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: Saturday 13th February, 2010 12:05

# Contents

# 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.

Nowdays 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*.

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 take 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* and the node that transmitted *resolved*;
- 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 at the beginning of the algorithm there is no *resolved node* and that they all are in the initial batch. Another consideration is that a node that has already been resolved in the current process no longer take place in it and it remain quiet until the current resolution ends up, eventually waiting for the next one to start.

# Chapter 2

# Batch Resolution

Most of the batch resolution algorithm were originally developed for slotted ALOHA 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.

Furthermore binary tree algorithms require explicit feedback about channel status. So they require 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.

## 2.1 Binary Tree Algorithms

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

### 2.1.1 Basic Binary Tree

At slot $\tau$ 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: B batch with |B| = n
  each node transmit its message
  if (idle or success) then
      conflict resolution ended.
  else
      each node flip a coin
      R ← { nodes that flipped head}
      S ← { nodes that flipped tail}
      COLLISION BINARY TREE (R)
      COLLISION BINARY TREE (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.

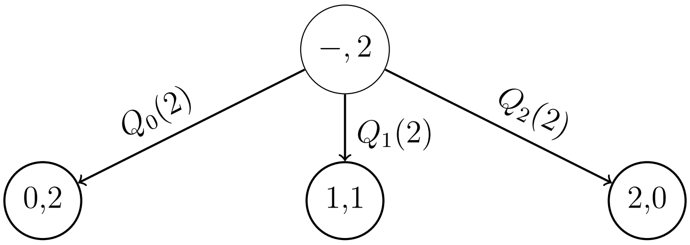


Figure 2.1: Transaction probabilities to split a set of 2 elements into two sets with $i$, $j$ elements

$L_n$ can be recursively computed (considering the factorial in $Q_i(n)$) collecting $L_n$ in the 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.

| | | | | |
|---|---|---|---|---|
| $L_2 = 5.0000$ | $L_7 = 19.2009$ | $L_{12} = 33.6238$ | $L_{17} = 48.0522$ | $L_{22} = 62.4783$ |
| $L_3 = 7.6667$ | $L_8 = 22.0854$ | $L_{13} = 36.5096$ | $L_{18} = 50.9375$ | $L_{23} = 65.3636$ |
| $L_4 = 10.5238$ | $L_9 = 24.9690$ | $L_{14} = 39.3955$ | $L_{19} = 53.8227$ | $L_{24} = 68.2489$ |
| $L_5 = 13.4190$ | $L_{10} = 27.8532$ | $L_{15} = 42.2812$ | $L_{20} = 56.7078$ | $L_{25} = 71.1344$ |
| $L_6 = 16.3131$ | $L_{11} = 30.7382$ | $L_{16} = 45.1668$ | $L_{21} = 59.5930$ | $L_{26} = 74.0198$ |

Considering the efficiency $\eta_n = n/L_n$ (messages over slots) we have a decreasing serie $\eta_1 = 1$, $\eta_2 = 0.40$, $\ldots$, $\eta_{16} = 0.3542$ , $\ldots$, $\eta_{31} = 0.3505$. It can be shown [4] 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.

