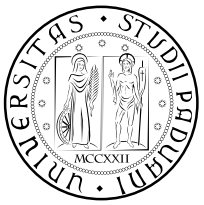

UNIVERSITY OF PADUA
DEPARTMENT OF INFORMATION ENGINEERING
MASTER DEGREE IN COMPUTER ENGINEERING

A.A. 2009/2010

BATCH SIZE ESTIMATE

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

Last Update: Tuesday 30th March, 2010 12:05

Contents

List of Figures

List of Tables

Chapter 1

Introduction

PARTI IN BLU AGGIUNTE O CORRETTE

NOTA, A FIANCO

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 case, resource access can be assimilated to a coordination problem.
In our scenario the contended resource is the physical transmission medium that is shared by several stations.
At the beginning of wired computer networks, multiple access control (MAC) was a big issue for efficient communications. The introduction of packet buffered switches in LANs reduced the conflicting set to only two stations (NOTA: *why??* RISPOSTA:perchè un cavo ha solo 2 estremità e nelle reti switched ogni porta appartiene ad un solo dominio di collisione (non è un hub ma uno switch)) simplifying the original problem. Switched networks, in fact, split large collision domains into smaller pieces thus realizing ring, star or mesh structures.
In a wireless context the problem can not be easily avoided, due to the broadcast nature of the wireless medium.

← NOTA, OK?

Nowadays wireless connectivity in pervasive computing has ephemeral character and can be used for creating ad-hoc networks, sensor networks, connections 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 broadcasts a query to the in-range nodes. Once the request is received, devices with data 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, while collision problems come in: only one successful transmission at time can be accomplished, concurrent transmissions result in destructive interference with inefficient waste of energy/time. This data traffic shows a bursty nature which is the worst case for all shared medium scenarios.

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

In our terminology a query determines a subset of nodes which have to reply with one (and only one) message: this set of nodes constitutes the *batch*. The size of the batch can be known in advance, in lucky and optimistic scenarios, or it can change in 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 interested in the exact number of nodes but rather we aim at an estimate of the actual batch size, as accurate as possible.

The knowledge of the batch size n is an important factor for parameters optimization in order to improve the resolution efficiency through the minimization of the time taken by the process.

1.1 Model

NOTA, A FIANCO

PARTI IN BLU AGGIUNTE O CORRETTE

We consider the following standard model of a multiple access channel. A large number of geographically distributed nodes communicate through a common radio channel. Any node generates a packet to be transmitted on the channel. Transmissions

start at integer multiples of time unit and last one unit of time called *slot*.

In *pure-slotted* systems some form of synchronization among nodes is required to inform the nodes about the beginning of slots (or at least the beginning of a cycle of slots). Nodes can start a transmission only at the beginning of the slot, otherwise they will stay quiet until the next slot to come.

In *carrier-sense multiple-access (CSMA)* networks each node is able to determine the beginning of a new slot by sensing the energy on the channel: when the channel is idle a device can start transmitting its message. In our scenario we assume that all the transmitted messages have a fixed length. Once a node has started transmitting it cannot sense the channel so that 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 results in a success or a collision. On the other hand, empty slots take less time than transmissions. Usually the duration of a transmission and idle slots are identified respectively by T_p and T_s. An important parameter in the CSMA channel is the *Carrier Sense Factor* $\beta = \frac{T_s}{T_p} < 1$.

We assume that there is no external source of interference and that a transmission can fail only when a collision takes place. In short, saying k nodes transmit simultaneously in a slot, we have what follows:

- If $k = 0$ then no transmission is 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 transmissions interfere destructively so that none succeeds. The slot is said to be *collided*.

Furthermore, throughout this work we assume that no new message is generated by the system or reaches it while it is running an *estimate* or *resolution algorithm*. In other words, newly generated packets are inhibited from being transmitted while an algorithm is in progress and they will eventually be considered only in the subsequent estimate or resolution process. This way to manage the information on the system is known as *obvious-access scheme*.

1.2 Goals

NUOVA SEZIONE

Batch Resolution Problem is implicitly present in many practical applications over wireless networks such as:

- *Neighbor Discovery.* After being deployed, nodes need to discover their one-hop neighbors. Regardless of the protocol used for message routing a node must absolutely inform its neighbors about its presence. Hence knowledge of one-hop neighbors is essential for almost all routing protocols, medium-access control protocols and several other topology-control algorithms such as construction of minimum spanning trees. Ideally, nodes should discover their neighbors as quickly as possible as rapid discovery of neighbors often translates into energy efficiency and it allows for other tasks to quickly start their execution on the system.

- *Batch Polling.* It consists in collecting a possibly very large number of messages from different devices in response to *time-driven* or *event-driven* constrains. *Time-driven* resolutions take place when an inquirer broadcasts a request to the nodes. *Event-driven* resolutions take place when the nodes are alarmed from their sensors that an environmental event of interest took place. The problem is not properly a *Batch Resolution Problem* when we are interested to obtain only 1 among n messages as rapidly as possible. This case was studied in [3].

- *Object identification*, where physical objects are bridged to virtual ones by attaching to each object a sensor or an RFID tag. This allows asset tracking (e.g. libraries, animals), automated inventory and stock-keeping, toll collecting, and similar tasks. Wireless connection allows unobtrusive management and monitoring of resources.

In these applications:

- communications show spatially and timely correlated contention

- in general, density of nodes is time-varying. When a node wakes up it has no knowledge of the environment around it. In particular this shows to be true when nodes sleep for most of time and seldom wake up to transmit.

BRAs can run oblivious of the batch multiplicity n: they would anyway solve the problem but expected time required would not be as short as possible. In fact, the knowledge of the conflict multiplicity n is the most critical factor to optimize the resolution and to allow the usage of advanced resolutions schemes which take advantage

of the knowledge of n. Thus the importance of *Batch Size Estimate* follows.

Hence, in this work, we will analyze different estimate techniques dealing with the quality and time taken by the estimate process.
Most of works in which estimate techniques are proposed (such as [5, 6]) define the estimate algorithm but do not provide data about the quality of the estimate or time taken by the process. In these works, in fact, after proposing an estimate scheme, the authors concentrate on the definition of an optimized resolution scheme considering a perfect knowledge of the batch size n and ignoring the fact that only and estimate of n can be provided without requiring the complete resolution of the batch.
In this work we will focus on the estimate phase preferring analytical analysis when possible and using computer based simulations when analytical analysis showed to be too complex or impractical.
Finally, we will try to propose improvements for some techniques in order to achieve better quality in the estimate and we will try to compare all the estimate algorithms to provide a comprehensive overview with pros and cons.

1.3 Document structure

NUOVA SEZIONE, NOTA A FIANCO

ANCORA PROVVISORIO
This master thesis is organized as follows:

- in Chapter 2 we will introduce the *Batch Resolution Problem* since it is the main motivation to further study the *Batch Size Estimate Problem*. We will deal about algorithms known in literature describing in details basic ones and providing an overview of the most recent and advanced ones. In particular we will concentrate on *binary tree algorithms*.

- Chapter 3 describes a few estimate techniques, using different approaches, in details. Pseudo code for the algorithms is provided and also mathematical analysis when possible.

- in Chapter 4 we will further analyze algorithms described in Chapter 3 to provide data for practical evaluation. We will also introduce a modified version of *Greenberg* algorithm to achieve better estimate quality.

- in Chapter 5 we will try to compare the different estimate algorithms and we will conclude our dissertation. Qui fondamentalmente vorrei confrontare greenberg

modificato con un approccio a finestra come Zanella o Lucent. (Finestra-> accuratezza, greenberg -> range opertativo) a parità di slot

Chapter 2

Batch Resolution

INIZIO COMPLETAMENTE RISCRITTO: spiega ALOHA e introduce il problema, chiarita in footnote la differenza tra probabilistici e deterministici NOTA, A FIANCO

Pure-ALOHA was the very first random-access protocol ever developed. It is trivially simple:

- If you have data to send, send the data immediatly
- If the message collides with another transmission, try resending "later"

Slotted-ALOHA is an improvement over Pure-ALOHA in which time is *slotted* and transmissions can start only at the beginning of a slot boundary. Slotted-ALOHA assumes there is feedback from the receiver at the end of each slot so that all nodes learn whether or not a collision occurred.
ALOHA protocol was studied under the assumption that a large number of identical sources transmit on the channel such that the number of new packets generated during any slot is a Poisson random variable with mean λ (packets/slot).
Slotted ALOHA has equilibrium rate (maximum throughput) of $1/e \approx 0.368$ packets/slot but it has proven maximum stable throughput 0 (hence it is unstable).
The attempt to obtain stable throughput random-access protocols brought to the discovery of CRAs.

A *Collision Resolution Algorithm* (CRA) can be defined as a random-access protocol such that, whenever a collision occurs, then at some later time all senders will simultaneously learn from the feedback information that all packets involved in that collision have now been successfully transmitted. The crux of collision resolution is the

exploitation of the feedback information to control the "random" retransmission process.

CRAs are interesting since their are able to solve conflicts of unknown multiplicities. Furthermore they are not tailored to solve only conflicts among packets arrivals characterized by Poisson's distributions but they are robust since they work for any arrival process characterized by an average arrival rate λ.

CRAs can also be used to solve collisions among a batch of nodes which have a message to deliver. In this case CRAs are commonly called *Batch Resolution Algorithms* (BRAs).

In particular, the scenario we consider is the following: the reader probes a set of nodes. In-range devices try to reply as soon as possible transmitting in the wireless medium. If two ore more devices reply at the same time we get a collision and the delivery of the messages fails. Consequently we require each node to run a distributed algorithm which implements anti-collision schemes in order to resolve all the nodes.
There are many algorithms that enable batch resolution, and these, according to [2], can be classified into two categories: (a) *probabilistic*, and (b) *deterministic* [1].

In *probabilistic algorithms*, a framed ALOHA[2] 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 tries to reduce the contending batch in the next slot based on the transmission result in the previous one.

[1]Both classes of algorithms use a probabilistic approach to solve the problem. As aforementioned, CRA were initially thought to solve Poisson-like packets arrivals. Hence, each packet is characterized by an arrival time. CRAs are able so solve collisions respecting the *first-come first-served* (FCFS) policy. In *Batch Resolution Problem* packets can be associated to virtual arrival times. Once the arrival time (alias node ID) is fixed each run of the same *deterministic* algorithm behaves in the same way and nodes always get resolved in the same order and time amount. On the other hand, *probabilistic* algorithms are not able to maintain the relative order among resolved nodes in different runs of the algorithms even if they globally solve the same initial collision.

[2] framed ALOHA is a modification of slotted ALOHA where consecutive slots are grouped. Each node is allowed to choose only one slot per group. Its transmission is allowed to take place only in the choosed slot.

These algorithms fall into the class of tree-based algorithms with the nodes classified on a binary tree 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 subsets of contending nodes 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. Consequently 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 they will continue to transmit in every resolution cycle. This lengthens the total time needed for the resolution process to complete. Wireless sensors, semi-active and active tags should allow to implement tree-based algorithms: the reader can acknowledge nodes that have succeeded at the end of each slot (immediate feedback), 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 transmits its message is said *resolved* and stays silent until the end of the algorithm.

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. Moreover, reader feedback in each slot adds overhead to the resolution. ←

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 wake up in the slot it has decided to transmit on. 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. NUOVA

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

2.1 Binary Tree Algorithms

INTRO
TUTTA
RIFATTA

In ‘70s, concern over the instability of most ALOHA-like protocols led some researchers to look for random-access schemes that were provably stable. The breakthrough in these efforts was reached in 1977 by J. Capetanakis [10], then a MIT doctoral student working

with Prof. R. Gallager[3], and independently achieved shortly thereafter by two Soviet researchers, B. Tsybakov and V. Mihhailov [9]. Basic Binary Tree is the result of their studies.
In the following sections we will describe and analyze the *Basic Binary Tree Algorithm* and some of its variants applied to the *Batch Resolution Problem*.

