Batch Conflict Resolution Algorithm with Progressively Accurate Multiplicity Estimation

Petar Popovski Frank H. P. Fitzek Ramjee Prasad

Center for TeleInFrastructure (CTIF)
Aalborg University, Aalborg, Denmark
{petarp, ff prasad}@kom.auc.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 $n \to \infty$ 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.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless Communications; C.2.5 [**Local and Wide-Area Networks**]: Access schemes

General Terms

Algorithm, Design, Performance

DIALM-POMC'04, October 1, 2004, Philadelphia, Pennsylvania, USA.

Keywords

Conflict resolution, splitting tree, probing, sensor inquiry, multiplicity estimation

1. INTRODUCTION

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, attempt to simultaneously transmit replies to the interrogator. This gives rise to the batch arrival of the replies and a batch conflict of batch size n, where n is unknown to the interrogator. Here the value n is referred to as conflict *multiplicity*. The *conflict resolution algorithm* is performed by the interrogator and the terminals in order to successfully transmit (resolve) the packets of the initial conflict (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 conflict 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].

The performance of a conflict 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). BRI, denoted by T_n, is 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 conflict resolution algorithm should resolve conflict 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 conflict 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 conflict resolution. For example, it can be the average messages per terminal until the desired accuracy of the estimate of n is achieved.

The conflict 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 conflict 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 conflict resolution methods have been designed to resolve conflict among messages with Poisson arrivals. Such is the Gallager's FCFS algorithm [8], which achieves efficiency 0.487. However, the conflict 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 papers, 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 ($n \longrightarrow \infty$) 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.

The context of the work presented here can be summarized as follows. The conflict 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 conflict 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 conflict resolution. In this paper we will propose a class of batch conflict 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 conflict 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.

We will first show that the binary tree algorithms, without any modification, offer a very simple way to obtain a statistical estimate $\hat{n}$ during the resolution. The accuracy of such $\hat{n}$ increases as the number of resolved terminals increases. First we will use this method of estimating $\hat{n}$ to speed up the conflict resolution using the binary tree. Next, the estimating method will be applied to propose a more efficient algorithm termed *Interval Estimation Conflict Resolution (IECR)*. For $n \rightarrow \infty$ we prove that the efficiency achieved by IECR for batch arrivals of packets is identical with the efficiency that Gallager's FCFS algorithm achieves for Poisson arrivals of packets. 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.

2. 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 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 use the model of *slotted channel* [10]. The transmissions of the terminals start at predefined, equally spaced instants. The delay introduced by the wireless channel is considered negligible. The duration between two neighboring instants is denoted as *slot*. 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.

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 a packet sent by a terminal and the acknowledgement from the interrogator. In the probing scenario, the slot also accommodates probe packet at the slot start. In the model of slotted channel, the slot duration is the same regardless of whether the channel state is I, S, or C.

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.

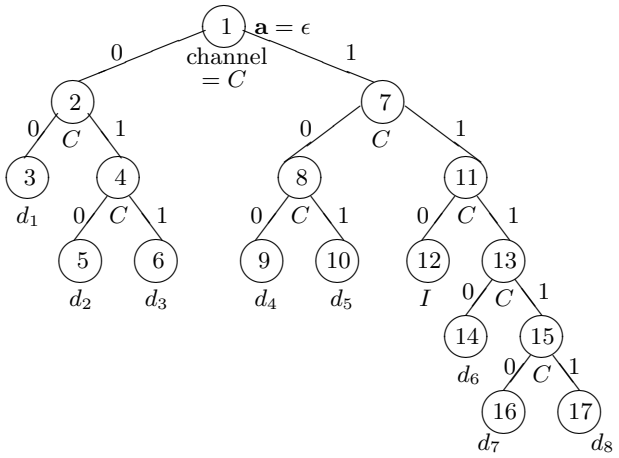

Figure 1: An instance of the binary tree algorithm for $n = 8$ terminals. The number denotes the slot in which a node is enabled. Below each node is the channel state in that slot. For channel state "single", d_i denotes the resolved terminal.

