Interference Cancellation Tree Algorithms with k-Signal Memory Locations

G. T. Peeters and B. Van Houdt
University of Antwerp - IBBT
Middelheimlaan 1, B-2020 Antwerp, Belgium

***Abstract*—Recently, tree algorithms have been combined with successive interference cancellation to achieve a substantially higher maximum stable throughput (MST). All previous work assumed either a single or an unbounded number of signal memory locations, with MSTs of 0.662 and 0.693, respectively. In this paper, we address the gap between these two algorithms by designing and analyzing two novel general k-signal memory location algorithms.**

I. Introduction

Tree algorithms have been proposed as important solutions for multiple access channels. For example, they have been recognized as important (if not, superior) contenders during the development of the 802.14 standard [8] for HFC networks. Tree algorithms also strongly outperform the class of backoff algorithms (including the binary exponential backoff (BEB)) in terms of their maximum stable throughput (MST); the MST of BEB is zero in the infinitely-many users multiple access model [1]. In the standard information theoretical setting, the MST is defined as the highest possible (Poisson) input rate for which a packet has a finite delay with probability one, with infinitely-many users. The first tree algorithms were independently developed in the late 1970s by Capetanakis [4] and Tsybakov, Mikhailov and Vvedenskaya [15]. These algorithms were the first to have a provable MST above zero. Afterward new tree algorithms were developed with MSTs as high as 0.4878 using the standard information theoretical multiple access model [3], [6], [13].

The 0.4878 MST, realized under the standard information theoretical model, has been exceeded in various manners by introducing additional mechanisms not available under the standard model, such as energy measurement techniques to determine the collision multiplicity [10] and additional control field/bits with separate feedback [9], [14]. Recently, the successive interference cancellation tree algorithm (SICTA) which uses a successive interference cancellation (SIC) mechanism, was designed and shown to achieve an MST as high as 0.693 [17]. The operation of this interference cancellation (IC) mechanism can be summarized as follows. Consider two signals a and b, where b contains the combination of signals $B_1, \ldots, B_n$. We denote $a - b$ as the interference cancellation operation, which only results in a valid signal if a consists of $B_1, \ldots, B_n$, $A_1, \ldots, A_m$, and has $A_1, \ldots, A_m$ as a result. Thus, when combined with a tree algorithm, which recursively splits each collision into two groups, SIC offers the possibility to obtain the signal of the second group by canceling the signal of the first group from the joint signal, removing the need to assign a slot to the second group. For example, consider a collision of two users and assume both users do not choose the same group, then only one more slot is required to receive both packets successfully as opposed to two in the classical model, thereby significantly improving the channel throughput. SIC was also employed in satellite systems [5], where a packet was transmitted in two different slots, in the hope that at least one copy was received successfully. If so, SIC could cancel out the successful signal in the second slot used by the same user, which in turn might result in additional successes.

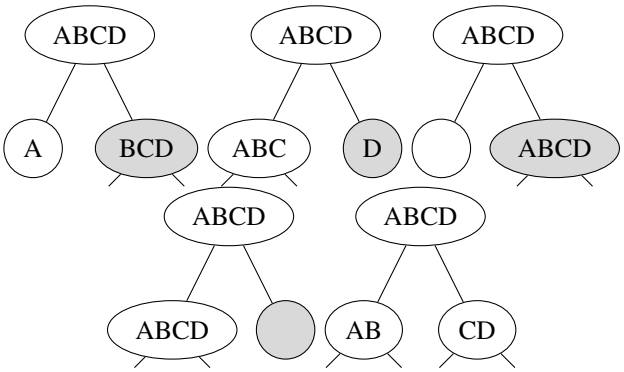


Fig. 1. Overview of the possible collision splits; the gray slots can be skipped, since IC retrieves the right slot by subtracting the left from the parent slot.