We notice that *tree algorthims require for the nodes in the batch to be unique*. This is a necessity since otherwise it would be impossible to force two nodes to behave in a different way. In this Chapter we will refer to this node uniqueness property as *id*, *address*, *token*. Throughout the following sections, we will consider *ids*, *addresses* and *tokens* equivalent.

We also assume to operate in a *slotted*-ALOHA-like scenario, unless otherwise specified.

2.1.1 Basic Binary Tree

Let's consider a batch $\mathcal{B}$ of size n.
Initially all the nodes try to transmit and we can have, according to what expressed in Section 1.1, three different events: *idle*, *success*, *collision*.AL POSTO DI SPIEGARE HO MESSO UN RIFERIMENTO ALLA SEZIONE DOVE VIENE DEFINITO IL MODELLO NOTA, OK?
The supervisor broadcasts the result of the transmission to all the nodes.
If we get *idle* or *success* events, the resolution process stops meaning respectively that there were no nodes to resolve or there was only one node that was successfully resolved. That node delivered its message and will no longer take part in the current batch resolution.
In case of *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 run the same algorithm.
Each node chooses to transmit with probability p and not to transmit with probability $1-p$. Nodes that transmit are said to belong to set $\mathcal{R}$ while the others to set $\mathcal{S}$. Of course $\mathcal{R} \cap \mathcal{S} = \emptyset$ and $\mathcal{B} = \mathcal{R} \cup \mathcal{S}$. The two sets are then resolved recursively starting from $\mathcal{R}$.
Nodes in $\mathcal{S}$ wait until all terminals in $\mathcal{R}$ transmit successfully their packets, then they

[3]FCFS splitting algorithm for Poisson's arrivals is one of his famous contributions.

transmit.

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

```
// current slot status can be idle, success, collision
Input: B batch with |B| = n
each node transmits its message
if (idle or success) then
    return
else
    each node flips a fair coin
    R ← { nodes that flipped head}
    S ← { nodes that flipped tail}
    BINARY TREE (R)
    BINARY TREE (S)
end if
```

Pseudo-code in Alg. 1 provides a very high level description of the splitting technique. The role of the supervisor is implicit in the code.

NOTA, OK?

Intuitively setting $p = 1/2$ can be a good choice since the algorithm is in some sense "symmetric": best performance are achived when $\mathcal{R}$ and $\mathcal{S}$ are balanced. ← "EXPLAIN BETTER"

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

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

with

$$L_0^{BT} = L_1^{BT} = 1.$$

To obtain an upper bound on the expected time as $n \to \infty$ further analysis techniques have to be used. NUOVO, INTRO

Here we want simply focus on how the algorithm behaves when n grows. DUCE

E

The behaviour of L_n^{BT} is presented in Figure 2.1 and was obtained evaluating equation (2.1) for $n = 0, 1, \ldots, 19$. L_n^{BT} grows almost linearly after the initial step from 1 to 2 which dramatically impacts on the performance ($L_2^{BT} = 5$). SPIEGA IL GRAFICO

Figure 2.1 also reports the results for MBT Alg. that will be introduced in following Section 2.1.2. We chose to anticipate the results for MBT to avoid inserting too many figures and, at the same time, provide a useful performance comparison between BT and MBT.

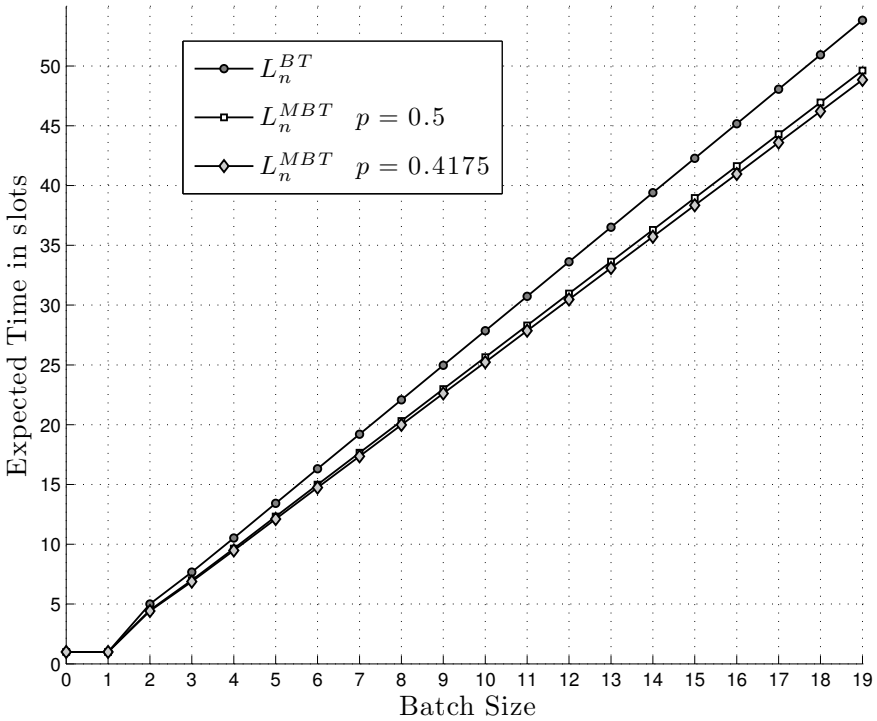

Figure 2.1: The plot illustrates the expected cost in slots to solve batches of size $n = 0, 1, \ldots, 19$ in a *slotted* Aloha-like scenario using all the basic variants of the tree based algorithms: BT, MBT with sub-optimal $p = 0.5$, MBT with optimal $p = 0.4175$. All the algorithms show the same behavior: almost linear grow for large n. Best performance are provided by MBT with optimal p.

Considering the efficiency $\eta_n = n/L_n^{BT}$ (number of resolved nodes over slots) we have

a decreasing series $\eta_1 = 1$, $\eta_2 = 0.40$, $\eta_3 = 0.3913$, ..., $\eta_{16} = 0.3542$, ..., $\eta_{31} = 0.3505$. It can be shown [8] that $\eta_\infty \approx 0.347$.

Since the algorithm is much more efficient in solving small rather than large batches we would prefer to have (ideally) n batches of size 1 rather than 1 batch of size n.
Hence, knowing exactly the cardinality n of the initial batch $\mathcal{B}$, we can split the nodes into small groups, of approximately 1 node each, and resolve them faster.
This is the idea behind many improvements over the basic BT and it reveals the importance of having an accurate estimate of n to efficiently solve a batch.

Example

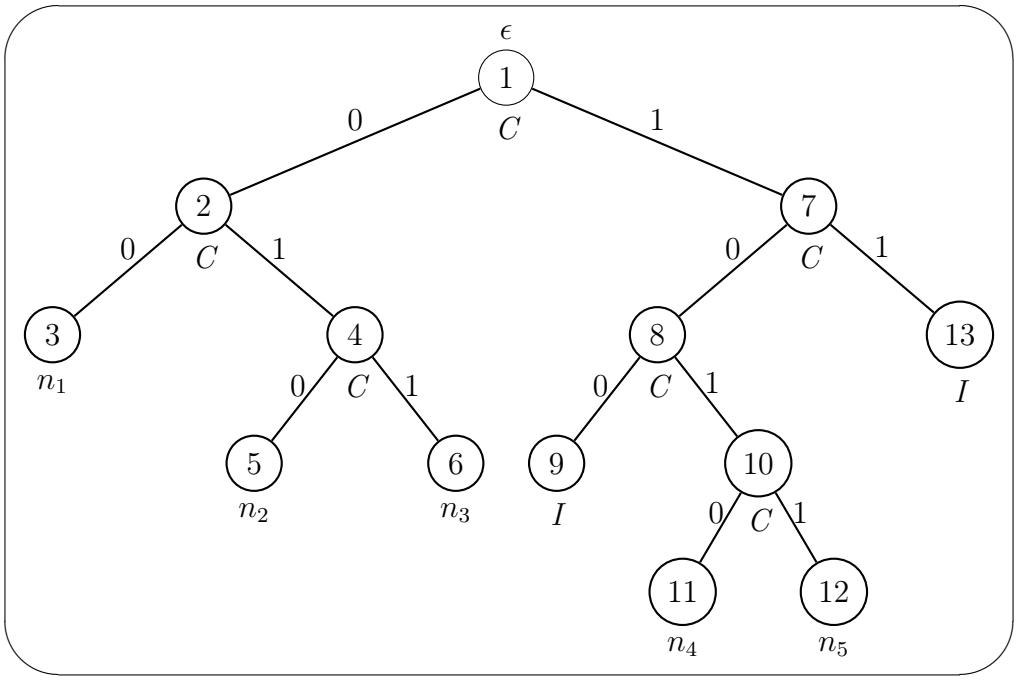

Figure 2.2: An istance of BT algorithm for $n = 5$ nodes. The number inside each circle identifies the slot number. The label below identifies the event occurring: I for *idle*, C for *collision*, n_i for resolution of node i. 0/1 branches is analogous to head/tail.

.

In Figure 2.2 we provide an example to further investigate the behavior of the algorithm. We notice that the instance starts with a collision in slot 1. Then nodes n_1, n_2, n_3 decide to proceed with a retransmission while n_4, n_5 remain idle. In slot 2 we see another collision, after it n_1 transmits again while n_2 and n_3 to stay quiet. In slot 3 we have the first resolution, n_1 successfully send its message and leaves the collision resolution algorithm.

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 *id* interpretation

TUTTO

BRAs require each node to have a *unique id* to solve the batch. Usually the nodes *id*s are random generated at each algorithm run and can be used to identify a node inside the algorithm. There are multiple ways to generate that the *id*, such as: RIFATTO

- flipping a coin on demand after each collision (step-by-step *id* generation),
- generating a 'long enough' random binary string at the beginning of the algorithm.

We do not want to enter in the details of these choices since there is no reason to prefer one method to the others but device technical limitations.
Now we want to introduce an interesting interpretation of the *id* that will be used later in algorithms such as EBT (Section 2.1.4) and Cidon (Section 3.2).

In general any infinite length binary 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 defined as follows:

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

Each node n_i can be associated to a point r_i within the real interval [0,1) as well as to the string $\mathbf{b_i}$.
For a finite length bit-string $\mathbf{a} = (a_1 a_2 \ldots a_L)$ with length $L = l(\mathbf{a})$ we adapt the definition of r as follows:

$$r(\mathbf{a}) = \sum_{j=1}^{L} \frac{a_j}{2^j} \tag{2.3}$$

In this case we have to carry L as auxiliary information to allow the map to remain bijective.
Following standard conventions, the empty string ϵ is prefix of any other string. It has length 0 and $r(\epsilon) = 0$.

Tree traversal rules

TUTTO

The duality in the interpretation of nodes' *id*s (bit-strings or real numbers) reflects on the duality of the tree traversal rules. RIFATTO
According to the adopted approach, enabled nodes can be specified by:

- a finite length string $\mathbf{a}$ which matches the path from the root to the first node in the sub-tree they belong to. In this case $\mathbf{a}$ is a prefix of the enabled nodes *id*s.
- the couple $\big(r(\mathbf{a}), l(\mathbf{a})\big)$ which enables the sub-interval $r(\mathbf{a}) \leq x < r(\mathbf{a}) + \frac{1}{2^{l(\mathbf{a})}}$

To complete the overview of the algorithm we now intuitively describe the tree visit. The following description uses the bit-string approach since a binary string can be immediately mapped to a path in the tree starting from the root. We assume, following the standard approach, to visit the tree in pre-order, giving precedence to left sub-trees, conventionally associated to 0 branches.
The visit starts from the root which has address ϵ.
Let $\mathbf{a}$ be the current enabled string, then the following rules apply:

- If $\mathbf{a} = \epsilon$ and last the event is a success or an idle slot then the whole conflict has been resolved;
- If the last event is a collision then we visit the left child of the current node ($\mathbf{a}0$);
- If the last event is a success or a collision and $\mathbf{a}_L = 0$ then we visit the right sibling of the current enabled node ($\mathbf{a}1$);
- If the last event is a success or a idle slot and $\mathbf{a}_L = 1$ then we look in the path back to the root for the first node whose sibling has not yet been visited. Since we visit the tree in pre-order the next enabled string will be in the form $\mathbf{a}_k 1$ with $k < L$.