3. BINARY TREE ALGORITHMS AND MULTIPLICITY ESTIMATION

3.1 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 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, and 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 ϵ (empty string) and is at level 0. In fact, the initial conflict among the terminals occurs upon they receive probe with address ϵ 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 the First Come First Serve (FCFS) tree algorithm in [8]. The FCFS algorithm and its slight modifications 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 4, where we will show how to design batch resolution algorithm with the same performance 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 make such estimation and we will further use that estimation method to speed up the conflict resolution.

3.2 Real-Time Multiplicity Estimation with the 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 $\mathbf{b}_i = (b_{i1}b_{i2}b_{i3}\ldots)$ be the binary representation of fractional part of r_i. Then each $\mathbf{b}_i$ is an infinite string of 0s and 1s and:

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

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 $\mathbf{a} = (a_1a_2a_3\ldots a_L)$, where $L = l(\mathbf{a})$, if and only if $b_{ij} = a_j$ for all $j = 1\ldots L$. 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 $[0, 1)$:

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

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:

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

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:

$$r(\mathbf{a}) \le r_i < p(\mathbf{a}) \tag{4}$$

Now we show how the basic 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 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. $\mathbf{c} = (a_1a_2\ldots a_L0)$ and $r(\mathbf{c}) = r(\mathbf{a})$.

For the example on Fig. 1, 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. 1, 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 $\mathbf{a} = (a_1a_2\ldots a_{L-1}0)$, then only the trivial subtree $\mathbf{a}$ is resolved - a subtree that consists of node $\mathbf{a}$. If the resolved address is $\mathbf{a} = (a_1a_2\ldots a_{L-1}1)$, then there can be several resolved subtrees. A subtree $\mathbf{a}'$ is resolved when the address $\mathbf{a} = (a_1a_2\ldots a_{L-1}1)$ is resolved if and only if $\mathbf{a}' = (a_1a_2\ldots a_{L_1})$ 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 $\mathbf{a}^1, \mathbf{a}^2\ldots$ that are resolved when the address $\mathbf{a}$ is resolved have identical upper bound in the equivalent enabled interval $p(\mathbf{a}^1) = p(\mathbf{a}^2) = \ldots$

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:

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

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 make a *maximum-likelihood* (ML) estimate of n, estimated when $\mathbf{a}$ is resolved. The ML estimation $\hat{n}$ is computed from:

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

Let us denote:

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

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

$$\hat{n}_{ML} = \frac{k}{p} \tag{8}$$

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 $\delta > 0$:*

$$E[\hat{n}|n] = n \tag{9}$$

$$Var[\hat{n}|n] = \frac{n}{p} - n \tag{10}$$

$$P\left(\frac{\hat{n}}{n} - 1 \ge \delta|n\right) \le \frac{1-p}{\delta^2 np} \tag{11}$$

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

$$\mathrm{Var}[\hat{n}|n] = \mathrm{Var}\left[\frac{k}{p}|n\right] = \frac{\mathrm{Var}[k|n]}{p^2} = \frac{n}{p} - n \tag{12}$$

The inequality in (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. the 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.

```
a = ϵ; end= no; k = 0; Tx= no; collision_ind= no;
generate r_i;
while(end== no)
    if(r_i ∈ [r(a), p(a)));
        transmit; set Tx= yes;
    else
        set Tx= no;
    get feedback at the end of the slot;
    % current address is a = (a_1 a_2 ... a_L)
    if(collision)
        set a = (a_1 ... 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
            n̂ = k / p(a)
            K = log_2 n̂;
            M = L;
            while(a_M == 1) M = M − 1;
            for i = 1 ... M − 1 set c_i = a_i;
            set c_M = 1;
            if(K < M)
                while(c_K = 1) K++;
                delete c_{K+1} ... c_M;
            if(K > M)
                for i = M + 1 ... K
                    set c_i = 0;
            for i = 1 ... K a_i = c_i;
            delete a_{K+1} ... a_L;
```

Figure 2: The estimating binary tree (EBT) algorithm as run by the i-th node.

