Asynchronous Multiple Access Tree Algorithms

Mart L. Molle

Department of Computer Science
Computer Systems Research Group
University of Toronto
Toronto, Canada, M5S 1A4

Abstract — **Random access protocols allow large numbers of low duty cycle stations to exchange messages over a shared communications channel under distributed control. Recently, a new class of random access protocols called 'tree algorithms' has emerged. Tree algorithms offer several performance advantages over other random access protocols (e.g. ALOHA and CSMA), namely higher capacity and inherently stable operation. However, only synchronous (slotted) tree algorithms have so far been defined. Any practical implementation of a synchronous protocol is complicated by the need for stations to perform the steps of the algorithm synchronously. Thus asynchronous (unslotted) protocols are of greater practical importance, especially for local networks. Here we show how to construct asynchronous versions of several well-known tree algorithms, and describe some of the performance limitations that result from asynchronous operation.**

I. Introduction

Computer networks allow a set of intelligent devices (henceforth called *stations*) to communicate by exchanging messages over some communications channel(s). A 'channel' may be a point-to-point connection between two stations, or a broadcast connection that more than two stations can access. With a broadcast channel, any station connected to the channel can transmit a message that will be received by all other stations connected to the channel. However, the channel can only successfully deliver one message at a time. Whenever several transmissions overlap on the channel we say that a *collision* has occurred and assume that none of the affected messages will be received correctly. Thus successful operation of a broadcast channel depends on using some algorithm (with one copy running in each station) for 'serializing' the transmissions on the channel. Such algorithms are called multiple access protocols.

The operation of a multiple access protocol can be deterministic (i.e., round-robin scheduling like Time Division Multiple Access [1], MSAP [2], or BRAM [3]), or random (i.e., on-demand scheduling with some method of collision resolution, such as ALOHA [4], various 'tree' and 'stack' algorithms [5, 6, 7], or Carrier Sense Multiple Access [8, 9]). When a network is to support large numbers of low duty cycle stations, random access protocols offer several advantages over deterministic protocols, notably the lack of any requirement to encode the number (and perhaps the addresses) of all stations into the protocol, and throughput−delay performance that is insensitive to the number of stations in the network. We shall thus focus our attention on random access protocols below.

The most well-known examples of random access protocols are ALOHA and CSMA, an extension of ALOHA for local area networks. The main feature of these protocols is their simplicity. In ALOHA, for example, stations are allowed to transmit at will. Should this cause a collision, the stations that were involved are simply forbidden from retransmitting for a random time. More recently, a new class of protocols called *tree algorithms* has appeared in the literature — see [10] for a brief survey. These algorithms gather information by monitoring channel activity (i.e., the outcome of each slot). This information allows tree algorithms to make inferences about the current state of the network, and thus to improve their scheduling efficiency. Thus tree algorithms can attain higher channel utilizations than can 'inference avoiding' protocols like ALOHA, and their operation can be shown to be stable without the imposition of any external controls.

The operation of a multiple access protocol can either be synchronous (slotted) or asynchronous (unslotted). In a synchronous protocol, the all stations execute the algorithm in lock-step: opportunities to transmit occur only at the beginning of a *slot* of duration equal to a transmission time.[1] Both synchronous and asynchronous versions of ALOHA (and the various CSMA protocols) have been defined. However, to date only synchronous tree algorithms have been proposed, possibly because it is easier to make inferences when the channel history information is identical for all stations.

Here we introduce the notion of asynchronous tree algorithms and show how certain well-known synchronous tree algorithms can be modified to operate asynchronously. This construction appears as a natural extension of the synchronous protocols when we describe the operation of the algorithms in terms of a 'virtual clock' instead of the commonly-used 'sliding window' abstraction (see below). As an example, asynchronous versions of well-known tree and stack algorithms developed by Capetanakis [5, 11] and Tsybakov [7], respectively, are found.

* This research was supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada.

[1] In local networks, stations monitor the state of the channel and thus are able to shorten unused slots through *carrier sensing*, and possibly are able to shorten slots where collisions occur through *collision detection*.