A detailed pseudo-code of an algorithm that implements the rules above (and more) can be found in [1]. The *Modified Binary Tree* algorithm presented in next section gets a remarkable performance improvement over BT by adding just one new rule.

2.1.2 Modified Binary Tree

Modified binary tree is a simple way to improve the BT algorithm.
To keep the notation simple, we will explain the idea illustrating what happens the first time it is applied. In this case node τ is visited in slot τ. This does not holds in general, but explanation would have required to use two different indexed for slots and nodes, and Figure 2.3 would have been less immediate to understand.

The observation is that, during the tree traversal, sometimes we know in advance if the next slot will be collided. This happens when, after a collided slot τ, we get an idle

slot $(\tau + 1)$ in the left branch of the binary tree. In this case, visiting the right branch $(\tau + 2)$, we will certainly get a collision .
In fact, after sensing slot τ is collided, we know that there are at least 2 nodes in the last visited sub-tree. None of them belongs to the left-branch of that sub-tree since slot $(\tau+1)$ is idle. Consequently they must be in the right branch of the sub-tree, whose enabling will hence result into a collision. This collision can be avoided by skipping node $(\tau + 2)$ and visiting its left-child node in slot $(\tau + 2)$.

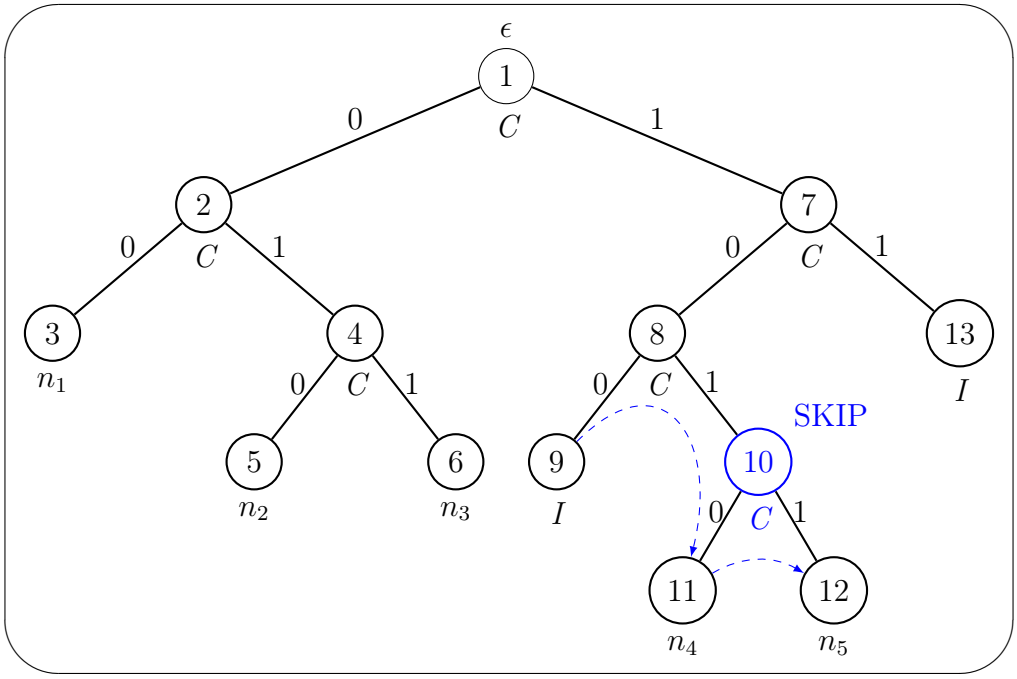

Figure 2.3: Same example as in Figure 2.2 but using MBT: tree structure do not change but node 10 is skipped in the traversal.

Expected time analysis is analogous to Section 2.1.1. The only difference is that after a collision, if we get an idle slot, we will skip the "next one" (saving a slot that would certainly be wasted otherwise). Consequently the expected slot cost is $\big[1 \cdot \big(1 - Q_0(n)\big) + 0 \cdot Q_0(n)\big]$. Let L_n^{MBT} be the expected cost in slots to solve a batch of size n using the MBT Alg., then

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

with

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

Intuitively in this case, since a higher probability to stay silent reduces the expected slot cost, optimal transmit probability will be no longer equal to one half. At the same time, reducing the transmit probability will increase the number of (wasted) idle slots. Thus the new optimal probability p will be somewhere in the interval (0,0.5).
It can be shown [7] that the best result is achieved for $p = 0.4175$, for which the efficiency is asymptotically equal to $\eta \approx 0.381$. This is +10% higher than basic BT.
In general we have

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

Using probability $p = \frac{1}{2}$ results, for large n, in about 1.6% peak performance loss ($C = 2.667$), which is a moderate decrease. It is important to notice that $p = 0.4175$ is close to the optimal bias for small n as well.

2.1.3 m Groups Tree Resolution

NUOVA

More advanced algorithms for batch resolution, such as those presented in [5] and [6], share a common idea: divide the initial batch into m groups. We will not deal here about the details of the algorithms and the different approaches they use to choose m. Instead we will concentrate on the common part of these algorithms: divide a batch of size n into m groups and apply a BRA to each group.
Given m groups, the probability to have exactly i among n nodes in a group is given by

$$\binom{n}{i} \left(\frac{1}{m}\right)^i \left(1 - \frac{1}{m}\right)^{n-i}. \tag{2.6}$$

Let L_k be the expected cost to solve a batch of size k with a chosen BRA. Then the expected cost to solve a batch of size n dividing it into m groups is given by

$$L'_n(\rho) = m \sum_{i=0}^{n} L_i \binom{n}{i} \left(\frac{1}{m}\right)^i \left(1 - \frac{1}{m}\right)^{n-i}, \tag{2.7}$$

where $\rho = \frac{n}{m}$ is the expected number of nodes in a group.

Figure 2.4 shows the results obtained computing (2.7) for some batch sizes n varying ρ.

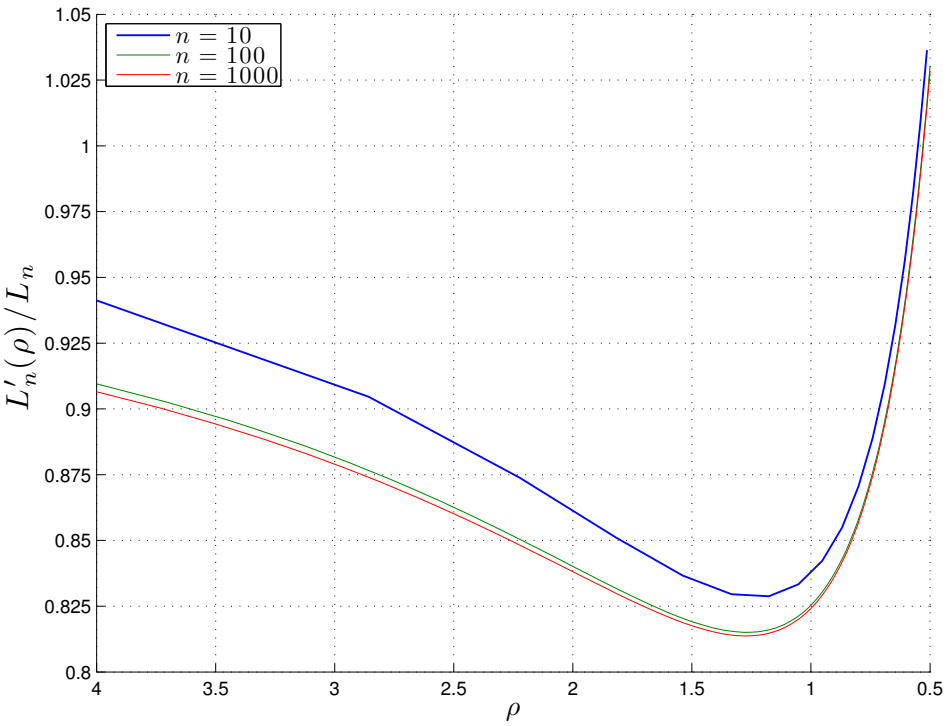

Figure 2.4: *m groups split: ALOHA scenario.* Results were obtained using MBT with optimal probability p in a slotted ALOHA scenario. Splitting the batch into groups provides better results than applying a BRA to the original batch size when y-axis values are below 1.

Figure 2.4 shows that $\rho \approx 1.26$ provides the best achievable performance. Furthermore we notice how the knowledge of the batch size is critical: m can be set to the optimal value only knowing exactly the batch size n but often we only have an estimate of n. If the estimate is smaller than n, performances smoothly degrade but still remain always better than the trivial application of the BRA. On the other hand overestimating n can lead to performance loss when $\rho \approx 0.5$ or smaller.

In a CSMA scenario the performances depends on β and on the ratio between the nodes packet size (S_n) and the supervisor feedback size (S_f). Figure 2.5 was obtained considering $\frac{S_n}{S_f} \approx 4$. We notice that in CSMA increasing idle slots probability (ρ small) can lower the expected cost. CSMA is also much more robust to an overestimate of n than *slotted*-ALOHA.

RICHIEDO VERIFICA PER CMSA... ←

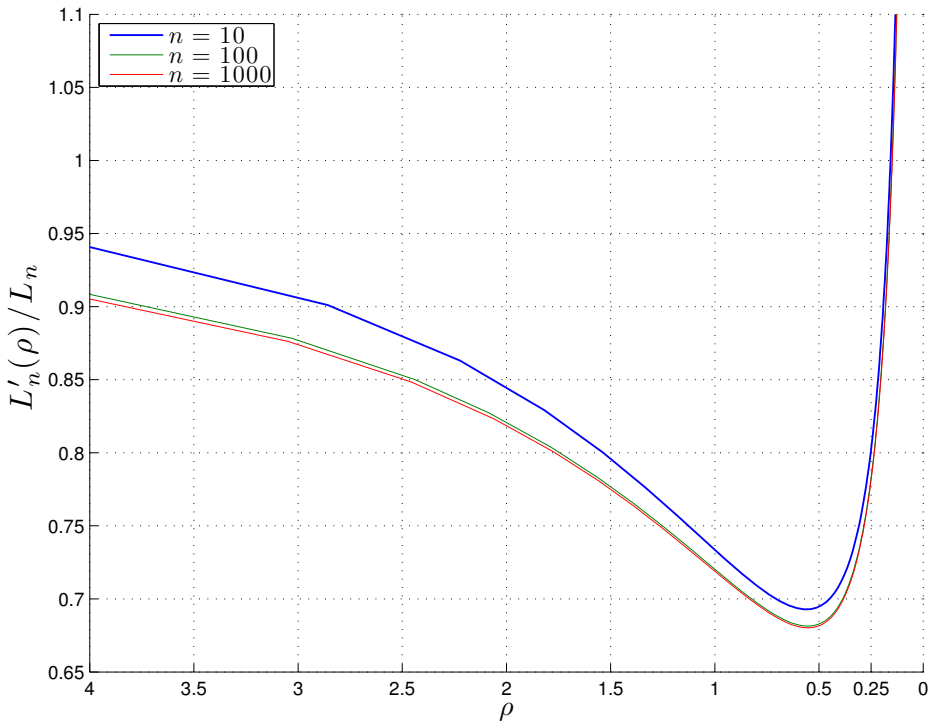

Figure 2.5: *m groups split: ALOHA scenario.* Results were obtained using MBT with optimal probability p in a slotted CSMA scenario. CSMA is more robust to inaccurate estimate than ALOHA.