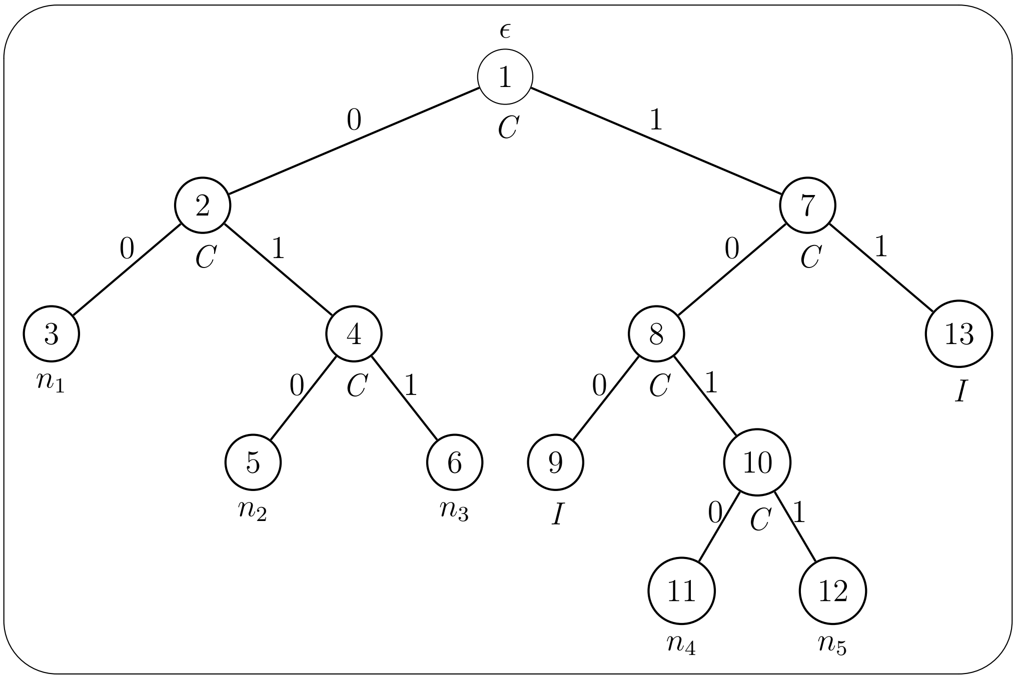


Figure 2.2: An istance of the binary tree algorithm for $n = 5$ nodes. The number inside the each circle identifies the slot number. The label below identifies the event occurring: $I$ for *idle*, $C$ for *collision*, $n_i$ for resolution of node $i$

.

### 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 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 $\epsilon$, the root address. $\epsilon$ is considered to be prefix of any string.
Let $b_{1..k}$, with $b_i \in \{0,1\}$, 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 successfully resolved the right part of the sub-tree, now we will look for right one. Let $t = \arg\max_{i \in 1..k} | b_i = 0$, in other words the position of the right most 0 in the prefix, if any. The new enabled interval will be $b_{t-1}1$.
If $t$ is undefined then in $b_{1..k}$ there is no 0. This means that $b_{1..k} = \epsilon$ or $b_{1..k} = \underbrace{1 \dots 1}_{k}$ in both cases this indicates all the three has been explored.
So this work also for termination condition.
You can see this rule applied after slot 6 and 12 for right branch exploration and 13 for termination.

### Real value approach

Decidere se inserire o meno le considerazioni sulla visione degli indirizzi dei nodi come numeri reali tra 0 e 1 e della risoluzione come intervalli reali abilitati contenenti un solo nodo

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.

So we could think, instead of tossing a coin only when needed, to initially to 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 $2^{-L}$

### 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 $(\tau)$, 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 $(\tau)$ 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.

Expected time analysis is similar 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

$$L_n^{MBT} = (1 - Q_0(n)) + \sum_{i=0}^{n} Q_i(n)(L_i^{MBT} + L_{n-i}^{MBT}) \tag{2.2}$$

with

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

Intuitively in this case, since an higher probability to stay silent, reduce 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 that best achievable result is for $p = 0.4175$ and, with this $p$, efficiency $\eta \approx 0.381$ as $n \to \infty$ which is asinthoticaly +10% faster than basic BT.