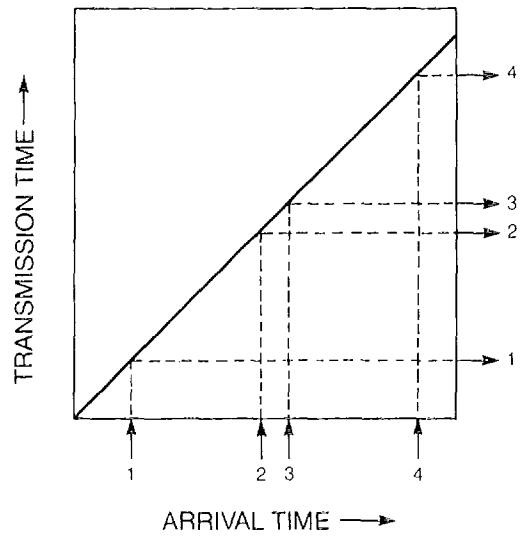

FIGURE 1a:
OPERATION OF UNSLOTTED ALOHA

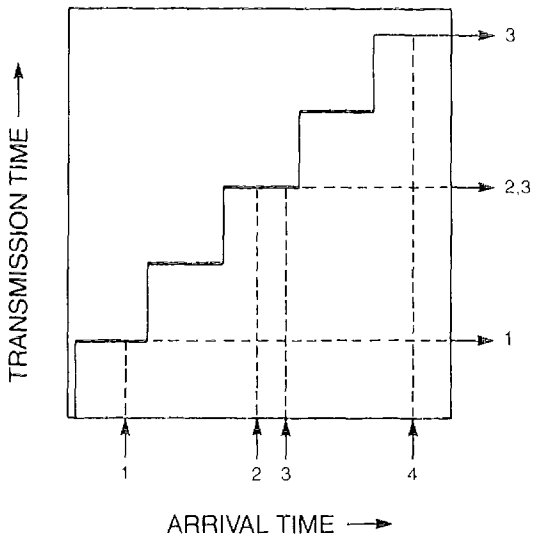

FIGURE 1b:
OPERATION OF SLOTTED ALOHA

Asynchronous algorithms simplify network implementation since there is no need to maintain a global time base. However, this simplification has its price. It is well known that the capacity of a protocol — its maximum attainable channel throughput — is less in the asynchronous case. In addition, we show that the FCFS property of certain synchronous sliding window tree algorithms (i.e., the delivery of messages in order of their generation times) is not preserved in the asynchronous versions.

II. The Virtual Clock Representation of Protocol Operation

An important contribution of the virtual time CSMA protocol was the 'virtual clock' representation of protocol operation [9]. In the virtual clock model, a message is transmitted whenever the virtual clock reading passes its arrival time. Many well-known multiple access protocols can be represented in this manner: if the transmission time for a message is some function of its arrival time (and possibly of the state of the channel), then that function can be used to control the motion of the virtual clock. For example, Figures 1a – 1b give a graphical representation of ALOHA. In 'pure' (unslotted) ALOHA — Figure 1a — each message is transmitted when it arrives. Thus the protocol may be represented by a virtual clock that runs continuously at constant speed, tracing out a straight line of unit slope on the Arrival time *vs.* Transmission time plane. In slotted ALOHA — Figure 1b — each message is transmitted in the first complete slot following its arrival. Here the virtual clock *ticks* (i.e., advances instantaneously) once at the beginning of each slot so that its reading forms regular 'staircase' along the diagonal.

The reason for introducing the virtual clock representation becomes evident when we consider 'sliding window' protocols [6]. In the synchronous case, the operation of a sliding window protocol is straightforward. At the beginning of each slot, the protocol chooses some time interval (the window) and enables all messages whose arrival times fall within the interval to be transmitted. From the outcome of that slot (i.e., whether it is idle, a success, or a collision) the protocol learns something about the distribution of messages in that window, which could affect the choice of subsequent windows. Slotted ALOHA is perhaps the simplest possible sliding window protocol; the choice of windows and duration of the corresponding slots is fixed *a priori*. If the normalized transmission time for a message is unity, then the window for the $K+1$st slot is simply the interval $(K, K+1]$. Minislotted virtual time CSMA [9] operates in a similar manner except for some complications necessary for it to choose constant-size windows (whenever the algorithm is far enough behind real time for this to be possible) even though the slot duration depends on the state of the channel. (Thus windowing is used as a means of flow control so that the channel will not become overloaded after a periods of activity — during which carrier sensing prevents stations from transmitting.)

The sliding window representation of protocol operation is clearly equivalent to the virtual clock representation for the case of synchronous protocols. However, the sliding window does not seem well suited to describing asynchronous protocols. Here the virtual clock representation suggests a natural generalization by allowing the motion of the virtual clocks at each station to occur *continuously* rather than in discrete ticks, depending on the algorithm and observed state of the channel. We note that the virtual clocks at different stations will no longer remain completely synchronized. However, for local networks (where the maximum propagation delay is small compared to the message transmission time) variations in virtual clock readings at different stations can be accounted for well enough to allow asynchronous versions of many algorithms to operate successfully.