2.1.4 Estimating Binary Tree

NUOVA

Estimating Binary Tree (EBT) has been recently proposed in [1]. It does not work on parameters optimization but tries to use simple heuristics to skip tree nodes that will result in collisions with high probability.
Given a batch of size n, the keys to understand EBT are the following:

- in the luckiest scenario running BT results in a balanced binary tree where all the leaves are at the same level,
- consequently it seems to be a good idea to assume that nodes at levels in the tree less than $\lfloor \log_2 n \rfloor$ will likely result in collided slots.

EBT tries to skip inner nodes of the tree by visiting only nodes at levels equal or deeper than $\lfloor \log_2 n \rfloor$. Since n is not *apriori* known, a dynamic estimating technique is adopted. To effectively use this estimate tecnique each node must be able to generate values from the standard uniform distribution on the interval [0,1) and to use that value as its unique *id*.

Assume that there are k nodes whose *id*s are in the sub-interval [0,p_ϵ) and that they have been previously resolved: all and only the nodes with *id* less than p_ϵ successfully transmitted their messages. Let $\hat{n}$ express the estimate of n. When p_ϵ is greater than 0, setting $\hat{n}$ to $\frac{k}{p_\epsilon}$ provides a good estimate of n that becomes more and more accurate as the algorithm goes on. EBT uses $\hat{n}$ (as soon as it is available) to choose the right level in the tree to analyze.

The most straightforward interpretation of the EBT algorithm can be:

1. Run the CBT algorithm.
2. Repeat until the end of the process the following steps:
   a) Update the estimate $\hat{n}$,
   b) Start a new CBT at the next node from level $\lfloor \log_2 \hat{n} \rfloor$.

Here we gave only a short insight about the algorithm and the estimate technique. In particular this estimate technique is a dynamic adaptation of that described later in Section 3.2 and originally proposed in [5].

2.1.5 EBT combined with optimum MBT (DA BUTTARE)

In this section we describe an improvement over EBT we developed.
EBT assumes the nodes to be uniformly distributed in the interval [0,1): this means that their *id* is generated flipping a fair coin. This is a necessary condition for the collision tree to be balanced (the expected height of each leaf is the same) and allows to estimate the level in the tree to analyze using $\lfloor \log_2 \hat{n} \rfloor$.
We remember also that the optimal probability p that provides the best performance for the MBT is $p = 0.4175$. Nevertheless, when p is optimum, the recursive splitting of the original set into 2 subsets, whose expected size is np and $n(1-p)$, returns an unbalanced collision tree.

The ideas are the following:

- Initially nodes *id*s are uniformly distributed in the tree,
- Changing a part of a node *id* is allowed if this doesn't affects the already visited conflicting sets,

- After a collision takes place in a node at level $\hat{N} = \lfloor \log_2 \hat{n} \rfloor$, the devices involved in the collision can be redistributed in the collided domain.

Let each node n_i to be identified by a L-length binary string $\mathbf{b_i} = (b_{i1} b_{i2} \dots b_{iL})$. Applying the MBT we resolve a sub-batch or, otherwise, get a collision. In case of collision each node involved updates its *id* as follows: $\mathbf{b_i} = (b_{i1} b_{i2} \dots b_{i\hat{N}} c_{i1} \dots c_{i(L-\hat{N})})$ where $c_{ik} = 0$ with probability $p = 0.4175$ and $c_{ik} = 1$ with probability $1 - p$.

This modified version of EBT can be briefly described as follows:

1. Run the CBT algorithm;
2. Repeat until the end of the process the following steps:
   a) Update the estimate $\hat{n}$;
   b) Start a new MBT at the next node at level $\lfloor \log_2 \hat{n} \rfloor$ and perform only the first transmission;
   c) If the transmission results in a collision, colliding nodes regenerate their *id*;
   d) Continue with the current MBT.

Figure 2.6 shows the expected improvement in percentage between the EBT and the modified EBT algorithm in terms of expected cost ratio. Performance gain results oscillatory with mean 1.42.

The *EBT using optimum MBT Algorithm*, proposed in this work, is the average case best *tree-based algorithm with implicit estimate* known so far.

2.2 Others

In this section we will shortly describe non *tree-based* algorithms for batch resolution.

2.2.1 IERC

NUOVA

The *Interval Estimation Conflict Resolution (IERC)* [1] is an adaptation of the FCFS [4] algorithm for Poisson's arrivals of packets to the batch resolution.
FCFS is the fastest known algorithm to solve Poisson's arrivals and it achieves efficiency $\eta = 0.4871$. It assumes *apriori* knowledge of the arrival rate λ of the packets and works by splitting the time in epochs of length τ.

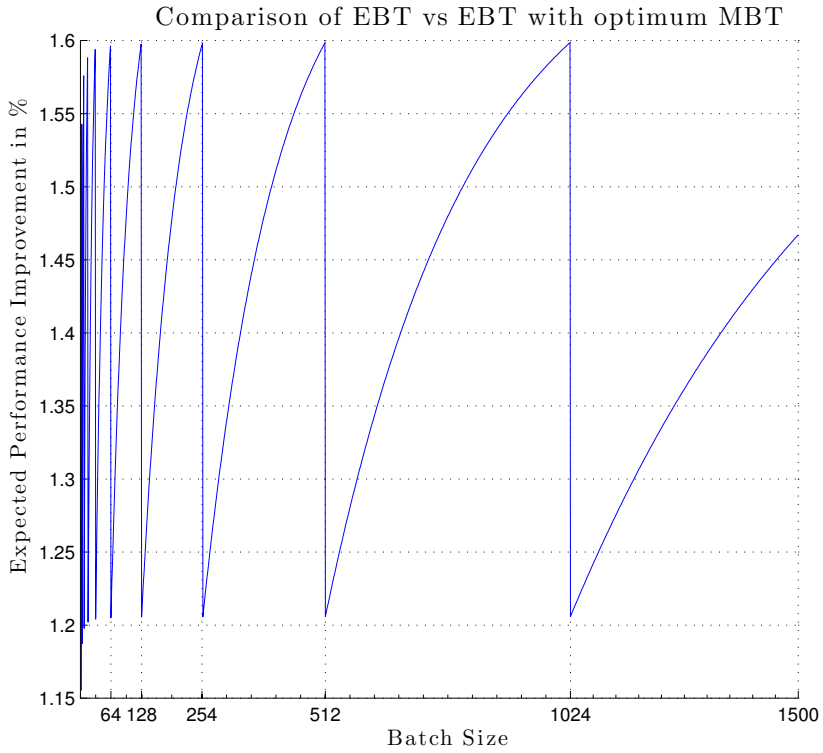

Figure 2.6: EBT with optimal MBT provides a minimum expected performance improvement over EBT of 1.2% and 1.42% in mean.

Now we briefly describe FCFS Alg. and then we will show how it can be translated to the batch resolution problem.

At any time k, let $T(k)$ be the start time of the current enabled allocation interval. By default, packets generated in the enabled window $(T(k), T(k) + \tau]$ are allowed to be transmitted. When an *idle* or *successful* event occurs enabled time window gets incremented (shifted towards current time) by τ. On the other hand, if a *collision* takes place, a *Clipped Binary Tree* (see Section 3.1) Alg. is started. Assume that CBT resolved the sub-interval $(T(k), T(k) + \alpha(k)]$, then at time k' next FCFS enabled window will be $(T(k'), T(k') + \min(\tau, k' - T(k'))]$ where $T(k') = T(k) + \alpha(k)$.

It can be shown [4] that setting $\tau = \dfrac{1.26}{\lambda}$ leads to the maximum efficiency. This means that, on average, there are 1.26 nodes in an allocation interval of length τ.
The batch resolution problem can be adapted to be solve with an FCFS-like strategy in the following way:

- nodes *tokens* can be interpreted as arrival times;

- nodes must be splitted into $m = \frac{n}{1.26}$ groups so that in each group we expect to have 1.26 nodes;
- behave FCFS-like to solve the problem.

Following these ideas, IERC achieves the same efficiency $\eta = 0.4871$ of FCFS to solve the *Batch Resolution Problem*.

2.2.2 Window Based Approaches

solo per dire che esistono, magari qualche riferimento bibliografico...

NUOVA,
RICHIEDO
VERIFICA
DEL
CONTENU

Another possible approach to the batch resolution is to consider a *framed*-ALOHA-like scenarios.
In *framed*-ALOHA, a frame (*alias* window) is a sequence of consecutive slots. When a node in the collision batch decides to transmit, it uniformly picks one and only one slot in the window and transmits in that slot.
In windows based scenario usually we can work on the optimization of two parameters: m, the length of the window in slots, and p, the probability that a node transmits in the current window. Innovate approaches to the problem uses hybrid transmission scheme working by cycles that tries to optimize operational parameters after each transmission windows. On very inexpensive devices usually only windows based algorithms can be implemented.

Chapter 3

Batch Size Estimate Techniques

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

In general, we assume to have no *a priori* statistical knowledge about the conflict multiplicity. Estimation techniques are thus required to provide accurate estimates for the general zero-knowledge scenario.

3.1 Clipped Binary Tree

A simple way to obtain an estimate of the batch size is to solve a minimum number of nodes. This can be done, for example, by using deterministic algorithms such as *Clipped Binary Tree Algorithm* (CBT).
CBT is a partial resolution algorithm since only a fraction of the packets of the batch are successfully transmitted. The algorithm is identical to the MBT (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 resolved nodes belong to the same level i of the tree (root is assumed to be at level 0).
Therefore, an estimate of the initial batch size is given by:

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

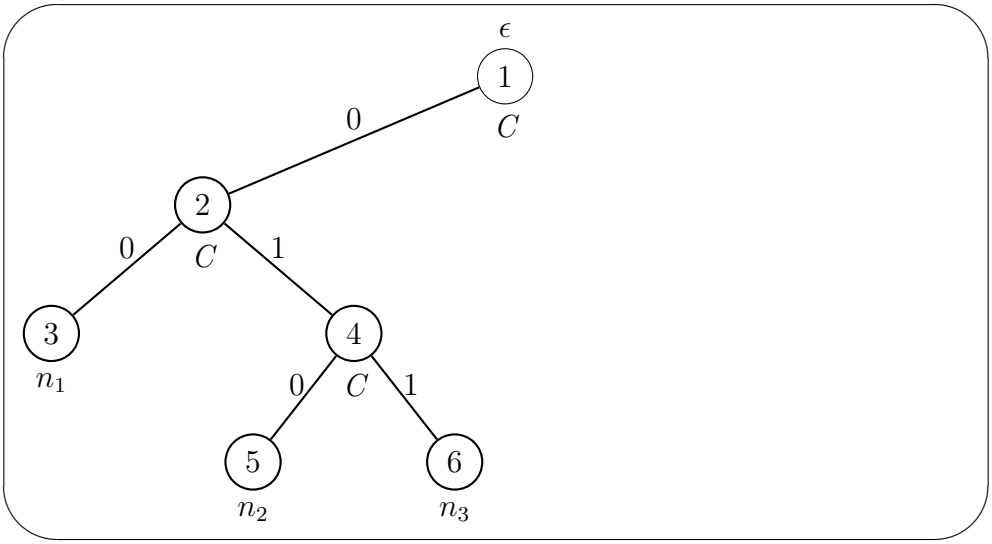

Figure 3.1: Same example as in Figure 2.2 but resolution using CBT ends up after two consecutive successful transmissions.

Experimental results show that the variance of this 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 at the most left in the tree). Consequently estimate is quite inacurate. 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 [5]. In their work they describe a complete resolution algorithm based on two phases:

1. Estimating the initial batch size using a partial deterministic resolution scheme.
2. Performing an optimized complete deterministic resolution based on the results of phase 1.

The strategy adopted to obtain the estimate is to resolve a small part of the initial batch and to accumulate the number of successful transmissions resulting.