Under the standard assumption of Poisson arrivals, the number of new arrivals that can occur in a slot is unbounded. This implies that the maximum number of users involved in a collision under SICTA is also unbounded and therefore SICTA requires a (theoretically) unbounded amount of memory locations for storing signals. In [12] we introduced a novel tree algorithm using SIC that requires the storage of at most one signal at a time, achieving minimal memory requirements. This single memory location stores the signal of the previous slot if it held a conflict. When this conflict splits into two groups it thus suffices to transmit the signal of the left group to obtain the signal of the users in the right group using SIC and therefore the corresponding slot of the latter group can be skipped. We will refer to the slot associated to the joint signal of the group of users that selected the first (second) group as the left (right) slot. If both groups consist of two or more users, the signal of the right group cannot be stored as the memory location is used by the left group. If on the other hand either the left or right group consists of less than two

Conditions		Actions		
Current slot	IC	Feedback	Updated ss	Store in cache
Collision	$ss = cs$	C/S	cs	
Collision	$ss - cs$ valid	C/S	cs	
Collision	$ss = \emptyset$	C	cs	
Collision	otherwise	C/C	cs	$ss - cs$
Success	$ss - cs$ valid	S/S	cache/$\emptyset$	
Success	$ss = \emptyset$	S	$\emptyset$	
Success	otherwise	S/C	$ss - cs$	
Idle	$ss = \emptyset$	S	$\emptyset$	
Idle	otherwise	S/C	ss	

TABLE I

ALL POSSIBLE SCENARIOS A RECEIVER MAY ENCOUNTER; DEPENDING ON EACH STATE, DIFFERENT FEEDBACK IS PROVIDED TO THE USERS, WHILE UPDATING THE SAVED SIGNAL ss AND STORED SIGNALS IN THE CACHE. cs HOLDS THE CURRENT SIGNAL, AS IN [12]. POSSIBLE UPDATES TO THE CACHE ARE DETERMINED BY THE REPLACEMENT POLICY AND ARE NOT DISCUSSED IN THIS TABLE.

(a) User table for cc

	S	S/S	S/C	C	C/S	C/C $fc > 0$	C/C $fc \leq 0$
$cc = 0$	-1	-1	-1	$0/1$	$0/1$	$0/1$	$0/1$
$cc = 1$	0	-1	$0/1$	2	-1	$2/3$	2
$cc \geq 2$	$cc - 1$	$cc - 2$	cc	$cc + 1$	cc	$cc + 2$	$cc + 1$

(b) User table for fc

	S/S	C/C
PER	$min(k - 1, fc + 1)$	$fc - 1$
LRU	$min(k - 1, fc + 1)$	$max(0, fc - 1)$ if $cc \geq 3$

TABLE II

USER ACTION TABLE FOR THE PROPOSED ALGORITHMS, DEPENDING ON THE FEEDBACK. WHEN A COIN FLIP IS USED TO DECIDE WHETHER cc IS UPDATED TO x OR y, WE DENOTE THIS AS x/y. cc REPRESENTS THE COUNTER VALUE AS IN [12]. WHEN $cc = 0$, A USER IS ALLOWED TO TRANSMIT, $cc < 0$ IMPLIES THE USER WAS SUCCESSFUL, WHILE fc TRACKS THE NUMBER OF FREE CACHE LOCATIONS, SO INITIALLY $fc = k - 1$ AND fc IS UPDATED *after* UPDATING cc.

users, we can skip the right slot as depicted in Figure 1.

A free access variant of this algorithm with an MST of 0.5698 was developed in [12], where free access implies that new users are allowed to transmit their packet at the start of the next slot. As random access algorithms with free access typically achieve lower MSTs, we also described a blocked access variant (i.e., where all the new arrivals are blocked until the ongoing conflict is resolved) in [12] and its MST of 0.6620 for windowed access and 0.5545 for gated access was derived in [11], [2]. In windowed access, the users are grouped into sets based on their arrival time, and the users are given access to the channel one group at a time. Typically, the time axis is partitioned into intervals of length α_0 and the i-th set is formed by the users in the i-th interval. If a group is granted access to the channel before the end of its corresponding interval, the interval is shortened and the subsequent intervals are shifted back such that the start of the next interval coincides with the current time epoch. As such gated access corresponds to having $\alpha_0 = \infty$, as all waiting stations are allowed to take part in the next CRP whenever a conflict gets resolved. Remark that SICTA employs gated access.