3.3 Estimating Binary Tree (EBT) algorithm

As a prelude to the EBT algorithm, 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 (or MBT) 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 $\mathbf{a}^0$ (i.e. address $\mathbf{a}^1$), then it can be inferred that $N(\mathbf{a}^1) \geq 1$ (i.e. $N(\mathbf{a}^0) \geq 1$). Since the n tokens are uniformly distributed in $[0, 1)$, the probability that i tokens fall in a particular interval of length 2^{-L} is:

$$P_L(i|n) = \binom{n}{i} 2^{-iL}(1 - 2^{-L})^{n-i} \tag{13}$$

The probability that the BT algorithm resolves address $\mathbf{a}$ at level L is given by:

$$\rho(L|n) = P_L(1|n) \cdot (1 - P_L(0|n)) \tag{14}$$

where the term $(1 - P_L(0|n))$ represents the fact that the sibling node of $\mathbf{a}$ contains at least one terminal. By using (13) in (14) we get:

$$\rho(L|n) = \frac{n}{2^L} \left(1 - \frac{1}{2^L}\right)^{n-1} \left(1 - \left(1 - \frac{1}{2^L}\right)^{n}\right) \tag{15}$$

We are interested in an integer value L that maximizes (15). Note that $\rho(L|n)$ is a concave function of L, such that it has only one maximum. Hence, we look for maximal value of L, denoted $L_{\max}$, for which the relation holds:

$$\frac{\rho(L|n)}{\rho(L-1|n)} > 1 \tag{16}$$

There is no simple exact expression for the integer $L_{\max}$ as a function of n, but in principle, $L_{\max}$ can be computed for each n. Yet, good approximation of $L_{\max}$ is:

$$L_{\max} = \lfloor \log_2 n \rfloor \tag{17}$$

which is almost exact for $n \leq 1000$. If we know $L_{\max}$, then we can design a more efficient tree algorithm by preferably enabling the nodes from level $L_{\max}$. Since the value n is not known, at least the terminals that are still unresolved can approximate the value of $L_{\max}$ as

$$L_{\max} = \lfloor \log_2 \hat{n} \rfloor \tag{18}$$

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.

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 1.

Table 1: 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 $l(\mathbf{c}) = K$.

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. 2 describes the EBT algorithm as a pseudocode, executed by the $i-$th terminal. The Fig. 3 depicts an instance of the EBT algorithm. The node 111 is not enabled due to the certain collision applied for MBT. In addition, due

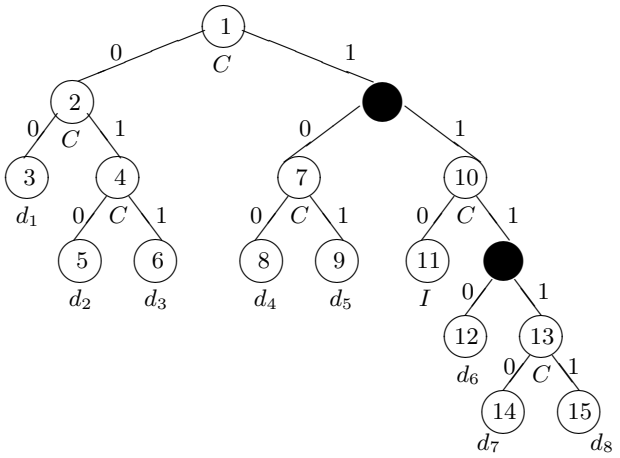

Figure 3: Instance of the EBT algorithm for $n = 8$. The black circle denotes that a node is not enabled during the algorithm execution. The terminals have the same tokens as in the example on Fig. 1

to the ML estimation of $\hat{n}$, in this instance of EBT algorithm the terminals conclude that node with $\mathbf{a} = 1$ leads to collision with high probability and hence it is skipped. 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 1. 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. 4 depicts the case when the address 010 is enabled two times in a single instance of EBT. In slot 5, the node 010 contains 3 terminals d_2, d_3, d_4 and a collision occurs. Note from Fig. 2 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 1 is $\mathbf{c} = 01001$. Step 2 of Table 1 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.