A node either takes part to the estimate phase or to the following resolution phase but not both. The probability p_ϵ, which is an algorithm input parameter, determines

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

As usual we consider 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. When a conflict occurs ($|\mathcal{E}| \geq 2$), a complete batch resolution algorithm is started to resolve the conflicts among the nodes in $\mathcal{E}$. By simply accumulating in j, during the estimate phase, the number of resolved nodes, we know the exact value of $|\mathcal{E}|$. Then we simply compute our estimate $\hat{n}$ as:

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

When the nodes are uniformly distributed in the real interval [0,1), $\frac{j}{p_\epsilon}$ identifies also the expect nodes density in the interval [0, 1) and the nodes in $\mathcal{E}$ are those whose id can be mapped to a value belonging to the sub-interval [0, p_ϵ).
Following Figure 3.2 shows the case providing a simple example.

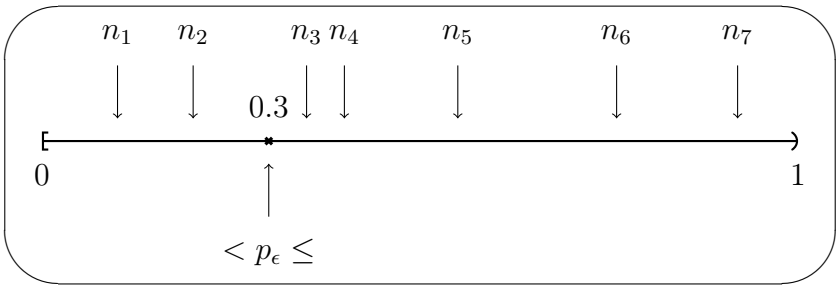

Figure 3.2: In this example $p_\epsilon = 0.3$. At the beginning of the algorithm each node generates its own id from the standard uniform distribution on the interval [0,1). Nodes whose id is less than p_ϵ owns to $\mathcal{E}$. Nodes whose id is greater or equal to p_ϵ owns to $\mathcal{D}$. Estimate of the batch returns $\lceil 2/0.3 \rceil = 7$ which, in this case, is the exact size of the batch.

After the first phase, the nodes in $\mathcal{E}$ are resolved, but those in $\mathcal{D}$ not yet. To speedup their resolution we are interested in the estimate $\hat{k}$ of $|\mathcal{D}|$, which we can compute as:

$$\hat{k} \leftarrow \frac{j}{p_\epsilon}(1 - p_\epsilon). \tag{3.3}$$

Then the resolution of $\mathcal{D}$ begins. We note that $|\mathcal{E}| = 0$ does not imply $|\mathcal{D}| = 0$ and consequently a complete resolution algorithm has always to be performed on $\mathcal{D}$.

Algorithm 2 CIDON($\mathcal{B}$, p_ϵ)

Input: $\mathcal{B}$ batch with $|\mathcal{B}| = n$

Input: p_ϵ, fraction of the whole batch to solve

1: // Phase 1
2: each node flips 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)

High level peudo-code of *Cidon* algorithm is presented in Alg. 2.
COMPLETE COLLISION RESOLUTION ($\mathcal{E}$) identifies any procedure able to resolve all the nodes in $\mathcal{E}$ allowing them to successfully transmit their messages. OPTIMIZED COMPLETE COLLISION RESOLUTION ($\mathcal{D}$,$\hat{k}$, p_ϵ) identifies an optimized way to resolve the batch $\mathcal{D}$: the speedup is allowed by the knowledge of its expected multiplicity.

The original paper proposes to use an m *groups split* (Section 2.1.3) approach to resolve $\mathcal{D}$ where

$$m = \max(1, \lceil \alpha\hat{k} - \beta \rceil), \tag{3.4}$$

and each group is resolved by applying the MBT algorithm.
The parameter α determines the number of groups, and therefore the efficiency, when n is large, while β reduces the number of groups when n is small: many of them would be empty resulting in a waste of time.

$\alpha = 0.786$ determines $\rho \approx 1.27$ which is the *unique* optimum nodes per group density. β and p_ϵ depends on operational requirements: setting $\beta = 8$ and $p_\epsilon = 0.1$ showed to be a good compromise to get efficient resolution for a wide range of batch sizes.

Expected cost of the estimate phase depends on the BRA used but in general can be considered $O(p_\epsilon n)$: time is linear in the size of $\mathcal{E}$.

3.2.1 Estimate accuracy

In the original paper [5] there is no detailed analysis of the behavior of the estimate algorithm but it is only shown the following fact: as n grows the estimator becomes more accurate.

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:

$$P(J = j|n) = \binom{n}{j} p_\epsilon^j (1 - p_\epsilon)^{n-j}, \tag{3.5}$$

$$E[J|n] = np_\epsilon, \tag{3.6}$$

$$\text{var}(J|n) = np_\epsilon(1 - p_\epsilon). \tag{3.7}$$

By applying Chebychev's Inequality (A.1), we have for any $\epsilon > 0$

$$P\Big(|J - np_\epsilon| \geq \epsilon n \,|\, n\Big) \leq \frac{p_\epsilon(1 - p_\epsilon)}{\epsilon^2 n}. \tag{3.8}$$

Let $\hat{N} = \dfrac{\hat{J}}{p_\epsilon}$ be a real-valued random variable that expresses our estimate of n. Then from the aforementioned equations (3.5) - (3.8) follows that:

$$P\left(\hat{N} = n|n\right) = \binom{n}{j} p_\epsilon^j (1 - p_\epsilon)^{n-j} \quad \text{with} \quad \hat{n} = j/p_\epsilon, \quad 0 \leq j \leq n, \tag{3.9}$$

$$E[\hat{N}|n] = n, \tag{3.10}$$

and, finally,

$$P\left(\left|\frac{\hat{N}}{n} - 1\right| \geq \epsilon \,\Big|\, n\right) \leq \frac{1 - p_\epsilon}{\epsilon^2 n p_\epsilon}. \tag{3.11}$$

which shows that in this estimation method we can trade off the accuracy with the consumption of resources, time or messages.

3.3 Greenberg

Basic Greenberg algorithm strategy is to search for *a power of 2* that is close to n with high probability.
The probabilistic test is defined to look for $\hat{n}$ which tries to satisfy:

$$\hat{n} \geq 2^i \approx n \tag{3.12}$$

Let each of the n conflicting stations either transmit or not transmit in accordance with whether the outcome of a biased binary coin is 0 or 1. The coin is biased to turn up 0 with probability 2^{-i} and 1 with complementary probability. Since the expected number of transmitters is $2^{-i}n$, having a conflict as event supports the hypothesis that $n \geq 2^i$.
Using this test repeatedly with $i = 1, 2, 3, \ldots$, leads to the Greenberg *base 2 estimation algorithm.*
Each of the conflicting stations executes Algorithm (3), resulting in a string of collisions whose lenght determines $\hat{n}$.

The probability that at most one node transmits monotonically grows and approaches 1 extremely rapidly as i increases past $\log_2 n$. Consequently, we expect i is close to $\log_2 n$.

Algorithm 3 BASIC GREENBERG ($\mathcal{B}$)

```
// Each node performs these operations
i ← 0
repeat
    i ← i + 1
    choose to transmit with probability 2^{-i}
until no collision occurs
n̂ ← 2^i
```

The idea behind algorithm 3 appears to be quite simple: as the algorithm goes on the initial unknown batch (of size n) is progressively sliced into smaller pieces. Only the nodes virtually inside the enabled slice are allowed to transmit. Slices get thinner and thinner until at most one node is contained in a slice. Figure 3.3 intuitively tries to explain the idea.

Expected running time is $O(log_2 n)$. In particular, since in the *slotted*-ALOHA model considered in the paper reader feedback is supposed to be transmitted at the end of each

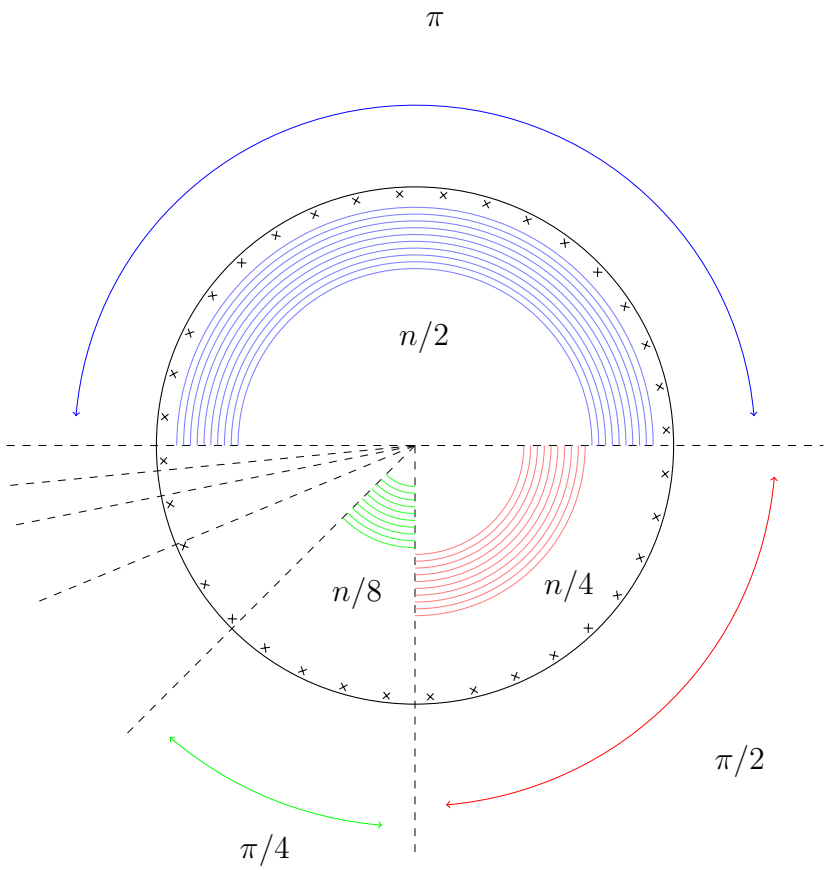

Figure 3.3: Visually nodes can be thought to be uniformly distributed on the circumference of a circle. By performing Greenberg's algorithm we go and analyze each time a smaller sector (in this case the half of the previous one) of the circle and find when a sector contains only 1 or no nodes. Not overlapping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

transmission slot, expected running time can be expressed, in slot numbers, as $\approx 1+\log_2 n$.

An important note is that the algorithm always involves 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 node did in the previous steps.
This is of great importance and allows $\hat{n}$ to have bounded moments: it can be shown for large n that:

$$E[\hat{n}] \approx n\phi, \tag{3.13}$$

$$E[\hat{n}^2] \approx n\Phi, \tag{3.14}$$

where

$$\phi = \frac{1}{\log 2} \int_0^\infty e^{-x}(1+x) \prod_{k=1}^{\infty} \left(1 - e^{-2^k x}(1+2^k x)\right) x^{-2}\, dx = 0.91422\ldots \tag{3.15}$$

$$\Phi = \frac{1}{\log 2} \int_0^\infty e^{-x}(1+x) \prod_{k=1}^{\infty} \left(1 - e^{-2^k x}(1+2^k x)\right) x^{-3}\, dx = 1.23278\ldots \tag{3.16}$$

ϕ and Φ where obtained in [6] using advanced mathematical analysis supported by Mellin integral transform[1]. In general ϕ and Φ depend on the size of the problem. Following Table 3.1 shows the behavior of the expected estimate (and therefore ϕ) as function of n.

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

[1] in this work we only report these as final results, please refer to the original paper to see how ϕ and Φ where obtained.

The fact that, for large n, $E[\hat{n}] \approx n\phi$ suggests a way for correcting the estimate bias and allows to assume $\hat{n}_+ = \dfrac{\hat{n}}{\phi}$ as a estimate of n. Owing to the contribution of periodic functions, $\hat{n}_+$ is not an asymptotically unbiased estimate of n, in the sense that $E[\hat{n}_+]/n$ does not tend to 1 as n gets large. Fortunately, the amplitude of the periodic functions turns out to be less than $2 \cdot 10^{-5}$, so this bias is negligible for all practical purposes.