An unanswered question is how many memory locations are necessary to achieve an MST close to the performance of the SICTA algorithm, which requires an unbounded number of signal memory locations. To address this question, we will describe and analyze two novel tree algorithms, based on the 0.6620 algorithm in Section II. Section III provides numerical results for both algorithms.

II. K-MEMORY TREE ALGORITHMS

Suppose that we have $k > 1$ memory locations. The idea is to use the first memory location in an identical manner as in the 0.6620 algorithm (see [12], [11] and Figure 1), to construct a windowed access tree algorithm with IC. The remaining $c = k-1$ memory locations will be used as a cache. Only collisions for which both the left and right slot contain two or more users, make use of the cache memory. Such an event will be denoted as C/C, to reflect that we have two collisions. The collision in the right slot has to be resolved after the left one, as in any tree algorithm, however instead of transmitting its corresponding signal, we can retrieve it from the cache (unless it got replaced by another signal).

As the limited size of the cache cannot store all collision signals, some right slots are no longer be skipped (as in SICTA). Whether a right slot will be skipped or not, will depend on the replacement policy used in the cache. We will

investigate two policies. In the first, persistent policy, signals remain in the cache until they are used, meaning a signal is never overwritten as long as it remains unused. Thus a cache location remains occupied until the tree algorithm starts with the resolution of the conflict stored in this location. The second policy, a least recently used (LRU) based policy, will always store new signals in the cache, possibly at the expense of removing an older signal. Notice, once a cached signal is reused, there is no further use for it, so it can be removed from the cache.

A. Persistent Cache Policy

In the persistent cache policy case, signals will remain cached until they are used. As such, the right slot of a C/C is only stored if some of the cache locations are still unoccupied. A possible algorithmic description for this collision resolution algorithm can be found in Tables I and II, describing the actions taken by the receiver and users, respectively. Similar to other tree algorithms, each user maintains a counter (cc), which controls the time a user has to wait to (re-)transmit, with $cc = 0$ meaning he is allowed to transmit in the next slot and $cc < 0$ indicates a successful transmission. This number is updated based on the feedback provided by the receiver, which can attain six possible values. Feedback values S and C correspond to the classical no-collision, collision feedback, and the operation on cc is identical to standard tree algorithms. When a collision in the right slot is encountered (S/C or C/C), the slot of these users (with $cc = 1$) is skipped immediately (unless there was no free location in the cache in case of a C/C). A success or idle in the right slot (S/S or C/S) considers the possible user with $cc = 1$ as received, by decreasing its counter to -1. We require a second counter, fc, to be maintained by each user, which can be regarded as the number of free cache locations (although it can also become negative). When a C/C feedback is issued and the cache is full, the users in the right slot are not split and the slot is not skipped. Thus, after each C/C feedback, fc is decremented by one (possibly making fc negative); when an S/S feedback is issued (meaning we have two successes), a cached signal is reused (if $fc \geq 0$) and fc is incremented by one. Finally, ss stores the content of the previous slot, if it was a conflict, identical to [12] and [11]. Figure 2 illustrates this procedure, as it resolves a collision of eight users, corresponding to the splitting tree of Figure 3. We remark that as the counter fc is identical for all users, we could store its value in the receiver; who can signal a full cache by sending a C feedback instead of a C/C.

