UNIVERSITY OF PADUA
DEPARTMENT OF INFORMATION ENGINEERING
MASTER DEGREE IN COMPUTER ENGINEERING

A.A. 2009/2010

BATCH SIZE ESTIMATE

SUPERVISOR: *Prof. Andrea Zanella*
STUDENT: *Marco Bettiol*

Last Update: Thursday 4th March, 2010 19:24

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Generally speaking a set of actors contending for a common resource define a *conflicting set*. As always, limited resources require policies to access them in an efficient and hopefully fair way. When the system is distributed, and this is our business, resource access can be assimilated to a coordination problem.
Physical medium access is our contended resource and several stations connect to the same physical medium to share it.
At the beginning of computer networks, multiple access channel (MAC) was a big issue for efficient communications: so packet switched buffered networks were introduced reducing the conflicting set to only two stations and simplifying the original problem. But switched networks could be realized only because the medium was wired: in fact a large collision domain could be sliced into smaller pieces and joined again together in ring, star or mesh structures.
In a wireless context the problem can be no more avoided.

Nowadays wireless connectivity in pervasive computing has ephemeral character and can be used for creating ad-hoc networks, sensor networks, connection with RFID (Radio Frequency Identification) 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 RFID tags that have a certain property, estimating the mean value contained in a group of sensors etc. Such an inquiry solicits replies from possibly large number of terminals.

In particular we analyze the scenario where a reader broadcast a query to the in-range nodes. Once the request is received devices with datas of interest are all concerned in

transmitting the information back to the inquirer as soon as possible and, due to the shared nature of the communication medium, collision problems come in: only one successful transmission at time can be accomplished, concurrent transmissions result in noise and an inefficient waste of energy/time since the channel is going to be occupied for some more time in future. This data traffic show a bursty nature which is the worst case for all shared medium scenarios.

This problem is referred in literature with different names: *Batch Resolution Problem*, *The Reader Collision Problem*, *Object Identification Problem.*
Algorithms trying to solve efficiently this problem are defined as *Batch Resolution Algorithms* (BRA) or *Collision Resolution Algorithms* (CRA).

For our terminology, given a query, it determines a subset of nodes which have to reply to it with one (and only one) message: this set of nodes constitute our *batch*. The size of the batch can ben know in advance, in lucky and optimistic scenarios, or can change in function of the query or time.

Since each node has exactly one message to deliver, the problem of obtaining all the messages or counting the number of nodes involved by the resolution process is exactly the same.

Instead the problem differs when we are not so interested in the exact number of nodes but we would appreciate a estimate of the actual batch size, rather accurate if possible.

The knowledge of the batch size n is an important factor for parameter optimization, for efficient resolution trying to minimize the time taken by the process.

1.1 Model, Scenario, Terminology

We consider the following standard model of a multiple access channel. A large number of geographically dispersed nodes communicate through a common channel. Any node generate and transmit data on the channel. Transmissions start at integer multiples of unit of time and last one unit of time, also called a "slot".

For this condition to be true in SLOTTED ALOHA, there must be some form of synchronization that inform the nodes about the beginning of the new slot (or at least the beginning of a cycle of slots). Slot size if equal to the fixed packet transmission time and a node can start a transmission only at the beginning of the slot, otherwise it will

stay quiet until the next slot to come.

In CSMA networks each node is able to determine the beginning of a new slot by sensing to the channel: when the channel is listened free a device can start transmitting it's message. In our scenario we assume that all the transmitted messages have a fixed length. Once a node has started transmitting it can sense no more to the channel and so it cannot be aware of the result of its transmission until it receives feedback. For this reason we have that a transmission always takes the same time, whether it is results in a success or a collision. On the other hand empty slots takes less time than transmissions.

CSMA/CD is not suitable for pervasive wireless devices such as sensors or RFID tags since they has to be keep as simple as possible to satisfy energy and cost requirements: they do not implement this MAC scheme and so no detection/reduction of collision time is possible.

We assume that there is no external source of interference and that a transmission can fail only when a collision takes place. In each slot, when k nodes transmit simultaneously, the success or the transmission depends on k:

- if $k = 0$ then no transmission were attempted. The slot is said to be *empty* or *idle*;
- if $k = 1$ then the transmission succeeds. The slot is said to be *successful*;
- if $k \geq 2$. there is a conflict, meaning that the transmission interfere destructively so that none succeeds. The slot is said to be *collided*.

We assume that nodes having new messages to deliver generated after the start of the resolution process will wait until the end before proceeding with their delivery.

Chapter 2

Batch Resolution

The general idea is as follows: the reader probes a set of nodes, and the nodes reply back. Since devices we consider operate in the wireless medium, collisions will result whenever a reader probes a set of nodes. The batch resolution algorithms use anti-collision schemes to resolve collisions. There are many algorithms that enable batch resolution, and these can be classified into two categories: (a) *probabilistic*, and (b) *deterministic*.

In *probabilistic algorithms*, a framed ALOHA scheme is used where the reader communicates the frame length, and the nodes pick a particular slot in the frame to transmit. The reader repeats this process until all nodes have transmitted at least once successfully in a slot without collisions.

Deterministic algorithms typically use a slotted ALOHA model, where the reader identifies the set of nodes that need to transmit in a given slot, and tries to reduce the contending batch in the next slot based on the result in the previous slot. These algorithms fall into the class of tree-based algorithms with the nodes classified on a binary tree based on their id, and the reader moving down the tree at each step to identify all nodes.

Deterministic algorithms are typically faster than probabilistic schemes in terms of actual node response slots used, however, they suffer from reader overhead since the reader has to specify address ranges to isolate contending tag subsets using a probe at the beginning of each slot.

Deterministic schemes assume that each node can understand and respond to complex commands from the reader, such as responding only if the *id* is within an address range specified by the reader. So not every device is able to support this class of algorithms. For

example passive tags, which are the most dummy devices, cannot understand this kind of requests and will continue to transmit in every resolution cycle, which lengthens the total time needed. Wireless sensors, semi-active and active tags should allow to implement tree-based algorithms: the reader can acknowledge nodes (immediate feedback) that have succeeded at the end of each frame, and hence those nodes can stay silent in subsequent slots, reducing the probability of collisions thereby shortening the overall identification time. Usually a node that successfully transmit its message and it? stays in silent until the end of the algorithm is said to be *resolved*.
They also assume a slotted model, and not a framed model, wherein the reader responds before and/or after every slot, adding overhead to the resolution.
Furthermore, since tree algorithms require explicit feedback about channel status, they force devices to be alway active and listening to the channel in each step of the algorithm. On the other hand windows based algorithms are more energy saving since a device can sleep for most of time in the transmission window and only to wake up in the slot it has decided to transmit. In a windows of w slots a node will be up only for $1/w$ of time and wait for feedback at the end of the window.

Most of the batch resolution algorithm were originally developed for ALOHA based scenarios.
These algorithms can be flawlessly ported to the CSMA scheme:

- by utilizing empty slots as "ALOHA slot delimiters"
- by explicitly sending a slot delimiter marker.

2.1 Binary Tree Algorithms

Basic binary tree algorithm was first introduced by Capetanakis [5] in 1979.

2.1.1 Basic Binary Tree

At slot τ we have a batch $\mathcal{B}$ of size n.
When a batch resolution process starts, initially all the nodes try to transmit and we can have 3 different events: *idle*, *success*, *collision*.
The supervisor broadcast the result of the transmission to all the nodes.
If we get *idle* or *success* events the resolution process stop meaning respectively that there were no nodes to resolve or there was only one node and that node's message was successfully received. That node delivered its message and will no longer take part in the

current batch resolution.
If we got a *collision* we know that at least 2 nodes are present and we have to solve the collision to obtain their messages. In this case all the n nodes play the algorithm.
Each node choose to transmit with probability p and to not transmit with probability $1-p$. Nodes that choosed to transmit are said to own to set $\mathcal{R}$ while the others to set $\mathcal{S}$. Of course $\mathcal{R} \cap \mathcal{S} = \emptyset$ and $\mathcal{B} = \mathcal{R} \cup \mathcal{S}$
Nodes in $\mathcal{S}$ wait until all terminal in $\mathcal{R}$ transmit successfully their packets, then they transmit.
Nodes in $\mathcal{R}$ are allowded to transmit in slot $\tau + 1$.