Interestingly, a simple variant of the estimation algorithm has provably disastrous performance. Consider the algorithm in which each station involved in the initial collision transmits to he channel with probability $\frac{1}{2}$. If this causes another collision, then those that just transmitted, transmit again with probability $\frac{1}{2}$. The others drop out. This continues, with the stations trying to transmit always being a subset of those that just transmitted, until there is no collision. Take 2^i as the estimate of the multiplicity of conflict where i is the number of the steps until ther is no collision. It can be shown that the second and all higher moments of this estimate are infinite.

3.3.1 base *b* variant

Using basic Algorithm 3, even though the expected value of $\hat{n}_+$ is quite close to n, $\hat{n}_+$ is likely to differ from n by a factor of 2. In the original work a small generalization of the base 2 algorithm is proposed to overcome this limitation: providing an estimate whose mean is close to n but whose distribution peaks more sharply about the mean.
Simply it is suggested to use b instead of 2 as base, with $1 < b \leq 2$.

Algorithm 4 BASE b GREENBERG ($\mathcal{B}$)

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

$\phi(b)$ corrects the bias of the estimator. $\phi(b)$ is the optimal correction when n is large. Looking at Table 3.2 you can notice how smaller b results in smaller $\phi(b)$. This means that b deeply biases the estimate: if $b' < b''$ then $E[\hat{n}(b')] < E[\hat{n}(b'')]$.
This is a result of the following fact: let i'' be the expected slot the base b'' algorithm will

end up given a batch size n, then $i'' \leq log_{b''} n$. Let i' be the same for b'. If $b' < b''$ then $b'^{i'} > b''^{i''}$ †

Table 3.2: Following table shows how ϕ and "Expected *Greenberg base b algorithm* cost in slots" vary for different b. Expected cost ($\lesssim \log_b n$) is expressed as a multiplicative factor for the basic Greenberg algorithm cost ($\lesssim \log_2 n$).

b	$\phi(b)$[a]	Expected cost in slots-1
2	≈ 0.9142	$\lesssim 1 \times \log_2 n$
1.1	≈ 0.3484	$\lesssim 7.27$
1.01	≈ 0.1960	$\lesssim 69.66$
1.001	≈ 0.1348	$\lesssim 693.49$
1.0001	≈ 0.1027	$\lesssim 6931.81$

[a] Code used to compute $\phi(b)$ is provided in Appendix A.3

It can be shown [6] that, *for all n greater than some constant* $n_0(b)$

$$\left| \frac{E[\hat{n}_+(b)]}{n} - 1 \right| < \epsilon(b), \tag{3.17}$$

$$\frac{\sigma\big(\hat{n}_+(b)\big)}{n} < \epsilon(b), \tag{3.18}$$

where $\epsilon(b) \to 0$ *as* $b \to 1$.

In other words when $b \to 1$ and n is large the estimate becomes unbiased and variance goes to 0: we have ideally a perfect estimator.

Our experimental results showed that this base b estimator is not so good for real life scenarios.

3.4 Window Based

†This comes from experimental observations.

Chapter 4

Estimate Performance Analysis

We presented some algorithms for multiplicity estimation in the previous Chapter.
Now we would provide the reader a practical overview of the results achieved by each estimation scheme.
We will analyze *Cidon* to show which is the minimum amount of nodes to be resolved for the estimate to be within user-required constrains with high probability.
We will analyze the behavior of *Greenberg* when varying the batch size.
Finally, we propose and analyze a modification of *Greenberg* which tries to refine the raw estimate provided by the original algorithm.

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. Consequently J is a binomial distribution with parameters $B(n, p_\epsilon)$ and recalling (3.6) we get:

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

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.

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

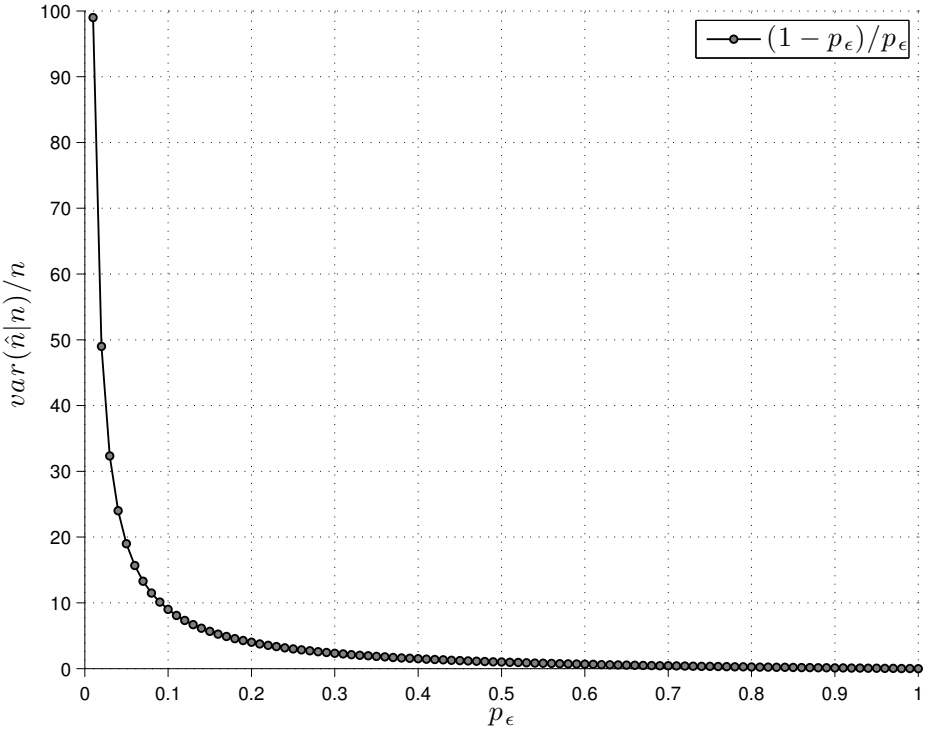

Figure 4.1: *Cidon*: estimate accuracy dramatically improves while $p_\epsilon \leq 0.1$.

Variance is strict monotonically decreasing in p_ϵ. Anyway it is difficult to establish in what measure an estimate can be considered accurate.

Given n, let $k \geq 1$ define the minimum required accuracy in the following way:

$$\frac{n}{k} \leq \hat{n} \leq kn \tag{4.3}$$

Let θ be the probability we require for constrains (4.3) to be satisfied.

If we set $\theta = 0.99$, we can find the minimum p_ϵ, ensuring the estimate to be within confidence interval (4.3), by solving the following problem.

$$\begin{aligned}P\left(\frac{n}{k} \leq \hat{n} \leq kn \middle| k, n\right) &= P\left(\frac{n}{k} \leq j/p_\epsilon \leq kn \middle| k, n, p_\epsilon\right) \\ &= P\left(\frac{np_\epsilon}{k} \leq j \leq knp_\epsilon \middle| k, n, p_\epsilon\right)\end{aligned} \tag{4.4}$$

Probability in (4.4) is well behaved and expresses a constrain for j, which is a value assumed by J mentioned in (3.5). Since J assumes positive integer values we introduce rounding operations. In particular, rounding effect is non neglectible when $n \leq 200$.

$$f(k, n, p_\epsilon) = P\left(\left\lceil \frac{np_\epsilon}{k} \right\rceil \leq j \leq \lfloor knp_\epsilon \rfloor \,\middle|\, k, n, p_\epsilon\right) \geq \theta \tag{4.5}$$

Fixed k, n and θ, p_ϵ can be found by numerically solving[1]:

$$f(k, n, p_\epsilon) = \theta \tag{4.6}$$

Figure below shows minimum p_ϵ in function of k for different batch sizes.

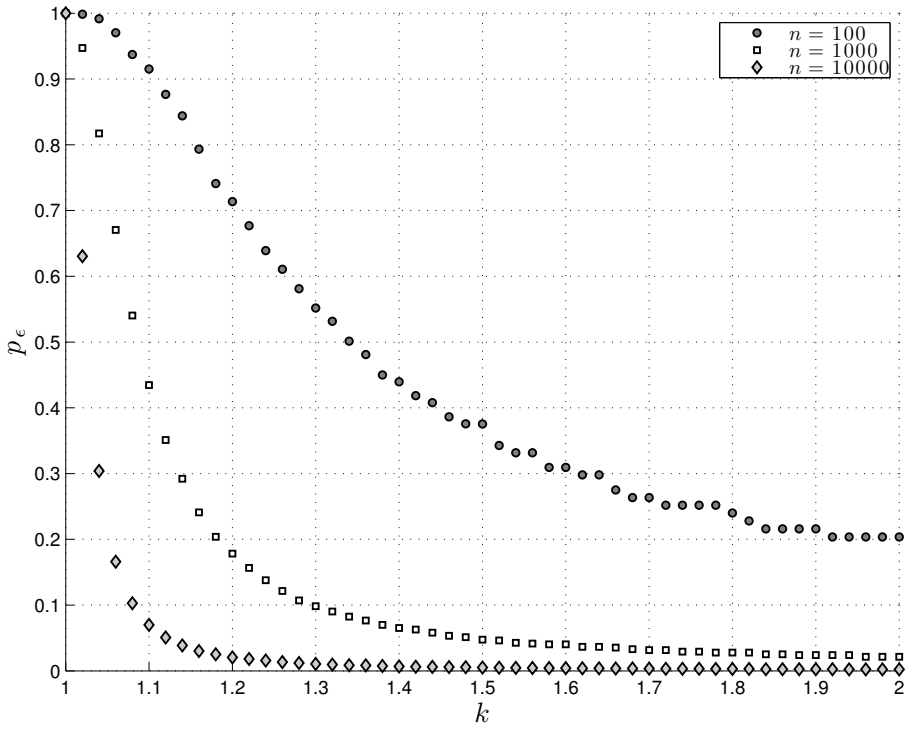

Figure 4.2: *Cidon*: Minimum p_ϵ required for accuracy k with $\theta = 0.99$. k step is 0.02

Figure 4.2 shows how the fraction of the initial batch to resolve for estimate to be within required confidence interval with high probability deeply depends on the size of the problem: smaller sizes require much higher p_ϵ.
At the same time, considering absolute cost in elapsed slots, Figure 4.3 shows that, for a wide range of k, the time required is quite independent of the size of the problem.
Time required by the largest considered n provides a bound for smaller ones.

[1] in this work we used bisection method

Note that in Figure 4.3 an upper bound on the expected BRI time taken is plotted: this bound is not so tight for very small batches. With very small batches BRAs performs better than what is reported.