Another peculiarity of EBT is the rule for case (b) in step 3. Observe the EBT instance in Fig. 5. 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 1 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.

3.4 Practical details for the EBT algorithm

3.4.1 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 $\mathbf{a}$ 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 2 random bits and obtain sequences of type $\mathbf{b}_i = 1 b_{i2} b_{i3}$ and only the terminals with $\mathbf{b}_i = 100$ transmit in that slot.

3.4.2 Initiation of the conflict resolution

We have already stated that the interrogator initiates the conflict resolution by sending probe packet with address ϵ, which calls for reply from all neighboring terminals. Hence, a lot of 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 $\mathbf{a} = 00 \cdots 0$, where the address length is $l(\mathbf{a}) = \lfloor N_{\min} \rfloor$. 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 1.

3.4.3 EBT for Identity-based Conflict Resolution

Each responder has $\mathbf{b}_i = (b_{i1} b_{i2} \dots b_{iM})$ as a unique device address. 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

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

Figure 4: 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
a	ϵ	0	00	01	010	0110	01100	011000	011001	011	100	$\cdots$
channel	C	C	d_1	C	I	C	C	d_2	d_3	d_4	C	$\cdots$

Figure 5: An instance of the EBT algorithm to illustrate the rule 3b from Table 1

size of the device address. This permutation can be used to scramble the addresses prior to the start of the EBT.

4. INTERVAL ESTIMATION CONFLICT 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.

First, we briefly explain the operation of the FCFS algorithm [14]. 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 λ, where λ is known to the terminals. The conflict is resolved based on the packet arrival times. When a time epoch of length τ 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. 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 τ is "returned" to the arrival axis. This is possible since the collision from the left half wipes out the information about the right half, gained through the initial collision. Hence, the right half is 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. The optimal choice of τ is numerically obtained to be [15]:

$$\tau = \frac{1.26}{\lambda} \tag{19}$$

such that there are on average 1.26 arrivals in a freshly enabled interval. The maximal achievable throughput for this algorithm is $\lambda = 0.487$ and hence it is often referred to as 0.487 FCFS 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 for the moment assume that the first enabled interval in the algorithm is $s = [0, x)$ and let there be collision. Then, it can be concluded that there are $k_0 \geq 2$ tokens in s. Next, let $s_l = 0, \frac{x}{2}$ be enabled and let there be again collision. The gained information is that there are $k_l \geq 2$ tokens in s_l. Let k_r be the random variable that denotes the number of tokens in $s_r = \frac{x}{2}, x$, such that $k_0 = k_l + k_r$. Then the probability distribution of k_r, conditioned on the information gained so far is:

$$P_n(k_r = i | k_l \geq 2, k_r + k_l \geq 2) \stackrel{(a)}{=}$$

$$= P_n(k_r = i | k_l \geq 2, k_r \geq 0) \stackrel{(b)}{=}$$

$$= P(k_r = i | k_r \geq 0) \stackrel{(c)}{=} P(k_r = i)$$
$$= \binom{n}{k} \left(\frac{x}{2}\right)^k \left(1 - \frac{x}{2}\right)^{n-k} \tag{20}$$

In (20), 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 equation (c) follows again from non-negativity of k_r. Like in the FCFS, the collision in the subinterval s_l erases the information that the collision in s gives about the subinterval s_r. The subinterval s_r becomes statistically identical to the subinterval $[x, 1)$.