Intuitively we can think that choosing with equal probability ($p = 1/2$) between retransmitting or waiting can be a good choice. This is the case, since the algorithm is in some sense "symmetric", but this is not true in general, as we will see for MBT. Since $p = 1/2$ we can think to simply toss a coin to split the batch.

Algorithm 1 COLLISION BINARY TREE ($\mathcal{B}$)

// current slot status can be *idle*, *success*, *collision*
Input: $\mathcal{B}$ batch with $|\mathcal{B}| = n$
each node transmit its message
if (*idle* or *success*) **then**
conflict resolution ended.
else
each node flip a coin
$\mathcal{R} \leftarrow$ { nodes that flipped head}
$\mathcal{S} \leftarrow$ { nodes that flipped tail}
COLLISION BINARY TREE ($\mathcal{R}$)
COLLISION BINARY TREE ($\mathcal{S}$)
end if

Let L_n be the expected running time in slots required to resolve a conflict among n nodes using SBT. Let $Q_i(n) = \binom{n}{i} p^i (1-p)^{n-i}$ the probability that i among n nodes decide to transmit in the next slot (probability that $|\mathcal{R}| = i$). So if i nodes decide to transmit we have first to solve a conflict of size $|\mathcal{R}| = i$ with expected time L_i and later a conflict of size $|\mathcal{S}| = n - i$ with expected time L_{n-i}. L_n is given by the cost of the current slot (1) plus the expected time to solve all the possible decompositions of the current set. L_n can be recursively computed (considering the factorial in $Q_i(n)$) collecting L_n in the

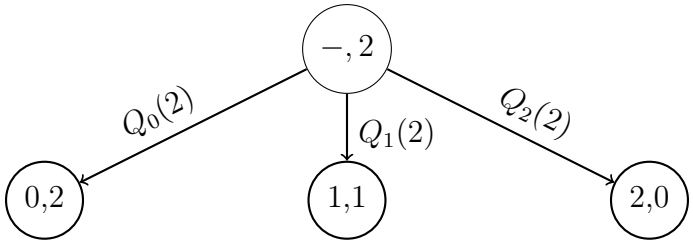

Figure 2.1: Transaction probabilities to split a set of 2 elements into two sets with i, j elements

following:

$$L_n = 1 + \sum_{i=0}^{n} Q_i(n)(L_i + L_{n-i}) \tag{2.1}$$

with

$$L_0 = L_1 = 1$$

To obtain an upper bound on the expected time as $n \to \infty$ further analysis techniques has to be used but here we want simply focus on how the algorithm behaves when n grows.

$L_2 = 5.0000$	$L_7 = 19.2009$	$L_{12} = 32.6238$	$L_{17} = 48.0522$	$L_{22} = 62.4783$
$L_3 = 7.6667$	$L_8 = 22.0854$	$L_{13} = 36.5096$	$L_{18} = 50.9375$	$L_{23} = 65.3636$
$L_4 = 10.5238$	$L_9 = 24.9690$	$L_{14} = 39.3955$	$L_{19} = 53.8227$	$L_{24} = 68.2489$
$L_5 = 13.4190$	$L_{10} = 27.8532$	$L_{15} = 42.2812$	$L_{20} = 56.7078$	$L_{25} = 71.1344$
$L_6 = 16.3131$	$L_{11} = 30.7382$	$L_{16} = 45.1668$	$L_{21} = 59.5930$	$L_{26} = 74.0198$

Considering the efficiency $\eta_n = n/L_n$ (messages over slots) we have a decreasing serie $\eta_1 = 1$, $\eta_2 = 0.40$, $\eta_3 = 0.3913$, ..., $\eta_{16} = 0.3542$, ..., $\eta_{31} = 0.3505$. It can be shown [5] that $\eta_\infty \approx 0.347$.

Since the algorithm is much more efficient in solving small batches respect to large ones we would prefer to have (ideally) n batches of size 1 rather than 1 batch of size n. So knowing exactly the cardinality n of the initial batch $\mathcal{B}$ can be used to split the nodes into small groups and resolve them faster.
This is the idea behind many improvements over the basic binary tree algorithm and it shows the importance of having an accurate estimate of n when the cardinality is initially unknown.

Example

In Figure 2.2 we provide an example to further investigate the behavior of the algorithm. We notice that the instance start with a collision in slot 1. Then nodes n_1, n_2, n_3 decide

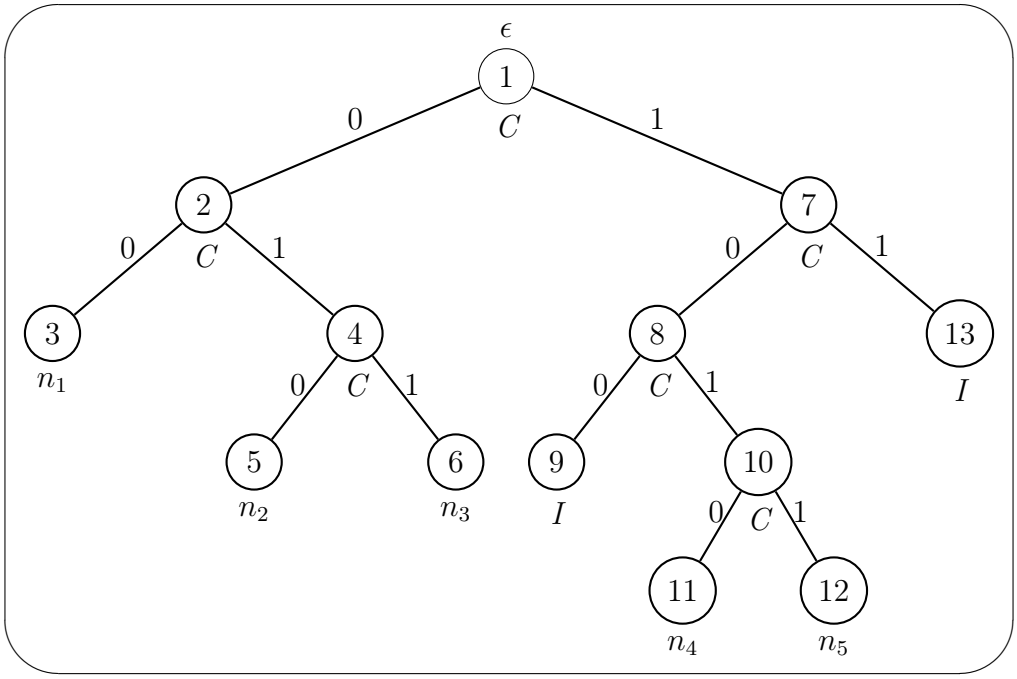

Figure 2.2: An istance of the binary tree algorithm for $n = 5$ nodes. The number inside the each circle identifies the slot number. The label below identifies the event occurring: I for *idle*, C for *collision*, n_i for resolution of node i. 0/1 branches is analogous to head/tail.

.

to proceed with a retransmission while n_4, n_5 remain idle. In slot 2 we see another collision, after it n_1 decide to transmit again while n_2 and n_3 to stay quiet. In slot 3 we have the first resolution, n_1 send successfully its message and won't no more take part to the collision resolution.
We notice that we can know the cardinality of a collision only after it has been fully resolved. For example we know only after slot 6 that the collision in slot 2 involved 3 nodes.