Define $L_{N,c}^{PER}$ as the average time required to resolve a collision of N users, with c available cache signal positions, hence we need $L_{N,k-1}^{PER}$ as $c = k-1$. This algorithm naturally leads to the following approach to analyze its performance. The case where we have $c = 0$ cache positions is identical in operation as the 0.6620 stack based algorithm (see [12], [11]). For the other cases, several situations require the use of one cache location, reducing the effective number of free cache locations by one. However, as we only store signals of right slots, the corresponding cache locations can be reused again as soon as the left slot is resolved. Whether the right slot is skipped can be determined as soon as the left slot is transmitted. Indeed, at this point, the IC operation can determine whether or not both left and right slot contain a collision and thus require a cache location. Only if this cache location is available, we skip the right slot. Thus, as long as we have one or more cache positions available, we can skip the right slot:

$$L_{N,c}^{PER} = \begin{cases} L_N & c = 0, \\ 1 & c > 0, N < 2, \\ \sum_{i=2}^{N-2} \binom{N}{i} p^i (1-p)^{N-i} \left(L_{i,c-1}^{PER} + L_{N-i,c}^{PER}\right) & \\ +(p^N + (1-p)^N)(L_{0,c}^{PER} + L_{N,c}^{PER}) & \\ +N(p(1-p)^{N-1} & \\ +\delta_{N>2} p^{N-1}(1-p))(L_{1,c}^{PER} + L_{N-1,c}^{PER}) & \\ & otherwise, \end{cases} \tag{1}$$

with $L_0 = L_1 = 1$, and L_N for $N > 1$ as defined in [12], [11]:

$$\begin{aligned} L_N =& 1 + \sum_{i=0}^{N} \binom{N}{i} p^i (1-p)^{N-i} (L_i + L_{N-i}) \\ & - p^N - (1-p)^N - Np(1-p)^{N-1} \\ & - Np^{N-1}(1-p) + \delta_{N=2} Np(1-p). \end{aligned} \tag{2}$$

Notice, the 1-term in Equation (2) has disappeared in Equation (1) for the case where $c > 0$ and $N \geq 2$, as the right slot can now be skipped. The other terms in this expression represent the cases which differ from the default behavior (i.e., for $i = 0, 1, N-1, N$).

B. LRU Cache Policy

The LRU policy employs a more flexible strategy by caching *all* signals, and discarding older signals. To be more precise, when a C/C occurs with a full cache, it will discard the oldest signal to make room for this new signal. Thus, the decision to skip a right slot can be reverted later, if too many other C/C's occur. The result is that this policy prevents a slot close to the root from occupying the cache, when several other slots further down the tree can benefit from this signal cache location.

An algorithmic description is also given by Tables I and II. The changes at the receiver side are located within the cache operation, which is not discussed in Table I. On the user side, the value of fc is no longer identical for all users and is initialized at $k-1$. As indicated by Table II every right slot is therefore initially split after a C/C. However, when $k-1$ additional C/C's follow before the right slot occurs, the split is undone (i.e., both groups are merged again). The counter fc keeps track of the additional number of C/C's that occurred and is updated only by users with $cc \geq 3$ when a C/C occurs. This latter rule implies that the users in the right and left slot created by the initial split obtain a different fc value ($k-2$ and $k-1$, respectively), which implies that when $k-1$ more C/C's have occurred both groups update their cc value differently and both groups are merged.