In fact, the conclusions stated by (20) are implicitly used in the EBT algorithm. Recall that, after finishing the 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 that 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 $x = 2^{-\lfloor \log_2 \hat{n} \rfloor}$ 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:

$$P(k) \doteq \frac{\Delta^k}{k!} e^{-\Delta} \tag{21}$$

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

$$\tau = \frac{\Delta}{\lambda} \tag{22}$$

From (19) it follows that for large n it should be $\Delta = 1.26$, while in the EBT algorithm we have $\Delta = n2^{-\lfloor \log_2 \hat{n} \rfloor}$. This observation gives a hint how to improve the EBT algorithm:

```
p_low = 0; p_up = 1; p_lim = 1;
end= no; k = 0; Tx= no; state=right;
generate r_i;
while(end==no)
    if(r_i ∈ [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 = (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 = (p_low+p_lim)/2;
            else
                p_up = p_lim;
                state=right;
        else
            Δ = 1.26p_lim/k
            p_low = p_lim;
            p_lim = min(1, p_low + Δ);
            p_up = p_lim;
            state=right;
```

Figure 6: The interval estimation conflict resolution (IECR) algorithm as run by the i-th node.

The terminals that are still unresolved should calculate the length of the next enabled subinterval to be

$$x = \frac{1.26}{\hat{n}} = \frac{1.26p}{k} \tag{23}$$

where k is the number of resolved tokens within $[0, p)$. Fig. 6 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 (23).

The choice (23) is suboptimal. In general, the value of Δ 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 Δ and the splitting, one can observe IECR as a Markov Decision Process and attempt to optimize it further. Nonetheless, such optimization is out of the scope of this paper.

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

THEOREM 1. *The time efficiency of the IECR algorithm satisfies*

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

The proof of the theorem is given in the appendix.

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. 6. To account for this case, the algorithm should be changed at the line in which $\Delta = \frac{1.26p_{lim}}{k}$ is set. Instead, Δ should be initialized at $\Delta = \frac{1}{N_{\min}}$ and while $k = 0$, Δ 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).

5. RESULTS

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.

Table 2 shows the speed of various conflict resolution algorithms, expressed through the efficiency η_n. Table 3 shows the average number μ_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 η_{10} for $N_{\min} = 500$), but it is not always the case (see η_{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 2 and 3 it can be inferred that, for the range of multiplicity $n \leq 1000$, picking $N_{\min} = 50$ is satisfactory.

The Table 4 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

Table 2: Time efficiency η_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 $N_{\min} = 500$
10	0.390442	0.412031	0.425351	0.417362	0.431965	0.432563	0.385416
50	0.379795	0.451712	0.452858	0.472313	0.462577	0.475321	0.467185
100	0.376736	0.455311	0.460473	0.470695	0.467181	0.479543	0.478593
500	0.375468	0.465272	0.460517	0.464126	0.480940	0.484302	0.485795
1000	0.375600	0.469228	0.463936	0.464516	0.482982	0.485356	0.485676

Table 3: Average number of messages per terminal μ_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 $N_{\min} = 500$
10	5.10020	4.49700	4.19100	2.47380	4.14600	2.55420	2.50360
50	7.45784	3.28600	4.55392	2.62232	4.44800	2.56524	2.54776
100	8.47456	3.17740	4.61668	2.65502	4.49380	2.54482	2.50940
500	10.79750	3.12772	4.74037	2.74230	4.48248	2.51318	2.47576
1000	11.80020	3.15586	4.77043	2.80559	4.46975	2.51264	2.47339

Table 4: 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	86.6	137.54
0.1	100	209.39	284.42
0.4	400	826.46	1027.56

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.

6. DISCUSSION

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.

The design of the EBT and IECR algorithm can be extended to CSMA channel model. In CSMA, the duration of the idle slots is much shorter than the time consumed by successful packet and collision. In such case, the idle response becomes "cheap" in terms of time, besides being "cheap" in messages. Intuitively, in this case the value x in (23) for IECR should be chosen much smaller, while the value of $L_{\max}$ in EBT should be much higher than (18). The CSMA design for IECR and EBT should follow the same guidelines from the CSMA design for FCFS, discussed in [14].