III. The Stack Algorithm of Tsybakov and Vvedenskaya

The stack algorithm of Tsybakov and Vvedenskaya [7] provides a simple example of a tree algorithm. It is assumed in the model that all slots are of constant length and that feedback of the outcome of each slot is provided to all stations at the end of that slot. Here a vector of random length (i.e.,the 'stack') defines the state of the protocol. The actions of the protocol in each slot simply consist of enabling all messages assigned to the top element of the stack to be transmitted. Messages are assigned to an element of the stack in the following way. When new messages enter the system, they are assigned to the (current) top of the stack, and thus will be transmitted in the next slot. Following each slot where a collision does not occur, the stack is popped, possibly enabling some old message(s) to be retransmitted in the next slot. (If a message had been at the top of the stack, it must have been

transmitted successfully and so leaves the network.) Following each slot where a collision does occur, the stack is pushed down by one. To prevent a recurrence of this collision in a later slot, a coin toss is used to spread out the messages that were on top of the stack. Those that toss '1' are reassigned to the top of the stack while the others remain at the second stack element.[1] Note that stations need not know the complete state of the stack to participate in the algorithm.

It is easy to find a virtual clock representation for this synchronous stack algorithm. For now let us continue the assumptions of constant slot length and that both positive and negative acknowledgements are provided (at no cost) at the end of each transmission. Each message is provided with a virtual clock that ticks once per slot, thereby advancing by one unit of virtual time. The reading of each clock is initialized to the arrival time of the message. Thereafter, each virtual clock will advance by one time unit at the beginning of each slot. In addition, whenever a collision occurs, the virtual clocks for all messages (except new arrivals) are set back before the start of the next slot. If the message was involved in the collision, its virtual clock is set back by a random amount chosen uniformly in the range (0, 2) units; otherwise its virtual clock is set back by exactly two units.

For asynchronous operation, we let each virtual clock run continuously at constant speed towards the corresponding message arrival time. In addition, we must choose a larger set-back value following collisions. It is easy to show that the probability that two points chosen uniformly over an interval of length X are separated by at least ζX is given by $(1-\zeta)^2$, $0 \leqslant \zeta \leqslant 1$. Thus, to insure that the retransmissions of two colliding messages do not collide again does not exceed .5 (as was the case in the synchronous version), it is sufficient to choose a set-back value no smaller than $1/\zeta^* \approx 3.4$, where $0 < \zeta^* < 1$ solves $(1-\zeta^*)^2 = .5$.

IV. The Tree Algorithm of Capetanakis

The tree algorithm of Capetanakis [5, 11] operates similarly to the stack algorithm described above with the following modification. Here new messages are not permitted immediate access to the channel. Instead, they must wait for the start of the next *service epoch*. All messages that arrive during one service epoch will be transmitted during the following service epoch. A service epoch ends when it becomes known to all stations that all these messages have been transmitted successfully. Otherwise, the same 'recursive binary splitting' algorithm is used to resolve collisions within the epoch. (Indeed, the stack algorithm was conceived as an extension of this tree algorithm to overcome the need for stations to track service epochs.)

To model the operation of the basic Capetanakis algorithm, we proceed as follows. Since time is only used to assign messages to service epochs in this algorithm, let us simplify the discussion by assuming that clocks run towards zero (at which point messages are enabled), but are set back whenever collisions occur on the channel. One virtual clock is used for tracking the service epochs. This clock advances at a constant rate, but is set back by the *maximum* amount following each collision. Whenever the service epoch clock reaches zero, a new service epoch begins and all new messages are transmitted. If a collision occurs when a message is first transmitted, the message is provided with a separate virtual clock that operates exactly as described above for the stack algorithm: following collisions it is set back by a random amount if the message was involved in the collision and by the maximum amount otherwise. Each time a message clock reaches zero, the corresponding message is transmitted.