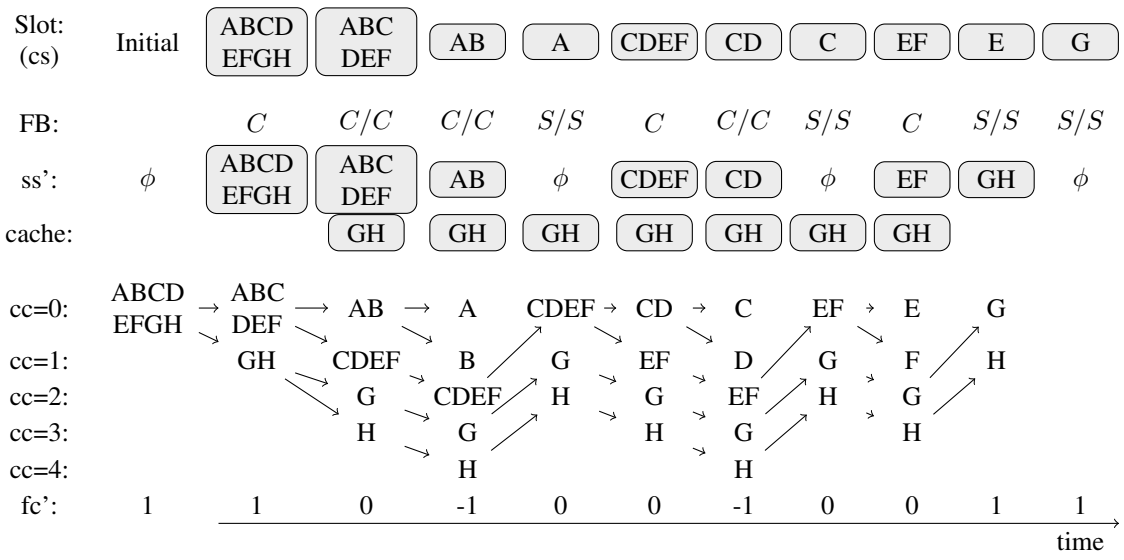


Fig. 2. Example of the persistent cache policy with $k = 2$ memory locations, solving the collision in Figure 3(a). The first memory location is used for ss, the remaining location for cached signals. After each slot, the receiver applies the rules of Table I, producing feedback FB. Based on this feedback, the users follow Table II to update their counter values cc and fc. ss' represents the new saved signal.

Before we can analyze the average time $L_{N,c}^{LRU}$ required to resolve a collision of N signals, with c cache locations and an LRU policy, we need to determine the probability that the right slot cannot be skipped (i.e., is reverted in the implementation). For this purpose we define $R_{N,c}$ as the probability that during the resolution of N users at least c memory locations are required at some point in time. Remark that in case of a collision between two users one does not require an additional memory location.

$$R_{N,c} = \begin{cases} 1 & c = 0, \\ 0 & c > 0, N \leq 2, \\ \sum_{i=2}^{N-2} \Big[\binom{N}{i} p^i (1-p)^{N-i} \cdot \\ (1 - (1 - R_{i,c-1})(1 - R_{N-i,c})) \Big] \\ +(p^N + (1-p)^N) R_{N,c} + N(p^{N-1}(1-p) \\ +p(1-p)^{N-1}) R_{N-1,c} \\ otherwise. \end{cases} \tag{3}$$

Now, computing $L_{N,c}^{LRU}$ is straightforward, assuming that each right slot can be skipped, except when the resolution of the left slot requires all available memory locations and one location is also needed for the right slot (i.e., there are at least two users). Remark that $L_{N,c}^{LRU}$ is not recursively defined as a function of $L_{N,c-1}^{LRU}$:

$$L_{N,c}^{LRU} = \begin{cases} 1 & N < 2, \\ \sum_{i=0}^{N} \binom{N}{i} p^i (1-p)^{N-i} (L_{i,c}^{LRU} \\ +L_{N-i,c}^{LRU} + \delta_{i \geq 2} \delta_{N-i \geq 2} R_{i,c}) & otherwise. \end{cases} \tag{4}$$

III. Numerical Results

For a windowed access algorithm stability corresponds to stating that the average length $\bar{L}$ of a CRP must be less than the window length α_0 [3, Section 4.3]. By multiplying both sides with λ, we can rewrite this as

$$\lambda < \frac{\lambda \alpha_0}{\sum_{N=0}^{\infty} L_N^* \frac{(\lambda \alpha_0)^N e^{-\lambda \alpha_0}}{N!}},$$

such that the right hand side becomes a function of $\lambda \alpha_0$, with λ the arrival rate of the Poisson process and $*$ either LRU or PER. Numerically maximizing this function produces the results as presented in Table III. A (close-to) optimal p was obtained by numerical maximization.