Nodes addresses

Looking carefully to the tree you can see that each node resolved is characterized by an *address*: the path from the root to node n_i gives a string of bits. For example node n_4's address has as prefix 1010. The prefix in this case can be equivalent to the address but, in a more general case, node address can be a longer string. Assuming in fact that node n_4's full address is the 8 bit long string 10100010, running the algorithm brings to the discovery of only the first 4 bits since the collision become resolved without requiring further split of the batch and deeper collision tree investigation (collision in level t provokes a split and a deeper investigation in the tree at level $t + 1$ and it requires to

consider bit $t+1$ of the nodes' addresses).

Tree traversal rules

The inquirer must provide feedback about the event in a slot but tree walking can be either explicit or implicit. It is explicit if, with feedback, the reader provide also the address in the root of the currently enabled sub-tree. Otherwise it is said to be implicit and each node compute autonomously the new enabled sub-tree.

We assume, following the conventional approach, to visit the tree in pre-order, giving precedence to sub-trees starting with 0.

Initially all nodes are enabled so the prefix is the empty string $\epsilon = b_{1..0}$, the root address. ϵ is considered to be prefix of any string.

Let $b_{1..k}$, with $b_i \in \{0,1\}$, $k \geq 0$, be the current enabled k-bit prefix and $event \in \{I, S, C\}$. The possible cases are: qui incasino un po' le cose con una notazione un po' imprecisa

i. *event* is C: no matter about $b_{1..k}$, next enabled interval will be $b_{1..k}0$;

ii. *event* is not C and $b_k = 0$: we successfully resolved the left part of the sub-tree, now we will look for right one. Next enabled prefix will be $b_{1..k-1}1$;

iii. *event* is not C and $b_k = 1$: we completed the resolution of a left sub-tree, now we will look in the way back to the root for the first right sub-tree still unresolved. Let t be $\arg\max_{i \in 1..k} | b_i = 0$ (or $t \leftarrow 0$ if $b_{1..k}$ having 1 or more 1), in other words the position of the right most 0 in the prefix, if any. The new enabled interval will be $b_{t-1}1$. You can see this rule applied after slot 6 and 12 in the example;

iv. termination condition is checking $b_{1..k} = \epsilon$.

Real value approach

Decidere se inserire o meno le considerazioni sulla visione degli indirizzi (alias ID) dei nodi come numeri reali tra 0 e 1 e della risoluzione come intervalli reali abilitati contenenti un solo nodo. Utile per collegarsi a popovski/cidon e alla suddivisione in insiemi in generale

Every length binary string can be also interpreted as a real number in the interval $[0, 1)$

$$11001 \leftrightarrow 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5}$$

by associating to each position in the string a different power of 2.

In general a given a string $\mathbf{b_i} = (b_{i1} b_{i2} b_{i3} \ldots)$, with $b_{ij} \in \{0,1\}$, can be associated to a

real number $r_i \in [0, 1)$ by a bijective map r:

$$r_i = r(\mathbf{b_i}) = \sum_{j=1}^{\infty} \frac{b_{ij}}{2^j} \tag{2.2}$$

So we could think, instead of tossing a coin only when needed, to initially flip a coin, in the same manner, L times to get a L-bits randomized string. In this way each node can be immediately be assigned to a set of length 2^{-L}. There are 2^L relatively ordered distinct sets in the interval [0,1).
Given a finite control string $\mathbf{a_i} = (a_{i1}a_{i2}\dots a_{ik})$, it enables all the nodes identified by real number x within the interval:

$$r(\mathbf{a_i}) \leq x < r(\mathbf{a_i}) + 2^{-k} \tag{2.3}$$

QUESTA OSSERVAZIONE é SOLO FRUTTO DEL MIO SACCO E MI SEMBRAVA INTERESSANTE.
An interesting observation is that the distribution of the nodes into the real interval depends upon p, the probability to obtain 0 or 1 tossing a biased coin. è interessante perchè per il basic binary tree p ottimo è 0.5 per cui si ottiene una distribuzione sperabilmente uniforme dei nodi (o poisson?). Mentre 0.5 non è ottimo per il Modified binary tree: p ottimo 0.4175. quindi la distribuzione migliore per il MBT è una specie di esponenziale discreta e la profondità dell'albero aumenta più ci si avvicina a 1. Questo è quello che mi dice l'intuizione e non ho visto scritto da nessuna parte (magari sul paper orginale del MBT c'è). per cui il MBT non può essere utilizzato banalmente per fare stime tramite una risoluzione parziale di un qualunque sotto intervallo $[0, x)$ con k nodi poichè $n \neq \frac{k}{x}$ (popovski) a meno che non sia nota la distribuzione dei nodi $f(x)$e si normalizzi per $f(x)$ al posto che x. NOTA: in popovski pg 295 dicono *To summarize, we can say that without any modification, the BT (or the MBT) algorithm offers a way to estimate the unknown conflict multiplicity.* il che è ok ma solo se MBT usa p=0.5 per cui $f(x) = x$. studiare $f(x, p)$?

2.1.2 Modified Binary Tree

Modified binary tree is a simple way to improve the basic variant for the binary tree algorithm.
The observation is that, during the tree traversal, sometimes we know in advance if the next slot there will be collided. This happens when, after a collided slot (τ), we get and

idle slot $(\tau+1)$ in the left branch of the binary tree: visiting the right branch $(\tau+2)$ we will get a collision for sure.
In fact in slot (τ) we know that in the sub-tree there are at least 2 nodes and none of them owns to the left-branch sub-tree $(\tau+1)$. So they must be in the right sub-tree and when enabled to transmit $(\tau+2)$ transmissions will disrupt. Solution is to keep previous node $(\tau+2)$ as a virtual node, to skip it, and continue visiting node $(\tau+1).sibling.leftchild$ in slot $(\tau+2)$. This let us save a slot.

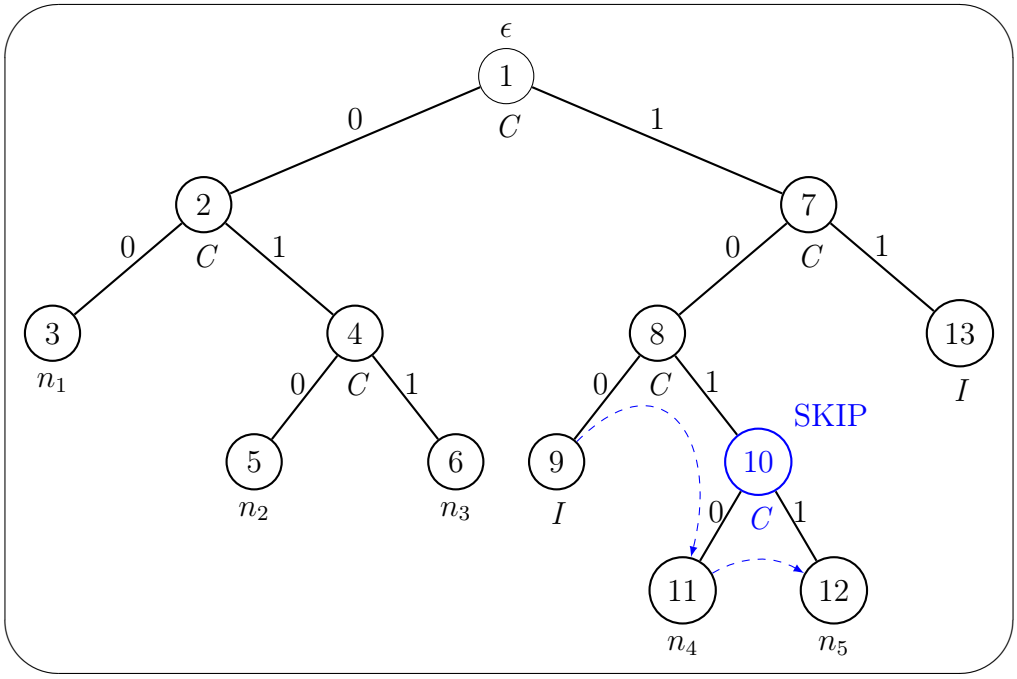

Figure 2.3: Same example as in Figure 2.2 but using MBT: tree structure do not change but node 10 is skipped in the traversal. $\tau = 8$