An interesting mechanism for partial resolution can be obtained if the CSMA version of IECR and EBT is combined with the SIFT algorithm from [16]. The SIFT algorithm provides efficient way to get the first reply from a set of terminals, where the upper bound on the number of terminals is known. Basically, SIFT determines what are the probing intervals that should be used after the interrogator initiates terminal replies. This can be used to replace the part where IECR or EBT start the procedure with known $N_{\min}$. If $1 < k < n$ replies are needed, then SIFT can be used to get the first reply and CSMA versions of the IECR or EBT can continue the resolution afterwards.

7. 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. Finally, we have designed the Interval Estimation Conflict Resolution (IECR) algorithm. IECR is shown to be the most efficient batch resolution algorithm among the algorithms reported so far in the literature. 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. We have already mentioned some

of them in the text. First, such is the task of ultimate optimization of the IECR algorithm by using the approach of Markov Decision Processes. Second, the approach presented here can be extended to the channels with carrier sensing. Third, it is interesting to consider the error-resilient conflict resolution by using the presented multiplicity estimation. 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 $n \to \infty$ can not be improved by knowing n in advance. The intuitive reason for this conjecture is that, when $n \to \infty$, the estimate of n can be made arbitrary close to n, such that the prior knowledge of n cannot improve the conflict resolution.

8. 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. Akylidiz, 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] P. F. A. G. Greenberg 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.

APPENDIX

[Proof of Theorem 1]

PROOF. We should show that, as n goes to infinity, the distribution of the number of tokens in interval $[p, p+x)$ where $p \in [0,1)$, approximates the Poisson distribution with average number of tokens $\Delta = 1.26$. In that case, the IECR algorithm moves across the interval $[0,1)$ with the same "speed" as the FCFS algorithm moves across the time axis, which means that the efficiencies of both algorithms are identical.

Let the interval $[0,p)$ be already resolved and let there be $k > 0$ tokens in it. The length x of the next enabled interval is chosen according to (23) and let us write it as:

$$x = \frac{\Delta}{\hat{n}}$$

where $\Delta = 1.26$. Assume that $p > 0$ and $p + x < 1$ - we will further account for the boundary cases. The distribution of the number of tokens within the interval of length x is given by:

$$P(k|n) = \binom{n}{k} x^k (1-x)^{n-k} = \frac{\Delta_1^k}{k!} e^{-\Delta_1} + o(1) \qquad (24)$$

where

$$\Delta_1 = nx = \Delta \frac{n}{\hat{n}} \qquad (25)$$

From (11) it follows that for any $\delta_1 > 0$:

$$P\left(|\Delta_1 - \Delta| \geq \delta_1 | n\right) \leq \frac{1-p}{\delta_1^2 np} \qquad (26)$$

Since for any k the function

$$f_k(y) = \frac{y^k}{k!} e^{-y} \qquad (27)$$

is continuous and $\lim_{y\to\infty} f_k(y) = 0$, then for each $\delta_1 > 0$ we can find δ_2 such that:

$$P\left(|f_k(\Delta_1) - f_k(\Delta)| \geq \delta_2 | n\right) \leq \frac{1-p}{\delta_2^2 np} \qquad (28)$$

We sketch the proof for the boundary cases. The Poisson approximation is not correct if $p + x \geq 1$. However, since x goes to 0 as $\frac{1}{n}$, we can say that the Poisson approximation is asymptotically valid for p arbitrary close to 1. Finally, the IECR includes the initial CBT algorithm before it starts to operate with estimation of $\hat{n}$. The average number of resolved tokens within the initial CBT algorithm has upper bound 2.6 as n goes to infinity. This fact can be shown by using the recurrent formulas from [15] and the bounding techniques presented in [10]. Hence, the initial CBT algorithm resolves the interval $[0, q_n)$, where $q_n \doteq \frac{2.6}{n} \to 0$ and p can be arbitrary close to 0. Consequently, the IECR algorithm becomes equivalent in efficiency to the FCFS over the complete interval $[0,1)$. $\square$