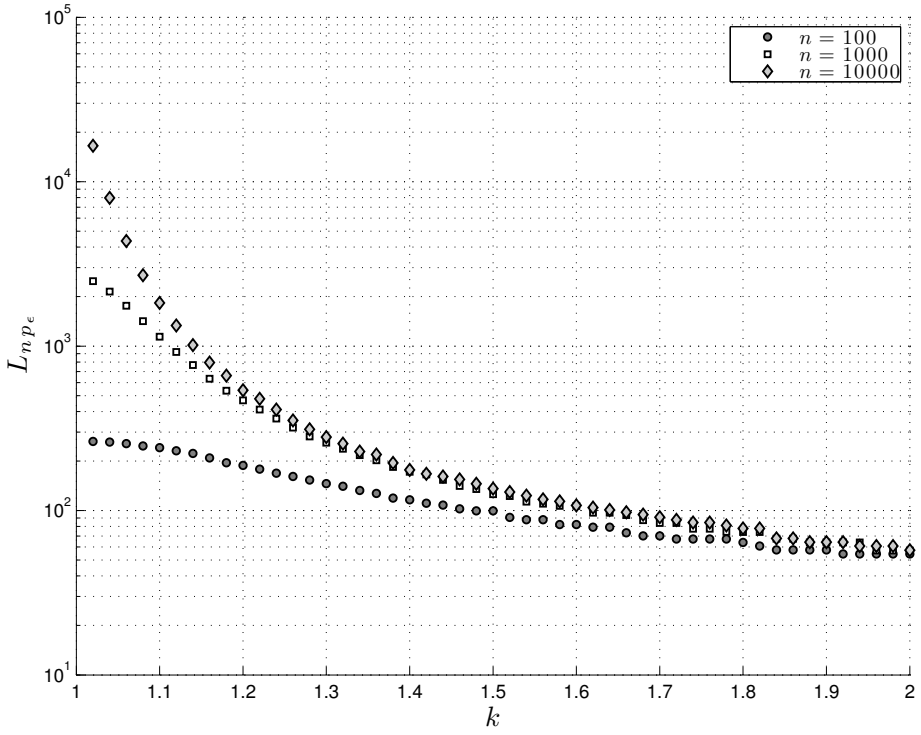

Figure 4.3: *Cidon*: Upper bounds on the expected time (in slots) required to achieve accuracy k with $\theta = 0.99$. k step is 0.02

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)

$$q_0(p, n) = (1 - p)^n \tag{4.7}$$

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

$$q_1(p, n) = np(1 - p)^{n-1} \tag{4.8}$$

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

$$q_{2+}(p, n) = 1 - q_0(p, n) - q_1(p, n) \tag{4.9}$$

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

$$p_i = p(i) = 2^i \tag{4.10}$$

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

$$f(n,i) = \prod_{k=1}^{i-1} q_{2+}(p_k, n) \cdot \big(q_0(p_i, n) + q_1(p_i, n)\big) \tag{4.11}$$

An overview of the behavior of $f(n,i)$ is presented in table A.2 on page 59.
Equation (4.11) defines the probability for biased estimate $\hat{n}$ to be equal to 2^i when the batch size is n.

$$\Pr\big(\hat{n} = 2^i | n\big) = f(n,i) \tag{4.12}$$

Following Figures 4.4 and 4.5 show how the distribution behaves respectively for small and large sizes. We note that, for any fixed n, the distribution is well behaved: initially it is monotonically increasing, then a monotonically decreasing part follows.
It turns out that, for batch sizes larger than 128, the distribution is "stable" in the sense that doubling the number of nodes produces a shift of 1 slot to the right but the values assumed are the same (see Figure 4.5).

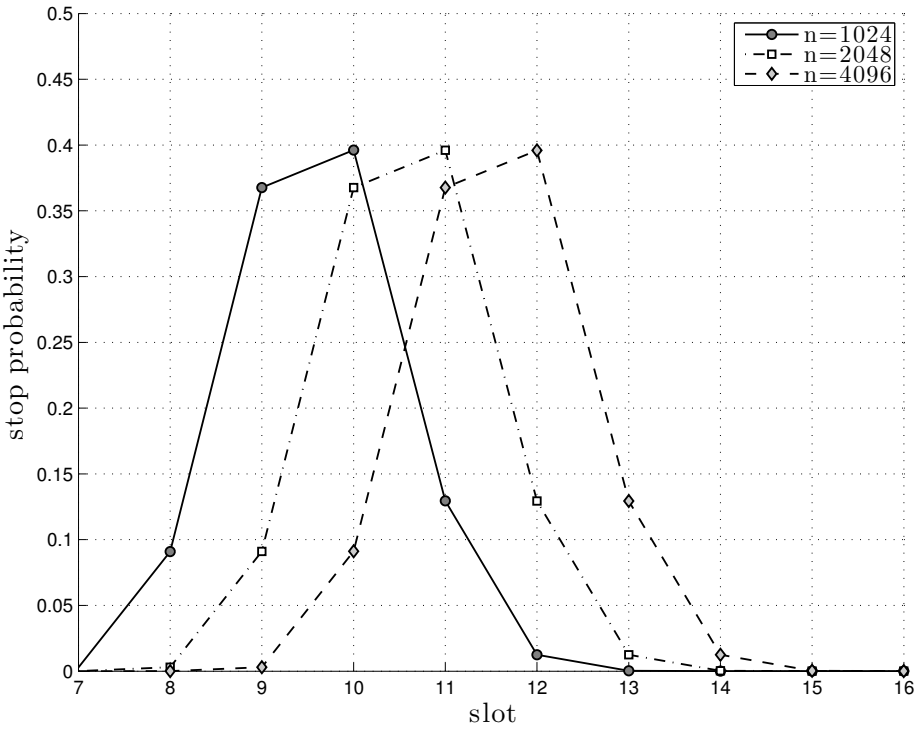

Figure 4.5: *Basic Greenberg*: large 2^k sizes distribution.

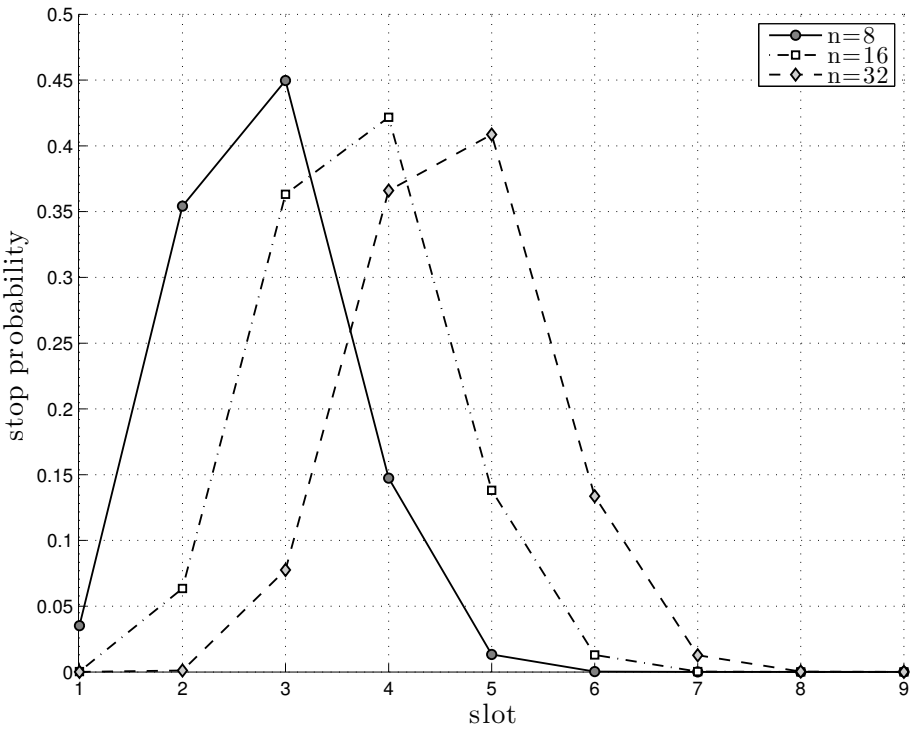

Figure 4.4: *Basic Greenberg*: small 2^k sizes distribution.

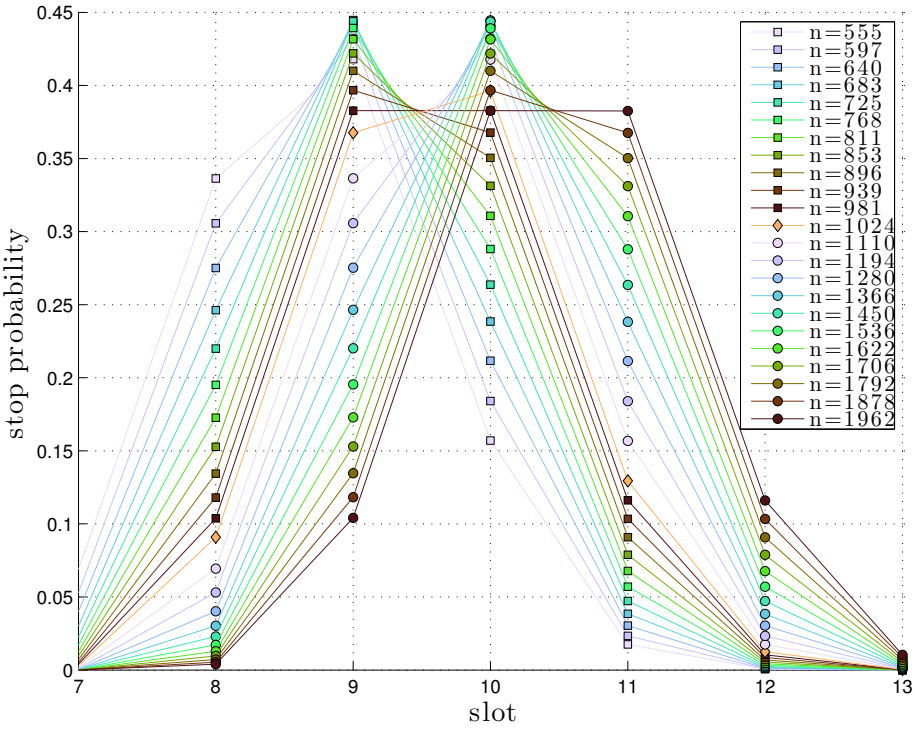

Figure 4.6: *Basic Greenberg*: general sizes distribution.

This is not only true for sizes power of two but for any size. Figure 4.6 shows the case where size n' has color c and size $2n'$ has, again, color c but a different marker: values are the same but shifted right. Interestingly the highest peaks are located in $n = 683$ and $n = 1366$.

4.2.1 base b Greenberg

4.2.2 Considerations

Greenberg method is really good from the point view of running cost since it is $O(\log_2 n)$ respect to the size n of the problem but it has also some non negligible drawbacks:

a) estimation phase results in a sequence of colliding messages. These provides informations about the cardinality of the batch but do not help to solve an, even small, portion of the eventually following batch resolution problem and can not carry auxiliary infromations. An algorithm that allows to get an estimate while transmitting successfully messages would offer some advantages when the problem is not only the pure estimation but also a subsequent resolution.

b) it does not allow to achieve arbitrary precision in the estimate.
In fact we have that:

i. the estimate is, by construction, a power of 2. Only a small subset of batch sizes can be mapped without any error.

ii. the end-up distribution is not sharp enough but it spans over a few slots: this is shown by Figure 4.5. The Figure shows that, for the examined batch sizes, fixed a problem of size n we have about:

- 0.02‰ estimate is $16n$;
- 3‰ estimate is $8n$;
- 1.2% estimate is $4n$;
- 12.9% estimate is $2n$;
- 39.6% estimate is n; ✓
- 36.8% estimate is $\frac{n}{2}$;
- 9.1% estimate is $\frac{n}{4}$;
- 3 ‰ estimate is $\frac{n}{8}$;
- 0.02‰ estimate is $\frac{n}{16}$;

It is difficult to discriminate between n and $\frac{n}{2}$

c) to get a tighter estimate, base b Greenberg has to be used. Anyway to improve the accuracy very small b has to be used (see Table 3.2) and this results in worse running times: even if theoretically it remains $O(log_b n)$ which is a lower order term compared to n, in practice estimate time can be no more neglected. In this case, for reasons expressed in **a)**, using Greenberg could be a bad choice.

4.3 Greenberg with MLE

Let n be a batch size and p be a given transmission probability. As expressed by (4.7) (4.8) (4.9), varying n while p is fixed results in very different probabilities for *idle*, *successful* and *collided* slots.