There is also an optimized version of the Capetanakis algorithm where the first-level splitting is into k parts, $k \geqslant 2$ [11, 12]. However, we find it useful to consider a slightly different optimization based on a sliding window. Here the messages selected for transmission during a service epoch must have arrived during a window (of constant size when possible) immediately following the window serviced during the previous service epoch. This windowing offers a slight performance advantage by avoiding some integer rounding in the epoch splitting. (Note that time is more significant for this algorithm, since the 'window' is now independent of the previous service epoch.) Once the set of messages has been selected, the same recursive binary splitting algorithm is used during the service epoch. In the virtual clock representation, one clock is used for tracking service epochs. As before, it runs forward at constant rate but is set back by the maximum amount for each collision. A message is first transmitted when this service epoch clock passes its arrival time. Should a collision occur, the message is provided with its own virtual clock, initialized to a random set-back value, which thereafter runs as described above for the stack algorithm.

V. Extension to Local Networks

Although the definitions of these protocols originally assumed that all slots were of constant length (and that feedback of the outcome in each slot was provided at the end of the slot), the extension to local networks is straightforward [10, 13].

In a local network, all stations monitor the state of the channel so that the length of a slot can be a function of the outcome (i.e., idle, success, or collision) in that slot. Let us assume that the average message transmission time is unity, that the end-to-end propagation time across the network is a, and that collision detection can be modelled by assuming that stations transmit message fragments of length b, $b \leqslant 1$, when a collision occurs. Thus, for synchronous protocols, the length of a slot will be a if it is idle, $1+a$ if it contains a successful transmission, and $b+a$ if it contains a collision.

It is worth noting that in some local networks, such as packet radio networks, the ratio of the collision- to idle-detect times is large. It can be shown [10] that the performance of standard tree algorithms is significantly worse than algorithms specifically designed to take advantage of the much-lower 'cost' of scheduling an idle period on the channel rather than a collision.

In this light, consider the recursive binary splitting strategy used by the tree and stack algorithms discussed above. Since the strategy does not apply sophisticated inferences to the channel feedback, the cost of idle slots and collisions was assumed to be equal, and (hopefully) the 'multiplicity' of collisions was usually two, binary splitting worked well in the original model. However, when $b/a \gg 1$, a protocol should be conservative, tolerating more (short) idle slots to avoid some (long) collisions. Thus we should consider a recursive $k$ary splitting strategy for such an environment, $k \geqslant 2$. Here the stack is pushed down by $k-1$ following each collision, so that the colliding messages at the top of the stack may be reassigned to k stack elements at random. It remains to find how k should depend on the ratio b/a.

If we wish to optimize the (generalized) Capetanakis algorithm, we must find how the expected length of a service epoch where j messages are transmitted, $\triangleq w_j$, depends on j. (Recall that in the Capetanakis algorithm, all messages that are ever allowed to be transmitted within an epoch must be transmitted successfully before the epoch ends.) It is not difficult to find recursive equations defining w_j in terms of w_i, $0 \leqslant i < j$, with binary splitting for the

[1] For illustrative purposes, we are ignoring the following improvement to the algorithm that was treated in [7]. If the slot following a collision is idle, then all messages from the collision must have tossed '0' so that a collision among these same messages is certain to occur in the next slot. This predictable collision can be avoided by treating idle slots where the nearest previous busy slot was a collision as a special case. Here the stack is neither pushed down nor popped, but some messages from the second stack element are probabilistically reassigned to the top of the stack.

case of fixed length slots [12] or for local networks [10]. Unfortunately, the corresponding results for the stack algorithm are surprisingly difficult to obtain because new messages can enter at any time and hence w_j depends on w_i for all $i=0,1,\cdots$ [7, 14]. Thus our optimization below will be directly applicable only to the tree algorithm.

For the case of local networks with $k$ary splitting, the combinatorics of the random reassignment becomes tedious for large j. However, the $j=2$ case is by far the most important since we will be using a conservative strategy. When two messages must be assigned to k stack elements at random, then for any assignment of one message, another collision can occur only if the other message is assigned to the same stack element, which occurs with probability $1/k$. Thus, it is easy to see that

$$w_2 = b + a + \frac{1}{k}[w_2 + (k-1)a] + \frac{k-1}{k}[2 + ka]$$
$$= 2 + a + \frac{k}{k-1}[b + ka]. \quad (1)$$

If we treat k as a continuous rather than integer valued variable, we can optimize Eq. (1) by differentiating with respect to k assuming b/a is fixed. Thus

$$k^* = 1 + \sqrt{1 + b/a}, \quad (2)$$

which is never less than two and grows as the square root of the ratio of collision- to idle-detect times. For example, if $a=.01$ and $b=1$ [8] then $k^* \approx 11$; here w_2 decreases from 4.05 to 3.23 as k increases from 2 to 11.

