As the number of resolved terminal increases, the number of contending terminals decreases, such that the transmission starts later in the contention window and the collision becomes more probable. Besides that, when  $n = 100$  and  $k = 50$  the Sift protocol still yields less collisions and thus less average number of messages per terminal. Note that when  $n = 1000 > N_{\max}$  the Sift/IECR outperforms Sift both in time and message complexity. In particular, the gain is more visible in the average number of messages transmitted per node. The increased number of collisions in IECR is due to the underestimation of  $N_{\max}$ . On the other hand, IECR constantly adapts to the current size of the population of contending terminals.

## VI. CONCLUSION

The communication tasks in emerging wireless networks involve inquiries over a shared wireless channel. In general, such inquiries can be represented by the scenario where a device called interrogator solicits replies from the terminals within its range. This scenario gives rise to batch conflicts, where an unknown number  $n$  of terminals attempt to simultaneously transmit message to the same receiver. In this paper we have presented a novel class of batch conflict resolution algorithms in which the conflict multiplicity is estimated in “real-time” and becomes progressively accurate as more terminals are resolved. Therefore, besides the traditional applications that require resolution of all  $n$  terminals, this new class of algorithms efficiently supports applications where partial resolution is required. We have shown that the standard binary tree algorithms offer inherent possibility for obtaining an estimate  $\hat{n}$  of the number of terminals. Since  $\hat{n}$  becomes progressively accurate, the accuracy of  $\hat{n}$  can be traded off with the amount of time/messages utilized in the conflict resolution. Next, we use the estimation to propose a more efficient binary tree algorithm, termed Estimating Binary Tree (EBT) algorithm. Such algorithm can be used in probing scenarios, where the terminals send reply only after receiving appropriate probe packet from the interrogator. We have generalized the principle behind the EBT algorithm and designed the Interval Estimation Conflict Resolution (IECR) algorithm. The IECR has been designed for both slotted and CSMA channels. The IECR for slotted channel is shown to be the most efficient batch resolution algorithm among the algorithms reported so far in the literature. We have proved that for  $n \rightarrow \infty$  the efficiency achieved by IECR for batch arrivals is identical with the efficiency that Gallager’s FCFS algorithm achieves for Poisson packet arrivals. Regarding the CSMA channel, we have shown that IECR can be combined with the recently proposed Sift algorithm. The Sift algorithm minimizes the number of collisions until the first terminal is resolved, while IECR continues the conflict resolution until the sufficient number  $k$  of replies is obtained. The main gain of using IECR is that it continuously adapts to the current size of the population of contending terminals. Finally, for known bounds on the number of terminals, the IECR and the EBT algorithms can be suitably engineered to minimize the amount of messages transmitted in the conflict resolution. This condition is very important, considering the fact that the terminals are usually battery-powered and each transmitted bit is valuable.

The novelty of the approach presented here leaves many challenges for future work, some of which have been hinted throughout the paper:

- Achieving optimization of the IECR algorithm through the approach of Markov Decision Processes
- The presented algorithms are assuming the only source of error in the wireless channel is the collision. It would be interesting to enhance the algorithms with error-resilient mechanisms.
- Finally, the approach in the IECR motivates research on a more fundamental result, stated in the following conjecture: The achievable performance of the “best” batch conflict resolution algorithm for multiplicity  $n \rightarrow \infty$  can not be improved by knowing  $n$  in advance. The intuitive reason for this conjecture is that, when  $n \rightarrow \infty$ , the estimate of  $n$  can be made arbitrary close to  $n$ , such that the prior knowledge of  $n$  cannot improve the conflict resolution.

## APPENDIX I

## ALTERNATIVE HEURISTICS FOR USING THE MULTIPLICITY ESTIMATION

In order to define an alternative heuristics, we seek to answer the following question: For a given conflict multiplicity  $n$ , at which level  $L$  there is a highest probability that the BT algorithm resolves a terminal? Refer to Fig. 1 when  $d_2$  is resolved, i.e.  $N(010) = 1$ . Recall that the basic BT algorithm traverses the tree in a strictly pre-order manner. Then it must be that the parent node 01 has  $N(01) > 1$ , otherwise  $d_2$  would have been resolved when 01 has been enabled. More generally, let  $\mathbf{a}^p$  be the parent of two nodes with addresses  $\mathbf{a}^0$  and  $\mathbf{a}^1$ , respectively. The nodes  $\mathbf{a}^0$  and  $\mathbf{a}^1$  are called siblings. If during its execution, the BT algorithm enables the node  $\mathbf{a}^0$  and  $N(\mathbf{a}^0) = 1$ , then  $N(\mathbf{a}^p) > 1$ . Equivalently, if the BT algorithm resolves the address  $\mathbf{a}^0$ , then it can be inferred that  $N(\mathbf{a}^1) > 0$ . Let  $x = 2^{-L}$  be the length of the interval that corresponds to enabling of the node  $\mathbf{a}^0$ . Let us for the moment neglect the fact that  $L$  must be integer, such that  $x$  takes values from a countable set. Since the  $n$  tokens are uniformly distributed in  $[0, 1)$ , the probability that  $i$  tokens fall in a particular interval of length  $x$  is:

$$P_i(x|n) = \binom{n}{i} x^i (1-x)^{n-i} \quad (31)$$

Clearly, the length of interval that corresponds to enabling the node  $\mathbf{a}^1$  has the same length  $x$ . We are interested in the following probability:

$$\begin{aligned} & P(N(\mathbf{a}^0) = 1, N(\mathbf{a}^1) > 0|n) = \\ &= P(N(\mathbf{a}^0) = 1|n) - P(N(\mathbf{a}^0) = 1, N(\mathbf{a}^1) = 0|n) = \\ &= nx(1-x)^{n-1} - nx(1-2x)^{n-1} \\ &= f(x) \end{aligned} \quad (32)$$

To maximize this probability we set  $\frac{df}{dx} = 0$  and, setting

$$\alpha(n) = nx$$

we obtain the following equation:

$$1 - \alpha + (2\alpha - 1) \left( \frac{1 - \frac{2\alpha}{n}}{1 - \frac{\alpha}{n}} \right)^{n-2} = 0 \quad (33)$$

where shortly  $\alpha(n) = \alpha$ . From this equation it is not straightforward to obtain solution for  $\alpha$  as function of  $n$ . For the case  $n = 3$ , the value is  $\alpha = 1.333$ . If  $n \rightarrow \infty$ , the Eq. (33) becomes:

$$1 - \alpha + (2\alpha - 1) \cdot e^{-\alpha} = 0 \quad (34)$$

which has solution  $\alpha(\infty) = 1.446\dots$

Although it can not be written in the closed form, through (tedious) further analysis it can be shown that the solution  $\alpha$  for (33) is an increasing function of  $n$  which very quickly reaches a value close to  $\alpha(\infty)$ .

Hence, with this heuristics, instead of Eq. (14), the level of the enabled node should be chosen as:

$$L_{\max} = \left\lceil \log_2 \frac{n}{\alpha(n)} \right\rceil \doteq \left\lceil \log_2 \frac{n}{1.45} \right\rceil \quad (35)$$

$n$	1	2	3	4	5	10	50	100	1000
$B_n$	1.000	4.000	5.833	6.476	6.670	7.541	9.803	10.796	14.111
$U_n$	1.000	2.000	2.500	2.571	2.533	2.508	2.506	2.506	2.506

TABLE VI

THE VALUES OF  $B_n$  AND  $U_n$  FOR DIFFERENT  $n$ 

## APPENDIX II

## ANALYSIS OF THE EFFICIENCY OF THE FCFS

A basic constituent of the IECR algorithm is the CBT algorithm. Here we outline the analysis of the efficiency of the CBT which is directly related, as given in [15].

Let  $B_n$  denote the average duration of CBT when initial conflict contains  $n$  packet arrivals within the newly enabled interval of length  $\tau$ . For each  $n \geq 2$  the value of  $B_n$  can be determined by the following recurrent relation:

$$B_n = \frac{1 + Q_1(n)(1 + B_{n-1}) + \sum_{i=2}^{n-1} Q_i(n)B_i}{1 - Q_0(n) - Q_n(n)} \quad (36)$$

while  $B_0 = B_1 = 1$ . The variable  $Q_i(n)$  is a probability from the binomial distribution:

$$Q_i(n) = \binom{n}{i} \frac{1}{2^n} \quad (37)$$

If the initial conflict is of size  $n > 2$ , then CBT does not necessarily resolve all  $n$  packets. Let us denote by  $U_n$  the average number of resolved packets upon the termination of the CBT that started with initial conflict of multiplicity  $n$ . For  $n = 0, 1, 2$  it holds that  $U_n = n$ , while for  $n \geq 3$  the following recurrent relation can be used:

$$U_n = \frac{Q_1(n)(1 + U_{n-1}) + \sum_{i=2}^{n-1} Q_i(n)U_i}{1 - Q_0(n) - Q_n(n)} \quad (38)$$