Expected time analysis is analogous to section 2.1.1. The only difference is that after a collision, if we get and idle slot, we will skip the "next one" (and we won't pay for it). So we can see that the expected slot cost is $[1 \cdot (1 - Q_0(n)) + 0 \cdot Q_0(n)]$. Then

$$L_n^{MBT} = (1 - Q_0(n)) + \sum_{i=0}^{n} Q_i(n)(L_i^{MBT} + L_{n-i}^{MBT}) \tag{2.4}$$

with

$$L_0^{MBT} = L_1^{MBT} = 1$$

Intuitively in this case, since an higher probability to stay silent, reduces the expected slot cost, optimal transmit probability won't no more be 1/2. At the same time lowering the

transmit probability will increase the number of (wasted) idle slots. So the new optimal probability p will be somewhere in the interval (0,1/2).
It can be shown[1] that best achievable result is for $p = 0.4175$ and, with this p, efficiency $\eta \approx 0.381$ as $n \to \infty$ which is asintotically +10% faster than basic BT.
In general we have:

$$L_n \leq C \cdot n + 1 \qquad \text{where} \qquad C = 2.623 \tag{2.5}$$

Not using optimal probability for p but $1/2$ results in about 1.5% peak performance loss which is a moderate decrease.

[1] J.L Massey,*Collision-Resolution Algorithms and Random-Access Communications*, CISM Courses and Lectures, vol. 265, pp. 73–137. Springer, Berlin (1981)

Chapter 3

Batch Size Estimate Techniques

We present here some noteworthy techniques for batch size estimate that can be found in literature. If the technique was not already identified by a name or associated to a acronym we used the name of one authors as reference.

In general, we assume not to have any *a priori* statistical knowledge about the multiplicity of the nodes involved in a collision. So estimation techniques must provide efficient ways to obtain an estimate for the general zero-knowledge scenario.

3.1 CBT

The most simple idea to obtain an estimate of the batch size could be to solve a minimum amount of nodes to obtain an estimate. This can be done, for example, by using deterministic algorithms such as CBT.
CBT is a partial resolution algorithm since only a fraction of the packets of the batch are successfully transmitted. The clipped binary tree algorithm is identical to the modified binary tree algorithm (with $p = 1/2$ since we require the nodes to be uniformly distributed in the interval [0,1)) except that it is stopped (the tree is clipped) whenever two consecutive successful transmissions follow a conflict.
When the algorithm stops we know than the last two nodes resolved owns to the same level i of the tree (root is at level=0).
We could think to obtain an estimate as:

$$\hat{n} \leftarrow 2^i \tag{3.1}$$

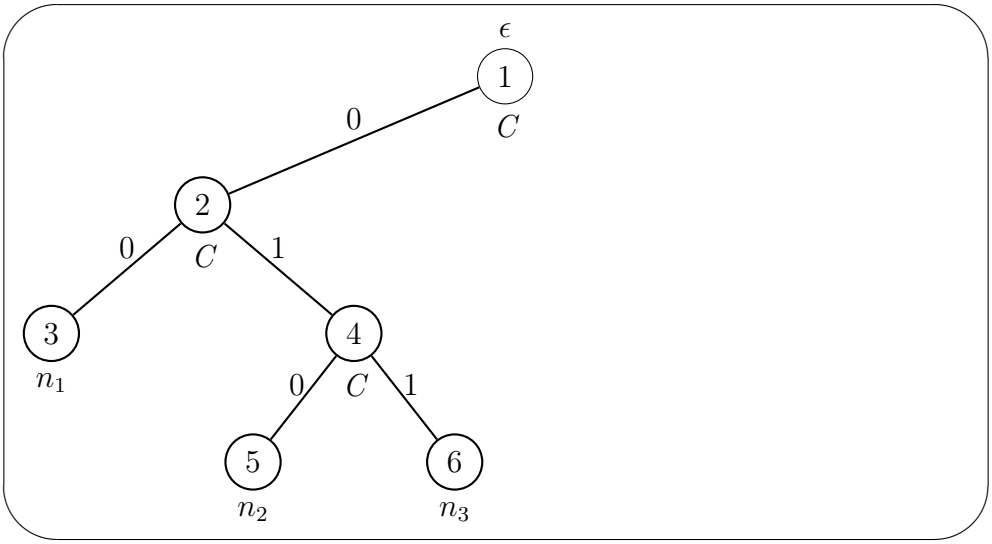

Figure 3.1: Same example as in Figure 2.2 but resolution using CBT ends up after two consecutive successful transmissions.

Experimental results show that the variance of the obtained estimate is extremely high and the resulting accuracy is really poor.
This is due to the fact that the batch of interest we use for the estimate becomes, at each level, smaller and smaller: the estimate, even for huge sizes, depends only on very few (3-5) nodes (those with lower addresses). So estimate is quite unstable.
Results are reported in Appendix in tables A.1. Notice how slowly the distribution probability decrease.

3.2 Cidon

Cidon and Sidi proposed this approach in [3]. In this work they describe a complete resolution algorithm based on two phases:

1. to get an estimate of the initial batch using a partial deterministic resolution scheme.
2. to perform an optimized complete deterministic resolution basing on the results of phase 1.

The strategy adopted to obtain the estimate is to resolve a small portion of the batch and to accumulate the number of successful transmission resulted.

Initially we fix a probability p_ϵ that determines how the whole batch is split at the very first time.

We named it p_ϵ to underlined than this initial choice reflects on the expected accuracy of the resulting estimate.

As usual we have initially a batch $\mathcal{B}$ of unknown size n. At the beginning of the algorithm each node chooses to transmit with probability p_ϵ. Thus the n nodes are partitioned into two set $\mathcal{E}$ and $\mathcal{D}$, where $\mathcal{E}$ consist of those that transmitted and $\mathcal{D}$ the rest. Clearly, $|\mathcal{E}| + |\mathcal{D}| = n$. If the resulting slot is empty or contains a successful transmission, we conclude that $|\mathcal{E}| = 0$ or $|\mathcal{E}| = 1$, respectively. If a conflict occurs, it is known that $|\mathcal{E}| \geq 2$, and the nodes in $\mathcal{E}$ use a complete batch resolution algorithm to resolve the conflicts among the nodes in $\mathcal{E}$. At the end of this part we know the exact value of $\mathcal{E}$ by accumulating the number of successful transmissions during the resolution. We call this counter j. So, after this estimation phase, since we expect that the nodes are uniformly distributed in the real interval [0,1) and we solved the first part of the interval from 0 to p_ϵ we found that our expected density can be supposed to be $\frac{j}{p_\epsilon}$. Then we simply compute our estimate $\hat{n}$ as:

$$\hat{n} \leftarrow \frac{j}{p_\epsilon} \tag{3.2}$$

In this case, since we already resolved the nodes in $\mathcal{E}$, we are more interested only in the cardinality of the remaining batch to solve $\mathcal{D}$ which we can compute as:

$$\hat{k} = E[size(\mathcal{D})] \leftarrow \frac{j}{p_\epsilon}(1 - p_\epsilon) \tag{3.3}$$

Algorithm 2 CIDON

Input: p_ϵ, fraction of the whole batch to solve

1: // Phase 1
2: each node flip a coin getting 0 with probability p_ϵ, 1 otherwise
3: $\mathcal{E} \leftarrow$ { nodes that flipped 0}
4: $\mathcal{D} \leftarrow$ { nodes that flipped 1}
5: COMPLETE COLLISION RESOLUTION ($\mathcal{E}$)
6: $\hat{k} \leftarrow |\mathcal{E}|/p_\epsilon$
7: // Phase 2
8: OPTIMIZED COMPLETE COLLISION RESOLUTION ($\mathcal{D}$,$\hat{k}$, p_ϵ)

Note that $|\mathcal{E}| = 0$ does not imply $|\mathcal{D}| = 0$ so a complete resolution algorithm has always to be performed on $\mathcal{D}$. Recalling subsection "Real Value Approach" in 2.1.1, operation on line 2 in Alg. 2 is equivalent to force each node to generate a unique ID expressed as a

fixed length real value in the interval [0,1).
Following Figure 3.2 show the case providing a simple example.