We are now ready to describe 'optimized' local network versions of the stack and tree algorithms described above. The stack algorithm acts like virtual time CSMA where the 'recursive $k$ary splitting' strategy is used as the retransmission strategy. The tree algorithm acts like virtual time CSMA with head-of-the-line priority classes [10]. All new messages are generated in the lowest priority class and thereafter, following each collision, the messages involved join the next higher priority class.

In the stack algorithm, one virtual clock is used to enable new messages to be transmitted. In the synchronous case, this clock advances at the beginning of each slot by the minimum of its current set-back from real time and the (constant) window size. In the asynchronous case, this clock advances at a constant rate whenever the channel is sensed idle. This rate is η times faster than real time if it is behind real time, and equal to real time otherwise. Messages that are not successfully transmitted on their first attempt are subsequently controlled by separate virtual clocks. These separate clocks are initialized to a set-back value chosen at random in the range $(0, T)$, an interval over which the virtual clocks can advance in k^* ticks in the synchronous case, and in time $k^* a/\zeta^*$ in the asynchronous case. Thereafter these clocks advance towards the message arrival time whenever the channel is sensed idle but are set back by T following collisions where the message is not involved, and by a random value $\leq T$ following collisions where the message is involved.

The operation of the tree algorithm in a two-station local network with carrier sensing and collision detection is shown in Figure 2. As in the stack algorithm, one virtual clock per station (the 'epoch clock') is used to enable new messages to be transmitted. The epoch clock advances as described above, ticking once per slot in the synchronous case and advancing at a constant rate when the channel is sensed idle in the asynchronous case. However, unlike the stack algorithm, in the tree algorithm all messages from one epoch are supposed to be transmitted before another epoch is begun. Thus, to implement the tree algorithm, the epoch clock is set back by T following each collision rather than being allowed to run ahead. Messages that have suffered a collision are controlled by separate virtual clocks as described for the stack algorithm above.

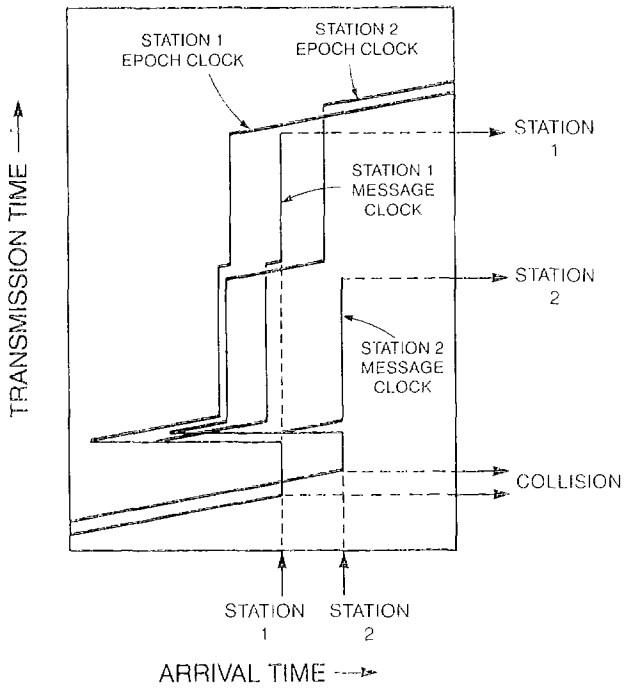

FIGURE 2: UNSLOTTED LOCAL NETWORK TREE ALGORITHM

VI. Conclusions

We have shown how to modify existing synchronous stack and tree algorithms to obtain asynchronous algorithms that could be used on local networks. Such asynchronous algorithms overcome one apparent disadvantage of tree algorithms over simpler random access protocols like ALOHA and CSMA, namely the need to maintain a global time base. In our choice of algorithms, we were also careful to avoid algorithms that use inferences that may be so 'clever' as to be troublesome in practice where the channel is unreliable and hence the channel state information could be inconsistent or incorrect [12, 15]. Thus, the algorithms described above seem to be good candidates for use in real single-hop random multiple access networks.