When operating under the persistent cache policy, a single

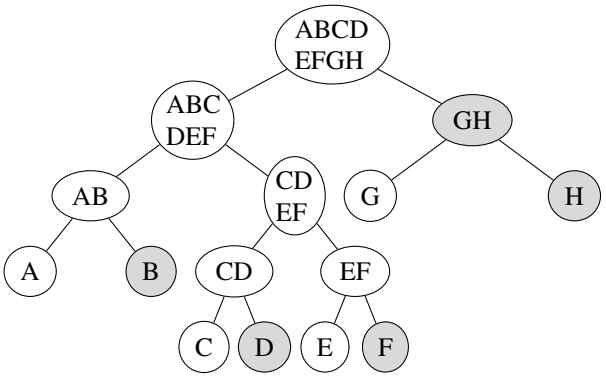
(a) $p = 0.5$

	MST	
k	Persistent	LRU
1	0.66204	0.66204
2	0.68898	0.68903
3	0.69277	0.69277
4	0.69312	0.69312
5	0.69314	0.69314
∞	0.69314	0.69314

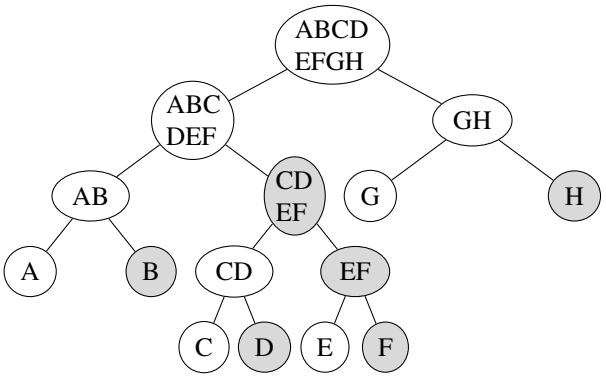
(b) p close-to optimal

	MST		
k	Persistent	LRU	Optimal p
1	0.66204	0.66204	0.5
2	0.68900	0.68904	0.4972
3	0.69277	0.69278	0.4993
4	0.69312	0.69312	0.4999
5	0.69314	0.69314	0.5
∞	0.69314	0.69314	0.5

TABLE III
MSTs FOR THE k-SIGNAL MEMORY TREE ALGORITHM.



(a) Persistent policy



(b) LRU policy

Fig. 3. Smallest example where the LRU and persistent cache policy differ in terms of the number of skipped slots, with $k = 2$ memory locations. The skipped slots are marked gray.

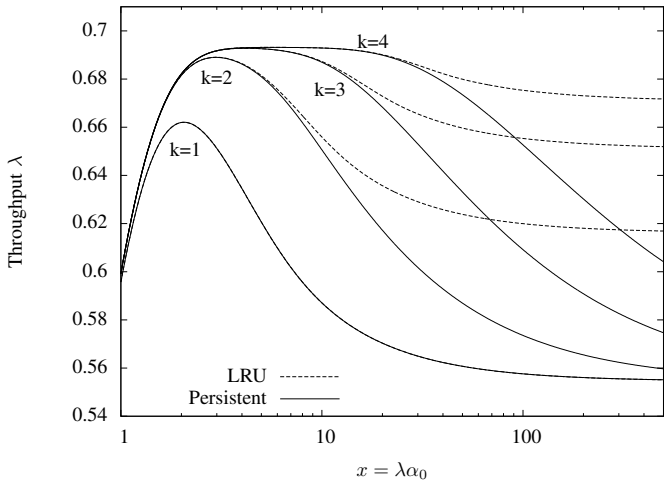


Fig. 4. Throughput for general k-memory windowed access algorithms, with $p = 0.5$. x represents the average number of users in a length α_0 window. Gated access corresponds to $x = \infty$.

slot typically occupies the cache longer than in the LRU case; intuitively, the LRU policy should therefore achieve a higher MST. This is confirmed by Table III, but the gain is marginal because the operation of both algorithms only differs when there are at least six initial users (when $k = 2$). In terms of the *number* of skipped slots, at least eight initial colliding users are required, see Figure 3 for an example. Since windows having that many participants are rare, the influence on the MST is small.

By choosing a (close-to) optimal p (i.e., the probability of choosing the left slot), the MSTs can be marginally increased. This asymmetrical optimal choice for p can be explained by noting that a right slot has more cache locations available during its resolution, thus having on average slightly more users in the right slot is optimal.