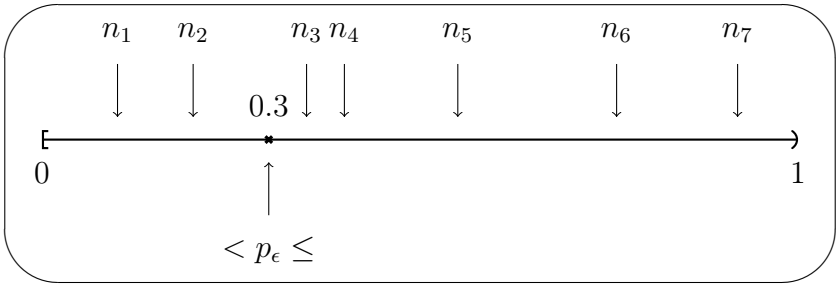

Figure 3.2: In this example $p_\epsilon = 0.3$. At the beginning of the algorithm each node generate its own ID. Nodes whose ID is less than p_ϵ owns to $\mathcal{E}$. Nodes whose ID is greater or equal to p_ϵ owns to $\mathcal{D}$. Estimate of the batch returns $\lceil 2/0.3 \rceil = 7$ which, in this case, is the exact size of the batch.

Procedure COMPLETE COLLISION RESOLUTION ($\mathcal{E}$) identifies any procedure able to resolve all the nodes in $\mathcal{E}$ allowing them to successfully transmit their messages while OPTIMIZED COMPLETE COLLISION RESOLUTION ($\mathcal{D}$,$\hat{k}$, p_ϵ) identifies an optimized way to resolve the batch $\mathcal{D}$: speedup is allowed by the knowledge of its expected multiplicity.

3.2.1 Estimate Accuracy

Let J be an integer random variable which expresses the number of nodes in $\mathcal{E}$. Given a batch of size n, J is binomially distributed with parameter p_ϵ. It can be thought as the probability distribution to put j among n nodes in two bins choosing with probability p_ϵ the first one and $1 - p_\epsilon$ the other one. Therefore, we have the following:

1)
$$P(J = j|n) = \binom{n}{j} p_\epsilon^j (1 - p_\epsilon)^{n-j} \tag{3.4}$$

2)
$$E[J|n] = np_\epsilon \tag{3.5}$$

3)
$$\text{var}(J|n) = np_\epsilon(1 - p_\epsilon) \tag{3.6}$$

By applying Chebychev's Inequality and remebering

$$\hat{n} = j/p_\epsilon \tag{3.7}$$

we can establish a relation between the relative error in the estimate and it's probability.

$$P\left(|\hat{n} - np_\epsilon| \geq \epsilon n \,|\, n\right) \leq \frac{1 - p_\epsilon}{\epsilon^2 n} \tag{3.8}$$

qua è da rivedere decisamente So if we want to obtain a relative error below ϵ with probability c we have to initially solve a fraction

$$p_\epsilon = 1 - c(\epsilon^2 n) \tag{3.9}$$

of the whole batch.

3.3 Greenberg

Greenberg algorithm is simply straightforward.

Algorithm 3 GREENBERG

$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  choose to transmit with probability 2^{-i}
until no collision occurs
$\hat{n} \leftarrow 2^i$
$\hat{n}_+ \leftarrow \hat{n}/\phi$

The idea behind algorithm 3 is quite simple. As the algorithm goes on the initial unknown batch size n comes progressively sliced into smaller pieces. Only the nodes virtually inside the slice are allowed to transmit.
If two o more nodes decide to transmit we get a collision,
An important note is that the algorithm always involve all the nodes in the batch: in each stage of the algorithm each node has to take a choice if transmit or not. Each choice is independent of what the nodes did in the previous steps.

Using Mellin's transform

$$\phi = \frac{1}{\log 2} \int_0^\infty e^{-x}(1+x) \prod_{k=1}^{\infty} (1 - e^{-2^k x}(1 + 2^k x)) x^{-2}\, dx \tag{3.10}$$

$$\Phi = \frac{1}{\log 2} \int_0^\infty e^{-x}(1+x) \prod_{k=1}^{\infty} (1 - e^{-2^k x}(1 + 2^k x)) x^{-3}\, dx \tag{3.11}$$

Table 3.1: Given a batch of size n the expected estimate applying base 2 Greenberg is $E[\hat{n}|n]$. The ratio $E[\hat{n}|n]/n$ monotonically decreases and gets stable at 0.9142. This shows that this estimate technique provide biased results.

n	$E[\hat{n}\|n]$	$E[\hat{n}\|n]/n$
1	2.00	2.0000
2	2.56	1.2822
4	4.21	1.0533
8	7.89	0.9863
16	15.20	0.9498
32	29.82	0.9320
64	59.08	0.9231
128	117.59	0.9186
256	234.60	0.9164
512	468.64	0.9153
1024	936.71	0.9148
2048	1872.86	0.9145
4096	3745.14	0.9143
8192	7489.72	0.9143
16384	14978.86	0.9142
32768	29957.16	0.9142
65536	59913.74	0.9142

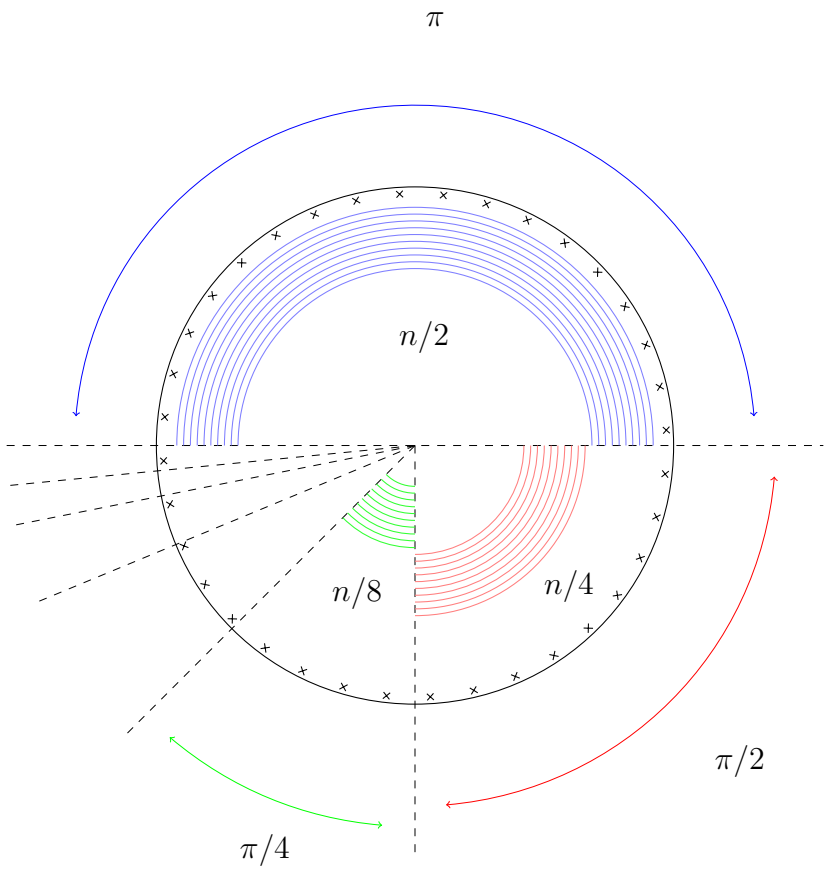

Figure 3.3: Visually nodes can be thought to be uniformly distributed on the circumference of a circle. By performing Greenberg's algorithm we go and analyze each time a smaller sector (in this case the half of the previous one) of the circle and find when a sector contains only 1 or no nodes. Not overlapping sectors are drawn to maintain the image simple but in general nodes gets redistributed to each step of the algorithm