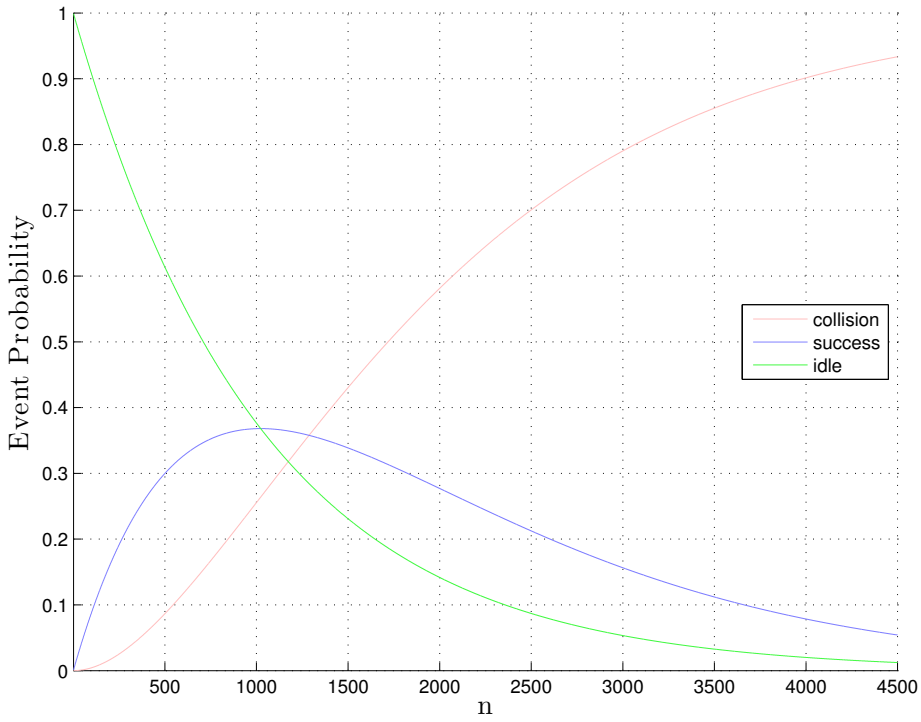

Figure 4.7: Probability of *idle*, *successful* or *collided* slots varying n while $p = 1/1024$. $q_0(p,n) \approx q_1(p,n)$ for $n = 1023$

It is quite immediate to see that:

- $q_0(p,n) \approx q_1(p,n)$ when $p \approx \dfrac{1}{n}$ and, obviously,
- $q_0(p,n) \gg q_1(p,n)$ and $q_0(p,n) \gg q_{2+}(p,n)$ when $p \ll \dfrac{1}{n}$,

- $q_{2+}(p,n) \gg q_0(p,n)$ and $q_{2+}(p,n) \gg q_0(p,n)$ when $p \gg \frac{1}{n}$.

$q_{2+}(p,n)$ is strictly increasing monotonic while $q_0(p,n)$ is strictly decreasing. Repeating a large number T of test transmission with probability p each, we can simply use the number of collisions or idle slots to uniquely determined the batch size. On the other hand, successful probability $q_1(p,n)$ is non-monotonic and hence can not be used to uniquely determine the batch size.

The proposed technique to refine the estimate is the following.
Running *Basic Greenberg Algorithm* provides a raw estimate and the associated transmission probability p. Then we perform T more transmissions and use a *Maximum Likelihood Estimation (MLE)* to refine the estimate.

Let T be a number of slots of our choice and N_0, N_1, N_{2+} random variables which represent the number of time slots with no transmissions, one transmission and collisions respectively.
N_0, N_1, N_{2+} are respectively binomial distributions with parameters $B(n, q_0(p,n))$, $B(n, q_1(p,n))$ and $B(n, q_{2+}(p,n))$. Of course $N_0 + N_1 + N_{2+} = T$ and

$$\begin{aligned} E[N_0] &= Tq_0(p,n), \\ E[N_1] &= Tq_1(p,n), \\ E[N_{2+}] &= Tq_{2+}(p,n). \end{aligned}$$

Let $f_T(i,s,c,p',n')$ denote the probability to get i idle, s success and c collision slots

$$\begin{aligned} f_T(i,s,c,p',n') &= \Pr(N_0 = i, N_1 = s, N_{2+} = c | n = n', p = p') \\ &= \Pr(N_0 = i | n = n', p = p') \Pr(N_1 = s, N_{2+} = c | n = n' - i, p = p'), \end{aligned} \tag{4.13}$$

when we transmit with probability p' and considered batch size is n'.
Let $\mathcal{N}$ be the set of our considered batch sizes, then we can compute

$$MLE(i,s,c,l) = \underset{n' \in \mathcal{N}}{\arg\max} \Big(f_T\big(i,s,c,p(l),n'\big) \cdot f\big(n',l\big) \Big). \tag{4.14}$$

$MLE(i,s,c,l)$ is $T \times T \times L$ matrix where L is the maximum slot that can be visited during *Basic Greenberg Algorithm*. L depends on the maximum cardinality n_{max} considered for the problem. Consequently L must be chosen so not to worse the performance of our estimator (if n_{max} is smaller than the real batch size n) and not to waste too much

memory. Setting $L = \lceil \log_2 n_{max} \rceil$ is a good practical choice.
Hence, our estimate is simply the result of a look-up in the MLE table,

$$\hat{n} \longleftarrow MLE(i, s, c, l) \tag{4.15}$$

Is it worth to mention that:

- The MLE table must be precomputed before running the estimate algorithm. The computational cost to fill the MLE table depends on $|\mathcal{N}|$ and L and, even if in general the cost is really reasonable, it does not allow for real-time computation.
- The size of the MLE table does not depend on $|\mathcal{N}|$ but only on T and L.
- The estimate deeply depends on the way the elements in $\mathcal{N}$ are chosen.

Chapter 5

Comparison

Appendix A

A.1 Probability

sezione provvisoria

A.1.1 Chebyshev's inequality

Let X be a *random variable* with expected value μ and finite variance σ^2. Then for any real number $k > 0$,

$$\Pr(|X - \mu| \geq k) \leq \frac{\sigma^2}{k^2} \tag{A.1}$$

A.1.2 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.1.3 Poisson Distribution

A.1.4 Normal Distribution

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

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

A.2 Binary Trees Performance Summary

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

A.3 Greenberg bounded m-moments

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

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

$$\Phi(b) = \frac{1}{\log b} \int_0^\infty e^{-x}(1+x) \prod_{k=1}^{\infty} (1 - e^{-b^k x}(1+b^k x)) x^{-3}\, dx \tag{A.4}$$

$\phi(b)$ was computed using the code below. Even if the integral is improper the integrand function goes fast to 0 and so the product does when k grows. We found that considering interval (0,40) for numerical integration provided good results.

matlab/Greenberg_base_b/phi/calculatephi.m

```
%sample script to calculate phi

clc;
clear all;
close all;

b=2 % 1<b<=2
phi=quadl(@(x)f4(x,b),0,40,1.e-9)/log(b) ;
phi

% Sample output
%
% b =
%     2
% phi =
%   0.914217701315935
```

matlab/Greenberg_base_b/phi/f4.m

```
function y=f4(x,a)

% y=f4(x,a)
%
% input:
%     a: base
%     x: x-point
%
% implements the function to ingegrate to calculate phi

y=exp(-x).*(1.+x).*f3(a,x).*x.^-2;
```

matlab/Greenberg_base_b/phi/f3.m

```
function p=f3(a,x)

% p = f3(a,x)
%
% compute the infinite product stopping when the terms converge to 1

l=length(x);
p=zeros(1,l);

for k=1:l
    do=true;
    i=1;
    %valore al passo precedente
    y0=f2(a,x(k),1);
    v0=1;
    p(k)=y0;
```

```
17      while (do)
18        v1=bitshift(1,i);
19        %valore al passo attuale
20        y1=f2(a,x(k),v1);
21        if (y0==y1)
22              % allora converge a 1
23              do=false;
24        else
25            for ii=v0+1:v1
26              p(k)=p(k).*f2(a,x(k),ii);
27            end
28            y0=y1;
29            v0=v1;
30        end
31        i=i+1;
32      end
33  end
```

matlab/Greenberg_base_b/phi/f2.m

```
1  function y=f2(a,x,k)
2
3  % y = f2(a,x,k)
4  %
5  % a term inside the product
6
7  c=a.^k;
8  y=(1-exp(-c.*x).*(1+c.*x));
```

A.4 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

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

% asc sorting
```

```
nodes=[sort(nodes); 1];

if (n<2)
    error('BRA must start with a collision');
end
% CBT Simulation

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

imax=length(nodes);  %index of the first node in the batch
imin=1;    % index of the first node in the batch
xmin=0;    % starting interval [0,1/2)
xlen=1/2;  % we suppose a collision already occurred.

while (waitforconsecutive)
    [e,imin,imax]=cbtsplit(nodes,imin,imax,xmin,xlen);
    % update next analyzed interval

    if(e==1)
        xmin=xmin+xlen;
        %xlen=xlen;
    elseif (e==0)
        xmin=xmin+xlen;
        xlen=xlen/2;
    else
        %xmin=xmin;
        xlen=xlen/2;
    end
    if (lastwassuccess==true && e==1)
        disp(' ');
        disp('CBT completed :');
        disp(['Estimate :' num2str(1/xlen)]);
        disp(['Last node transmitting :' int2str(imin-1)]);
        waitforconsecutive=false;
    end
    if(e==1)
        lastwassuccess=true;
    else
        lastwassuccess=false;
    end
end

% DEBUG
% estimate is given by the first serie of descending differences in the
% nodes ID's
dif=-1*ones(n-1,1); %negative init
for i=1:n-1
    dif(i)=nodes(i+1)-nodes(i);
end

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
47 % if event is collision
48 % update the next enabled set
```

```
49      e=2;
50      while ((imax-1)~=0 && xmax<nodes(imax-1))
51          imax=imax-1;
52      end
53  end
```

matlab/CBT/cbtfulltest.m

```
1  % Marco Bettiol - 586580 - BATCH SIZE ESTIMATE
2  %
3  % CBT Full Test
4  %
5  % Estimate distributions obtained with CBT fixed different n
6  %
7  % Based on cbtsimpletest code
8
9  clear all;
10 close all;
11 clc;
12
13 % input
14 logn_max=15;
15 c=100000;
16
17 % generate the test batch size
18 x=1:logn_max;
19 testsizes=2.^x;
20
21 % resulting estimate distribution
22 % ED(log2(batch size), log2(estimate batch size));
23
24 ED=zeros(length(testsizes),length(testsizes)+40);
25
26 for i=1:length(testsizes)
27     n=testsizes(i);
28     disp(['Testing size :' int2str(n)]);
29     if (n<2)
30         error('BRA must start with a collision');
31     end
32     for ii=1:c
33         %generate the batch
34         nodes=rand(n,1);
35         nodes=[sort(nodes); 1];
36         % CBT Estimate Simulation
37
38         % true if we got a success in the last transmission
39         lastwassuccess=false;
40         %false to end CBT
41         waitforconsecutive=true;
42
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.5 Greenberg Estimate Distribution

In following table A.2 we report how the end up probability (equation 4.11) 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'*)

n	$\hat{n}$ slot:	2 1	4 2	8 3	16 4	32 5	64 6	128 7	256 8	512 9	1024 10	2048 11	4096 12	8192 13	16384 14	32768 15	65536 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

n	$\hat{n}$ slot:	128 7	256 8	512 9	1024 10	2048 11	4096 12	8192 13	16384 14	32768 15	65536 16	2^{17} 17	2^{18} 18	2^{19} 19	2^{20} 20	2^{21} 21	2^{22} 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] K. Jamieson, H. Balakrishnan, and Y. C. Tay, *Sift: a MAC protocol for event-driven wireless sensor networks*, Proc. 3rd European Workshop on Wireless Sensor Networks, Zurich, Switzerland, February 2006, pp. 260–275

[4] Dimitri Bestekas, Robert Gallager, *Data Networks*, 2nd Ed., Prentice-Hall, New Jersey,1992.

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

[6] 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

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

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

[9] B. S. Tsybakov, V. A. Mikhailov, *Slotted multiaccess packet broadcasting feedback channel*, Probl. Peredachi Inform. vol. 13, December 1978, 32-59

[10] J.I. Capetanakis, *A protocol for resolving conflicts on ALOHA channels*, Abstracts of Papers, IEEE Int. Symp. Info. Th., Ithaca, New York, October 1977, 122-123