The average number of arrivals in a newly enabled epoch of length  $\tau$  (expressed in slots) is  $z = \tau\lambda$  where  $\lambda$  is the rate of Poisson packet arrivals. Then average duration of a CBT is:

$$\bar{B} = \sum_{n=0}^{\infty} B_n \frac{z^n}{n!} e^{-z} \quad (39)$$

while the average number of messages resolved by the CBT is given by:

$$\bar{U} = \sum_{n=0}^{\infty} U_n \frac{z^n}{n!} e^{-z} \quad (40)$$

The efficiency by which the FCFS resolves the packet arrivals should be greater or equal to the arrival rate  $\lambda$ :

$$\lambda \leq \frac{\bar{U}}{\bar{B}} = \frac{\sum_{n=0}^{\infty} U_n \frac{z^n}{n!} e^{-z}}{\sum_{n=0}^{\infty} B_n \frac{z^n}{n!} e^{-z}} \quad (41)$$

The efficiency is maximized when  $z = 1.266$  i.e.  $\tau = \frac{1.266}{\lambda}$ , which yields to  $\lambda = 0.487$ .

## APPENDIX III

## OPTIMIZATION OF THE BATCH CONFLICT RESOLUTION WITH A PRIORI KNOWN MULTIPLICITY

Let  $n$  tokens be uniformly distributed in the interval  $[0, 1)$  and let  $n$  be known a priori. In this case an algorithm analogous to IECR can be applied, consisting of a sequence of executions of the CBT algorithm. Clearly, instead of using estimation, with known  $n$  it is always exactly known how many tokens are left in the unresolved part of the interval.

Before going to the optimization of the average duration for a batch conflict resolution of given  $n$ , we need to derive the following probability. Let us consider a CBT that starts with conflict multiplicity  $m$  i.e. the number of tokens in the enabled interval is  $m$ . Then let  $R(k|m)$  denote the conditional probability that  $k$  tokens out of  $m$  are not resolved upon the termination of the CBT execution. After the initial conflict of multiplicity  $m$  the enabled interval is halved and let the probability  $Q_i(m)$ , given by (37), be interpreted as a probability that  $i$  tokens are in the right half of the enabled interval. Then the following recurrent relation holds:

$$R(k|m) = Q_m(m) \cdot R(k|m) + Q_{m-1}(m) \cdot R(k|m-1) + \sum_{i=0}^k Q_i(m) \cdot R(k-i|m-i) \quad (42)$$

and

$$R(k|m) = \frac{Q_{m-1}(m) \cdot R(k|m-1) + \sum_{i=1}^k Q_i(m) \cdot R(k-i|m-i)}{1 - Q_m(m) - Q_0(m)} \quad (43)$$

which holds for  $m > 2$  and  $0 \leq k \leq m-2$  (since when  $m > 2$  at least 2 tokens should be resolved in the CBT). The initial conditions are  $R(0|1) = R(0|2) = 1$ .

The initial enabled interval is  $[0, x)$ . Let  $T_n$  denote the minimized averaged duration of the batch conflict resolution when it is a priori known that the conflict multiplicity is  $n$ . First note that  $T_0 = 0$  (it is optimal not to start the conflict resolution if there are no tokens) and  $T_1 = 1$ . Let us assume that  $T_{n_1}$  is known for each  $0 \leq n_1 < n$ . With a slight abuse of the notation, we can write that  $T_n(x)$  is the average duration of the batch conflict resolution algorithm when the length of the initially enabled interval is  $x$ . The goal is to find  $x_n$  that minimizes  $T_n(x)$ , given  $T_{n_1}$  for  $0 \leq n_1 < n$ , and thus obtain  $T_n = T_n(x_n)$ . The functional relation that determines  $T_n(x)$  is:

$$T_n(x) = P_0(x|n)(B_0 + T_n(x)) + P_1(x|n)(B_1 + T_{n-1}) + P_2(x|n)(B_2 + T_{n-2}) + \sum_{m=3}^n P_m(x|n) \left[ \sum_{k=0}^{m-2} R(k|m) \cdot (b_{k|m} + T_{n-m+k}) \right] \quad (44)$$

where

- $B_m$  is the average duration of the CBT, as defined in Section II
- $P_m(x|n) = \binom{n}{m} x^m (1-x)^{n-m}$
- $b_{k,m}$  is average duration of the CBT that starts with  $m$  tokens when  $k$  out of them are left unresolved upon the termination of the CBT execution. The next equality can be easily checked:

$$\sum_{k=0}^{m-2} R(k|m) \cdot b_{k|m} = B_m \quad (45)$$

$n$	2	5	10	20	50	100	1000	3000
$z_n$	1.000	1.204	1.236	1.251	1.261	1.262	1.266	1.266
$T_n$	3.000	8.733	18.683	38.884	100.029	202.336	2048.480	6154.100
$\eta_n$	0.667	0.573	0.535	0.514	0.499	0.494	0.488	0.487