3.4 Window Based

Chapter 4

Initial batch size estimate

We investigated about a fast algorithm for multiplicity estimation.

4.1 Cidon Evaluation

We remember from section 3.2.1 that

$$P(J = j|n) = \binom{n}{j} p_\epsilon^j (1 - p_\epsilon)^{n-j}$$

and

$$\hat{n} = j/p$$

Note that in Alg. 2 (Cidon) we have p_ϵ *a priori* fixed since it is an input parameter of the algorithm. So J is a binomial distribution with parameters $B(n, p_\epsilon)$ and recalling (3.5) we got:

$$E[\hat{n}|n, p_\epsilon] = \frac{1}{p_\epsilon} E\left[j|n, p_\epsilon\right] = n, \qquad \forall\, p_\epsilon \tag{4.1}$$

This shows that Cidon provides an unbiased estimator ($E[\hat{n}|n] = n$) independently from p_ϵ: p_ϵ influences only the variance of the estimator.

$$\begin{aligned} \mathrm{var}(\hat{n}|n) &= E[\hat{n}^2|n] - E[\hat{n}|n]^2 \\ &= \frac{1}{p_\epsilon^2} E[j^2|n] - n^2 \\ &= \frac{np_\epsilon(1 - p_\epsilon) + n^2 p_\epsilon^2}{p_\epsilon^2} - n^2 \\ &= n\frac{(1 - p_\epsilon)}{p_\epsilon} \end{aligned} \tag{4.2}$$

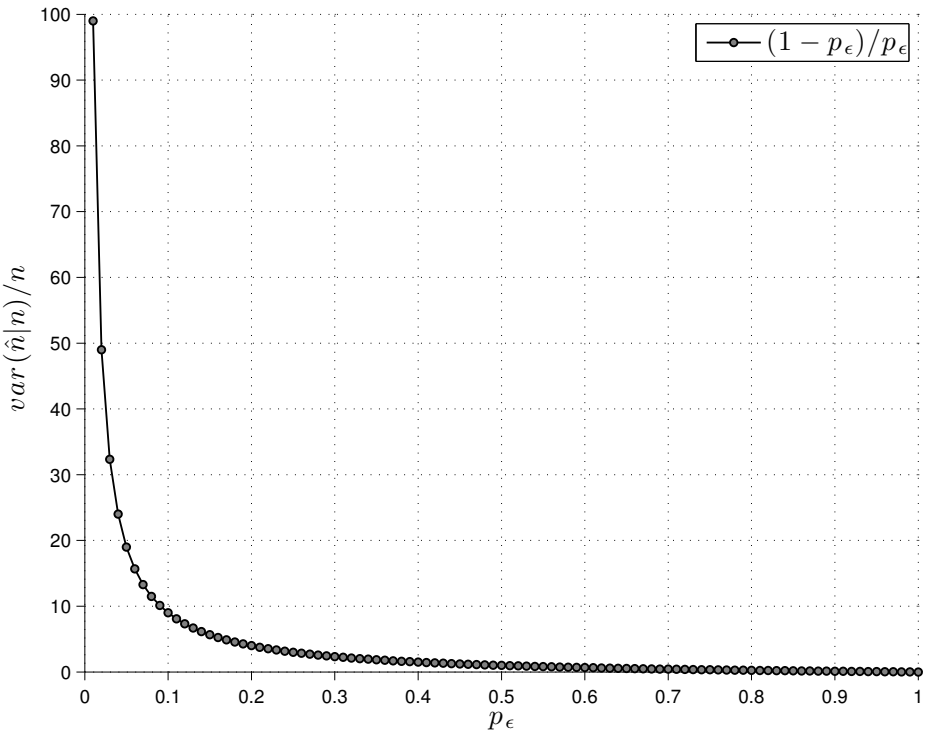

Figure 4.1: Cidon's estimate accuracy dramatically improves until $p_\epsilon = 0.1$.

Given k and n

$$P\left(\frac{n}{k} \leq \hat{n} \leq kn \middle| k, n\right) =? \tag{4.3}$$

$$P\left(\frac{n}{k} \leq j/p \leq kn \middle| k, n, p\right) \tag{4.4}$$

$$P\left(\frac{np}{k} \leq j \leq knp \middle| k, n, p\right) \tag{4.5}$$

J is $Binomial(n, p)$ which, under some conditions (precisare), can be considered $\mathcal{N}(np, np(1-p))$

k esprime dei bound per l'errore della stima, vogliamo trovare p affinchè con probabilità maggiore del 0.99 il vincolo sia rispettato

$$P\left(\frac{np}{k} \leq j \leq knp \middle| k, n, p\right) \quad = \quad P(j \leq knp) - P(j < \frac{np}{k}) \quad \geq \quad 0.99 \tag{4.6}$$

il cui valore, visto che k e n sono fissati, dipende solo da p

Risolvo per via numerica

$$solve\left(P(j \leq knp) - P(j < \frac{np}{k}) = 0.99\right) \rightarrow p_\epsilon \tag{4.7}$$

4.2 Greenberg Evaluation

Given a current slot transmission probability p and a batch of size n we define respectively:

1. the probability to get an empty slot (no transmissions)

$$q_0(p, n) = (1 - p)^n \tag{4.8}$$

2. the probability to get a successful transmission (one transmission)

$$q_1(p, n) = np(1 - p)^{n-1} \tag{4.9}$$

3. the probability to get a collision (two or more transmissions)

$$q_{2+}(p, n) = 1 - q_0 - q_1 \tag{4.10}$$

In basic Greenberg (*Alg.* 3) each slot is associated with a different probability p. Naming each slot i starting with 1, 2, ..., we have:

$$p_i = p(i) = 2^i \tag{4.11}$$

Given n nodes, the probability to terminate algorithm 3 in slot i is given by:

$$f(n, i) = \prod_{k=1}^{i-1} q_{2+}(p_k, n)\dot{(}q_0(p_i, n) + q_1(p_i, n)) \tag{4.12}$$

An overview of the behaviour of $f(n, i)$ is presented in table A.2.

4.2.1 base b Greenberg

Chapter 5

Comparison

Appendix A

A.1 Probability

sezione provvisoria

A.1.1 Binomial Distribution

$B(n, p)$

Poisson Approximation

The binomial distribution converges towards the Poisson distribution as the number of trials goes to infinity while the product np remains fixed. Therefore the Poisson distribution with parameter $\lambda = np$ can be used as an approximation to $B(n, p)$ of the binomial distribution if n is sufficiently large and p is sufficiently small. According to two rules of thumb, this approximation is good if $n \geq 20$ and $p \leq 0.05$, or if $n \geq 100$ and $np \leq 10$.