We must point out, however, that asynchronous operation can have surprising effects on the behaviour of some tree algorithms. In particular, some algorithms such as the synchronous Gallager-Tsybakov algorithm [6, 16, 17] guarantee that messages will be successfully transmitted in first-come first-served order. In general, this is not possible in the asynchronous version because there is no global time base for comparing message arrival times. Indeed, *every* station involved in a collision was the first to begin transmitting during a collision according to its own view of the channel history! When there is collision detection, however, there are some cases where the first-come first-served property can be preserved. For example, in a collision of multiplicity two, only the first station transmits longer than it detects interference. Thus if two colliding stations were to monitor the durations of both of their transmissions, both could agree on which station should retransmit first.

Finally, it must be pointed out that proper operation of an asynchronous tree or stack algorithm in a local network requires there to be a minimum length for messages (or message fragments if there is collision detection). Otherwise, it can be shown [18] that if transmissions of duration less than $2a$ were allowed, then it would be possible for stations to have an inconsistent count of the *number* of idle and busy periods on the channel.

References

[1] J. Martin, *Teleprocessing Network Organization,* Prentice-Hall, Englewood Cliffs, N. J. (1970).

[2] L. Kleinrock and M. O. Scholl, "Packet Switching in Radio Channels: New Conflict-Free Multiple Access Schemes," *IEEE Transactions on Communications* **COM-28**(7), pp.1015-1029 (July 1980).

[3] I. Chlamtac, W. R. Franta, and K. D. Levin, "BRAM: The Broadcast Recognizing Access Method," *IEEE Transactions on Communications* **COM-27**, pp.1183-1190 (August 1979).

[4] R. Binder, N. Abramson, F. F. Kuo, A. Okinaka, and D. Wax, "ALOHA Packet Broadcasting — A Retrospective," *AFIPS Conference Proceedings, NCC* **44**, pp.203-215 (1975).

[5] J. I. Capetanakis, "Tree Algorithms for Packet Broadcast Channels," *IEEE Transactions on Information Theory* **IT-25**, pp.505-515 (September 1979).

[6] R. G. Gallager, "Conflict Resolution in Random Access Broadcast Networks," *Proceedings of the AFOSR Workshop in Communication Theory and Applications*, pp.74-76 (Sept. 17-20, 1978).

[7] B. S. Tsybakov and N. D. Vvedenskaya, "Stack Algorithm for Random Multiple Access," *Problemy Peredachi Informatsii* **16**(3) (1980).

[8] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part I — Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics," *IEEE Transactions on Communications* **COM-23**(12), pp.1400-1416 (December 1975).

[9] M. L. Molle and L. Kleinrock, "Virtual Time CSMA: A New Protocol with Improved Delay Characteristics," CSD Report No. 810113, Computer Science Department, University of California, Los Angeles (January 13, 1981). Submitted to *IEEE Transactions on Communications.*

[10] M. L. Molle, "Unifications and Extensions of the Multiple Access Communications Problem," CSD Report No. 810730 (UCLA-ENG-8118), Computer Science Department, University of California, Los Angeles (July 1981). Ph.D. Dissertation.

[11] J. I. Capetanakis, "Generalized TDMA: The Multi-Accessing Tree Protocol," *IEEE Transactions on Communications* **COM-27**, pp.1476-1484 (October 1979).

[12] J. L. Massey, "Collision-Resolution Algorithms and Random-Access Communications," UCLA-ENG-8016, School of Engineering and Applied Science, University of California, Los Angeles (April 1980).

[13] D. Towsley and G. Venkatesh, "Window Random Access Protocols for Local Computer Networks," *IEEE Transactions on Computers* **C-31**(8), pp.715-722 (August 1982).

[14] G. Fayolle, P. Flajolet, and M. Hofri, "On a Functional Equation Arising in the Analysis of a Protocol for a Multi-access Broadcast Channel," 131, Institut National de Recherche en Informatique et en Automatique (April 1982).

[15] M. L. Molle, *Optimal CSMA Protocols for Poisson Traffic,* Computer Systems Research Group, University of Toronto. (in preparation).

[16] B. S. Tsybakov, M. A. Berkovskii, N. D. Vvedenskaja, V. A. Mikhailov, and S. P. Fedorzov, "Methods of Random Multiple Access," *Fifth International Symposium on Information Theory* (July 7-9, 1979).

[17] J. Mosely, "An Efficient Contention Resolution Algorithm for Multiple Access Channels," LIDS-TH-918, Laboratory for Information and Decision Systems, MIT, Cambridge, Mass. (June 1979).

[18] J. A. Field and J. W. Wong, "An Analysis of a Carrier Sense Multiple Access System with Collision Detection," CCNG E-Report E-95, Computer Communications Networks Group, University of Waterloo, Waterloo, Canada (May 1981).