TABLE VII

NUMERICAL VALUES RELATED TO THE MINIMIZED DURATION  $T_n$  OF THE BATCH CONFLICT WITH A PRIORI KNOWN MULTIPLICITY  $n$

Hence, the relation used to minimize  $T_n(x)$  is:

$$T_n(x) = \frac{P_0(x|n)B_0 + P_1(x|n)(B_1 + T_{n-1}) + P_2(x|n)(B_2 + T_{n-2}) + \sum_{m=3}^n P_m(x|n) \left[ B_m + \sum_{k=0}^{m-2} R(k|m)T_{n-m+k} \right]}{1 - P_0(x|n)} \quad (46)$$

We have found the values of  $x_n$  and  $T_n$  numerically for  $n \leq 3000$ . The results are shown in Table VII. The quantity  $x_n = \frac{z_n}{n}$ , while the time efficiency is  $\eta_n = \frac{n}{T_n}$ .

From Table VII it can be seen that, as  $n$  grows, the value of  $z_n$  approaches  $z = 1.266$  obtained for the FCFS algorithm. Furthermore, the efficiency  $\eta_n$  approaches the value 0.487, but contrary to the IECR algorithm, the efficiency monotonically decreases with  $n$  and approaches 0.487 from above. From this we can conclude that, for this particular algorithm (sequence of CBT executions with halving upon collision), the a priori knowledge of  $n$  cannot help to speed up the conflict resolution when  $n$  is very large. This further supports our conjecture that the achievable performance of the “best” batch conflict resolution algorithm for multiplicity  $n \rightarrow \infty$  cannot be improved by knowing  $n$  a priori.

## REFERENCES

- [1] I. Chlamtac, M. Conti, and J. Liu, “Mobile ad hoc networking: Imperatives and challenges,” *Ad Hoc Network Journal*, vol. 1, no. 1, Jan. 2003.
- [2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayrici, “A survey on sensor networks,” *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–116, Aug. 2002.
- [3] D. R. Hush and C. Wood, “Analysis of tree algorithms for RFID arbitration,” in *Proc. IEEE International Symposium on Information Theory*, Boston, USA, Aug. 1998.
- [4] I. Cidon and M. Sidi, “Conflict multiplicity estimation and batch resolution algorithms,” *IEEE Trans. Inform. Theory*, vol. IT-34, pp. 101–110, Jan. 1988.
- [5] J. F. Hayes, “An adaptive technique for local distribution,” *IEEE Trans. Commun.*, vol. 26, pp. 1178–1186, Aug. 1978.
- [6] J. I. Capetanakis, “Tree algorithms for packet broadcast channels,” *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 505–515, Sept. 1979.
- [7] B. S. Tsybakov and V. A. Mikhailov, “Free synchronous packet access in broadcast channel with feedback,” *Probl. Peredach. Inform.*, vol. 14, no. 4, pp. 32–59, Oct. 1978.
- [8] R. G. Gallager, “Conflict resolution in random access broadcast networks,” in *Proc. AFOSR Workshop Commun. Theory Appl.*, Provincetown, MA, Sept. 1978, pp. 74–76.
- [9] A. G. Greenberg, P. Flajolet and R. E. Ladner, “Estimating the multiplicities of conflict to speed their resolution in multiple access channels,” *J. ACM*, vol. 34, no. 2, pp. 289–325, Apr. 1987.
- [10] J. L. Massey, *Collision-Resolution Algorithms and Random-Access Communications*, ser. CISM Courses and Lectures. Springer-Verlag, 1981, no. 265, pp. 73–137.
- [11] N. D. Vvedenskaya and M. S. Pinsker, “Non-optimality of the Part-and-Try algorithm,” in *Abstracts Intl. Workshop Conv. Codes, Multiuser Comm.*, Sochi, USSR, 1983, pp. 141–148.
- [12] J. Mosely and P. Humblet, “A class of efficient contention resolution algorithms for multiple access channels,” *IEEE Trans. Commun.*, vol. COM-33, no. 2, pp. 145–151, 1985.
- [13] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge UK and New York: Cambridge University Press, 1995.
- [14] D. Berstekas and R. Gallager, *Data Networks*, 2nd ed. New Jersey: Prentice-Hall, 1992.

- [15] R. Rom and M. Sidi, *Multiple Access Protocols: Performance and Analysis*. New York: Springer-Verlag, 1990.
- [16] Y. C. Tay, K. Jamieson, and H. Balakrishnan, "Collision-minimizing CSMA and its applications to wireless sensor networks," *IEEE J. Select. Areas Commun.*, vol. 22, no. 6, pp. 1048- 1057, Aug. 2004.
- [17] M. L. Molle and G. C. Polyzos, "Conflict resolution algorithms and their performance analysis," Department of Computer Science and Engineering, UCSD, Tech. Rep. CS93-300, July 1993.
- [18] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley & Sons, 1994.
- [19] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11, 1997.