Normal Approximation

If n is large enough, then the skew of the distribution is not too great. In this case, if a suitable continuity correction is used, then an excellent approximation to $B(n, p)$ is given by the normal distribution $\mathcal{N}(np, np(1-p))$
The approximation generally improves as n increases and is better when p is not near to 0 or 1. Various rules of thumb may be used to decide whether n is large enough, and p is far enough from the extremes of zero or unity: One rule is that both np and $n(1-p)$ must be greater than 5. However, the specific number varies from source to source, and depends on how good an approximation one wants; some sources give 10.

oppure dal libro $np(1-p) \geq 10$.

A.1.2 Poisson Distribution

A.1.3 Normal Distribution

$\mathcal{N}(\mu, \sigma^2)$

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

A.2 Greenberg bounded m-moments

In general for base b greenberg algorithm the first and second moments are bounded by:

$$\phi(b) = \frac{1}{\log b} \int_0^\infty e^{-x}(1+x) \prod_{k=1}^{\infty} (1 - e^{-b^k x}(1+b^k x)) x^{-2}\, dx \tag{A.2}$$

$$\Phi(b) = \frac{1}{\log b} \int_0^\infty e^{-x}(1+x) \prod_{k=1}^{\infty} (1 - e^{-b^k x}(1+b^k x)) x^{-3}\, dx \tag{A.3}$$

note sul calcolo

va velocemente a 0 quind basta considerare un intervallo iniziale limitato

anche per produttoria con k è lo stesso.

risolto con quad matlab

A.3 CBT Estimate Experimental Distribution

Following tables A.1 shows the behavior of CBT Algorithm (section 3.1) for estimation. Simulation was implemented in matlab. The resulting distribution of $\hat{n}$ fixed n is the result of averaging 100'000 runs of CBT Algorihm applied on uniformly random generated nodes ID batches.

Table A.1: Experimentally computed CBT Estimate Distributon. Table 1/3

n	$\hat{n}$:	2	4	8	16	32	64	128	256	512	1024	2048
2		0.499	0.253	0.125	0.061	0.031	0.015	0.007	0.004	0.002	9e-04	4e-04
4			0.189	0.303	0.225	0.133	0.072	0.038	0.020	0.010	0.005	0.002
8			0.055	0.212	0.261	0.201	0.126	0.071	0.037	0.019	0.009	0.005
16			8e-04	0.070	0.209	0.252	0.197	0.125	0.070	0.038	0.019	0.010
32				0.003	0.075	0.213	0.249	0.195	0.123	0.069	0.037	0.018
64					0.004	0.077	0.208	0.250	0.193	0.124	0.069	0.037
128						0.005	0.081	0.208	0.247	0.191	0.123	0.069
256						2e-05	0.006	0.079	0.209	0.246	0.193	0.123
512								0.005	0.081	0.207	0.245	0.193
1024									0.005	0.080	0.208	0.245
2048									2e-05	0.005	0.080	0.209
4096										1e-05	0.006	0.082
8192											1e-05	0.006
16384												2e-05
32768												

Table A.1: Experimentally computed CBT Estimate Distributon. Table 2/3

n	$\hat{n}$:	4096	8192	16384	32768	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
2		2e-04	1e-04	1e-04				1e-05				
4		0.001	5e-04	3e-04	1e-04	6e-05	4e-05		2e-05	1e-05		
8		0.002	0.001	6e-04	3e-04	9e-05	8e-05	4e-05	1e-05			
16		0.005	0.003	0.001	6e-04	3e-04	2e-04	6e-05	2e-05	2e-05		
32		0.009	0.005	0.003	0.001	6e-04	3e-04	1e-04	1e-04	6e-05	1e-05	1e-05
64		0.019	0.009	0.005	0.002	0.001	7e-04	3e-04	7e-05	4e-05		2e-05
128		0.037	0.019	0.010	0.005	0.003	0.001	5e-04	4e-04	2e-04	5e-05	4e-05
256		0.068	0.038	0.019	0.009	0.005	0.002	0.001	6e-04	3e-04	6e-05	8e-05
512		0.123	0.071	0.037	0.019	0.010	0.005	0.002	0.001	6e-04	3e-04	2e-04
1024		0.193	0.122	0.070	0.037	0.019	0.010	0.005	0.002	0.001	6e-04	2e-04
2048		0.246	0.194	0.123	0.068	0.037	0.019	0.010	0.005	0.002	0.001	6e-04
4096		0.210	0.246	0.193	0.121	0.068	0.037	0.019	0.009	0.004	0.003	0.001
8192		0.080	0.208	0.247	0.192	0.123	0.070	0.037	0.019	0.009	0.005	0.002
16384		0.006	0.080	0.208	0.247	0.192	0.123	0.069	0.037	0.019	0.010	0.005
32768			0.006	0.079	0.209	0.248	0.194	0.122	0.069	0.036	0.019	0.010

Table A.1: Experimentally computed CBT Estimate Distributon. Table 3/3

n	$\hat{n}$:	2^{23}	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}	2^{31}	2^{32}
2											
4											
8											
16											
32		1e-05									
64		2e-05				1e-05					
128		4e-05		1e-05							
256		4e-05	1e-05		2e-05			1e-05			
512		7e-05	2e-05	1e-05	3e-05	1e-05					
1024		2e-04	1e-04	4e-05		2e-05	1e-05				
2048		3e-04	1e-04	3e-05	3e-05	3e-05	3e-05				
4096		6e-04	3e-04	9e-05	7e-05	1e-05					
8192		0.001	8e-04	3e-04	2e-04	8e-05	7e-05	2e-05	3e-05	1e-05	
16384		0.002	0.001	6e-04	3e-04	1e-04	1e-04	4e-05			
32768		0.005	0.002	0.001	6e-04	3e-04	2e-04	1e-04	1e-05	2e-05	1e-05

Here is the matlab code used for the simulations

matlab/CBT/cbtsimpletest.m

```
% Marco Bettiol - 586580 - BATCH SIZE ESTIMATE
%
% CBT Simple Test
%
% This script implements a simulation of the estimate obtained
% using CBT in a batch of size n.
%
% Nodes are initially uniformily picked-up in the interval [0,1)


clear all;
close all;
clc;

n=16; % batch size
disp(['Size :' int2str(n)]);

nodes=rand(n,1);
% virtual node with value 1 to get easier search algorithm
% among the nodes
```

```
21
22 % asc sorting
23 nodes=[sort(nodes); 1];
24
25 if (n<2)
26     error('BRA must start with a collision');
27 end
28 % CBT Simulation
29
30 % true if we got a success in the last transmission
31 lastwassuccess=false;
32 %false to end CBT
33 waitforconsecutive=true;
34
35 imax=length(nodes);  %index of the first node in the batch
36 imin=1;    % index of the first node in the batch
37 xmin=0;    % starting interval [0,1/2)
38 xlen=1/2;  % we suppose a collision already occurred.
39
40 while (waitforconsecutive)
41     [e,imin,imax]=cbtsplit(nodes,imin,imax,xmin,xlen);
42     % update next analyzed interval
43
44     if(e==1)
45         xmin=xmin+xlen;
46         %xlen=xlen;
47     elseif (e==0)
48         xmin=xmin+xlen;
49         xlen=xlen/2;
50     else
51         %xmin=xmin;
52         xlen=xlen/2;
53     end
54     if (lastwassuccess==true && e==1)
55         disp(' ');
56         disp('CBT completed :');
57         disp(['Estimate :' num2str(1/xlen)]);
58         disp(['Last node transmitting :' int2str(imin-1)]);
59         waitforconsecutive=false;
60     end
61     if(e==1)
62         lastwassuccess=true;
63     else
64         lastwassuccess=false;
65     end
66 end
67
68 % DEBUG
69 % estimate is given by the first serie of descending differences in the
70 % nodes ID's
71 dif=-1*ones(n-1,1); %negative init
72 for i=1:n-1
73     dif(i)=nodes(i+1)-nodes(i);
74 end
```

```
75
76 nodes
77 dif
```