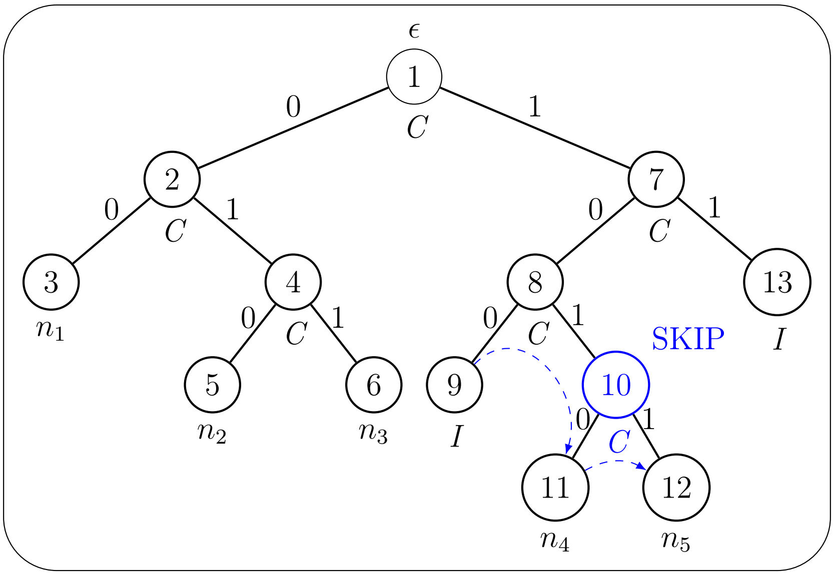


Figure 2.3: Same example as in Figure 2.2 but using MBT: tree structure do not change but node 10 is skipped in the traversal. $\tau = 8$

# 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.

## 3.1 CBT

non è propriamente di stima ma di risoluzione parziale, cmq può essere usato come alg di stima

## 3.2 Cidon

risoluzione di un batch parziale. mapping come in “Real value approach”

## 3.3 Greenberg

Greenberg algorithm is simply straightforward.

**Algorithm 2** GREENBERG

```
i ← 0
repeat
    i ← i + 1
    choose to transmit with probability 2^{-i}
until no collision occurs
n̂ ← 2^i
```

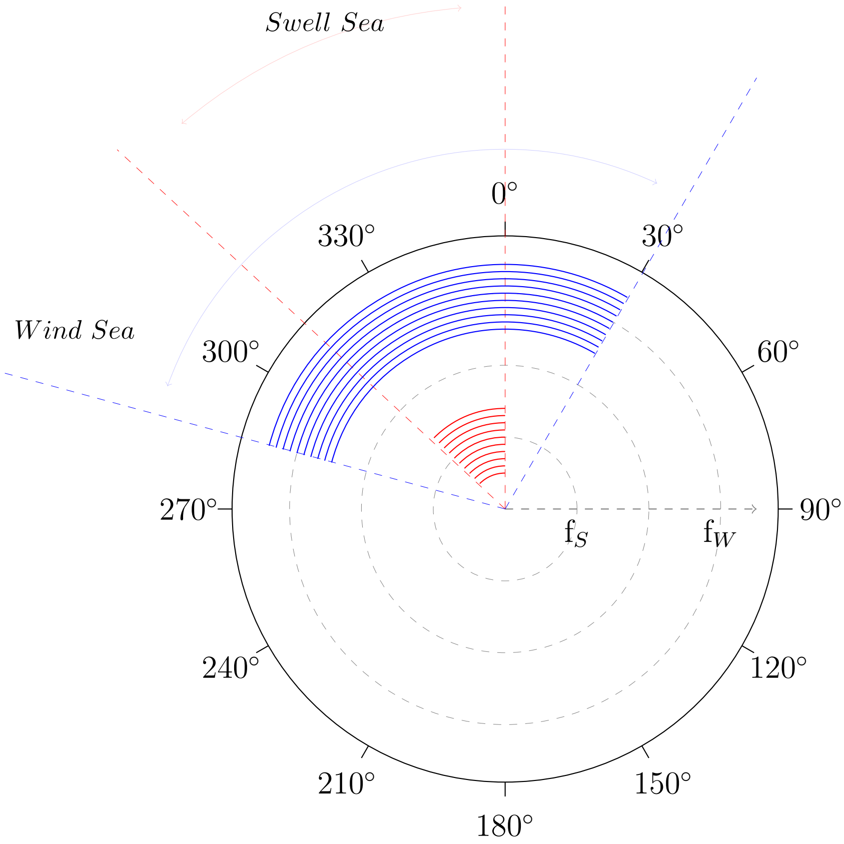


Figure 3.1: Immagine provvisoria da usare come base di partenza

The idea behind algorithm 2 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.

## 3.4 Window Based

# Chapter 4

# Inital Batch size estimate

# Chapter 5

# Comparison

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

[3] 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

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