matlab/CBT/cbtsplit.m

```
1  % Marco Bettiol - 586580 - BATCH SIZE ESTIMATE
2
3  function [e,imin,imax]=cbtsplit(nodes,imin,imax,xmin,xlen)
4
5  %
6  % CBT BATCHSPLIT
7  %
8  % [e,imin,imax]=cbtsplit(nodes,imin,imax,xmin,xlen)
9  %
10 % Finds the nodes possibily enabled in the future conflicting set given the
11 % current enabled interval [xmin,xmin+xlen) and establish the event type
12 % for the current enabled interval
13 %
14 % Input:
15 %
16 % nodes : asc ordered vector of the nodes
17 % imin  : index of the first node that collided
18 % imax  : index of the first node in the sleeping set
19 % xmin  : lower bound of the new enabled interval
20 % xmin  : higher bound of the new enabled interval
21 %
22 % Output:
23 % e     : event obtained
24 % imin : new index of the first node that collided
25 % imax : new index of the first node in the sleeping set
26
27 xmax=xmin+xlen;
28
29 % idle slot happens when imin is greater than current max allowded value.
30 % Future set of nodes to analyze do not change
31 if (nodes(imin)>=xmax)
32     e=0;
33     return;
34 end
35
36 % this is always false by algorithm construction
37 % used to verify a trivial condition
38 %while (nodes(imin)<xmin)
39 %    imin=imin+1;
40 %end
41
42 % if event is a success
43 if (nodes(imin)<xmax && nodes(imin+1)>=xmax)
44     e=1;
45     imin=imin+1;
46 else
```

```
% if event is collision
% update the next enabled set
    e=2;
    while ((imax-1)~=0 && xmax<nodes(imax-1))
        imax=imax-1;
    end
end
```

matlab/CBT/cbtfulltest.m

```
% Marco Bettiol - 586580 - BATCH SIZE ESTIMATE
%
% CBT Full Test
%
% Estimate distributions obtained with CBT fixed different n
%
% Based on cbtsimpletest code

clear all;
close all;
clc;

% input
logn_max=15;
c=100000;

% generate the test batch size
x=1:logn_max;
testsizes=2.^x;

% resulting estimate distribution
% ED(log2(batch size), log2(estimate batch size));

ED=zeros(length(testsizes),length(testsizes)+40);

for i=1:length(testsizes)
    n=testsizes(i);
    disp(['Testing size :' int2str(n)]);
    if (n<2)
        error('BRA must start with a collision');
    end
    for ii=1:c
        %generate the batch
        nodes=rand(n,1);
        nodes=[sort(nodes); 1];
        % CBT Estimate Simulation

        % true if we got a success in the last transmission
        lastwassuccess=false;
        %false to end CBT
        waitforconsecutive=true;

```

```
43          imax=length(nodes);  %index of the first node in the batch
44          imin=1;     % index of the first node in the batch
45          xmin=0;     % starting interval [0,1/2)
46          xlen=1/2;   % we suppose a collision already occurred.
47          l=1;        % current level in the tree
48
49          while (waitforconsecutive)
50              [e,imin,imax]=cbtsplit(nodes,imin,imax,xmin,xlen);
51              if(e==1)
52                  xmin=xmin+xlen;
53              elseif (e==0)
54                  xmin=xmin+xlen;
55                  xlen=xlen/2;
56                  l=l+1;
57              else
58                  xlen=xlen/2;
59                  l=l+1;
60              end
61              if (lastwassuccess==true && e==1)
62                  waitforconsecutive=false;
63              end
64              if(e==1)
65                  lastwassuccess=true;
66              else
67                  lastwassuccess=false;
68              end
69          end
70          ED(i,l)=ED(i,l)+1;
71      end
72  end
```

A.4 Greenberg Estimate Distribution

In following table A.2 we report how the end up probability (equation 4.12) is distributed among slots given a batch of size n. Column "n" lists the considered batch sizes. $\hat{n}$ is the resulting estimation (without corrections) when ending up in the underneath slot.
For sake of simplicity considered values are all powers of 2.
Datas presented were post-processed to become more accessible:

- values above 10^{-3} are reported in format ('*%1.3f*');
- values below 10^{-12} are not presented since are tight close to 0.
- other values are presented in exponential notation and rounded to the first meaningful digit ('*%1.e*')

	$\hat{n}$	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536
n	slot:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		1.000															
2		0.750	0.234	0.015	2e-04	1e-06	9e-10										
4		0.312	0.508	0.166	0.014	3e-04	2e-06	2e-09									
8		0.035	0.354	0.450	0.147	0.013	3e-04	2e-06	4e-09	1e-12							
16		3e-04	0.063	0.363	0.422	0.138	0.013	3e-04	2e-06	4e-09	2e-12						
32		8e-09	0.001	0.078	0.366	0.409	0.134	0.013	3e-04	2e-06	4e-09	2e-12					
64			2e-07	0.002	0.084	0.367	0.402	0.131	0.013	3e-04	2e-06	4e-09	2e-12				
128				7e-07	0.002	0.088	0.367	0.399	0.130	0.013	3e-04	2e-06	5e-09	2e-12			
256					1e-06	0.003	0.090	0.368	0.397	0.130	0.013	3e-04	2e-06	5e-09	2e-12		
512						2e-06	0.003	0.090	0.368	0.397	0.130	0.012	3e-04	2e-06	5e-09	2e-12	
1024							2e-06	0.003	0.091	0.368	0.396	0.129	0.012	3e-04	2e-06	5e-09	2e-12

	$\hat{n}$	128	256	512	1024	2048	4096	8192	16384	32768	65536	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
n	slot:	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
2048		2e-06	0.003	0.091	0.368	0.396	0.129	0.012	3e-04	2e-06	5e-09	2e-12					
4096			2e-06	0.003	0.091	0.368	0.396	0.129	0.012	3e-04	2e-06	5e-09	2e-12				
8192				2e-06	0.003	0.091	0.368	0.396	0.129	0.012	3e-04	2e-06	5e-09	2e-12			
16384					2e-06	0.003	0.091	0.368	0.396	0.129	0.012	3e-04	2e-06	5e-09	2e-12		
32768						2e-06	0.003	0.091	0.368	0.396	0.129	0.012	3e-04	2e-06	5e-09	2e-12	
65536							2e-06	0.003	0.091	0.368	0.396	0.129	0.012	3e-04	2e-06	5e-09	2e-12

Table A.2: Analytically computed basic Greeenberg Estimate Distribution

Bibliography

[1] Peter Popovski, Frank H.P. Fitzek, Ramjee Prasad, *A Class of Algorithms for Collision Resolution with Multiplicity Estimation*, Springer, Algorithmica, Vol. 49, No. 4, December 2007, 286-317

[2] Murali Kodialam, Thyaga Nandagopal, *Fast and Reliable Estimation Schemes in RFID Systems*, MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking, ACM , September 2006, 322-333

[3] Israel Cidon, Moshe Side, *Conflict Multiplicity Estimation and Batch Resolution Algorithms*, IEEE Transactions On Information Theory, Vol. 34, No. 1, January 1988, 101-110

[4] Albert G. Greenberg, Philippe Flajolet, Richard E. Ladner, *Estimating the Multiplicities of Conflicts to Speed Their Resolution in Multiple Access Channels*, Journal of the Association for Computing Machinery, Vol 34, No. 2, April 1987, 289-325

[5] J.I. Capetanakis, *Tree algorithms for packet broadcast channels*, IEEE Transactions On Information Theory, Vol. 25, No. 5, September 1979, 505-515