The effect of the window length α_0 is shown in Figure 4. We notice that the optimal number of arrivals (i.e., window size α_0) increases as a function of k. This seems obvious as more memory locations only become beneficial if larger initial collisions occur and SICTA is optimal for gated access. We also see that the difference between the LRU and the persistent policy becomes more pronounced for larger window lengths α_0. The persistent policy typically skips slots close to the root, whereas the LRU policy skips mostly slots near the leafs. As α_0 approaches infinity (i.e., gated access) the effect of the having additional memory locations vanishes for the persistent policy. Intuitively, its limiting behavior should coincide with the single memory location algorithm that has an MST of 0.5545 for gated access [11], [2]; simulation runs support this intuition. As the LRU policy is mostly effective near the leaf nodes, we do not expect convergence to the same point for all k, as Figure 4 seems to confirm.

IV. Conclusion

Two variations of the SICTA algorithm that uses only k signal memory locations were defined which differ only in the replacement policy of the cache. The persistent policy, which is less complex to design and analyze, has only a marginally lower MST than the more complex LRU based policy. As k increases, we can obtain throughputs arbitrarily close to the SICTA algorithm, the MST of which equals $ln(2) = 0.6931$. Indeed for $k = \infty$ both algorithms coincide. The results also show that choosing k around 4 is already sufficient to obtain an MST $\in [0.693100, 0.693147]$.

References

[1] D. Aldous. Ultimate instability of exponential back-off protocol for acknowledgement-based transmission control of random access communication channels. *IEEE Trans. Inf. Theory*, IT-33:219–223, 1987.

[2] Sergey Andreev, Eugeny Pustovalov, and Andrey Turlikov. Sicta modifications with single memory location and resistant to cancellation errors. In *NEW2AN '08 / ruSMART '08: Proceedings of the 8th international conference, NEW2AN and 1st Russian Conference on Smart Spaces, ruSMART on Next Generation Teletraffic and Wired/Wireless Advanced Networking*, pages 13–24, Berlin, Heidelberg, 2008. Springer-Verlag.

[3] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall Int., Inc., 1992.

[4] J.I. Capetanakis. Tree algorithms for packet broadcast channels. *IEEE Trans. Inf. Theory*, 25(5):319–329, 1979.

[5] E. Casini, R. De Gaudenzi, and O. del Rio Herrero. Contention resolution diversity slotted ALOHA (CRDSA): An enhanced random

access scheme for satellite access packet networks. *IEEE Trans. Wireless Commun.*, 6:1408–1419, April 2007.

[6] A. Ephremides and B. Hajek. Information theory and communication networks: an unconsummated union. *IEEE Trans. Inf. Theory*, 44(6):2416–2434, October 1998.

[7] N. Golmie, F. Mouveaux, and D. Su. A comparison of MAC protocols for hybric fiber/coax networks: IEEE 802.14 vs. MCNS. In *Proceedings of the 16th International Conference on Communications*, pages 266–272, Vancouver, Canada, June 1999.

[8] N. Golmie, Y. Saintillan, and D.H. Su. A review of contention resolution algorithms for IEEE 802.14 networks. *IEEE Communication Surveys*, 2(1), 1999.

[9] D. Kazakos, L.F. Merakos, and H. Deliç. Random multiple access algorithms using a control mini-slot. *IEEE Trans. Comput.*, 46(4):473–476, 1997.

[10] S. Khanna, S. Sarkar, and I. Shin. An energy measurement based collision resolution protocol. In *Proceedings of the 18-th ITC conference*, Berlin Germany, 2003.

[11] G. T. Peeters and B. Van Houdt. Improved high maximum stable throughput fcfs tree algorithms with interference cancellation. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2008.*, OCT 2008.

[12] G. T. Peeters, B. Van Houdt, and C. Blondia. A multiaccess tree algorithm with free access, interference cancellation and single signal memory requirements. *Performance Evaluation*, 64:1041–1052, 2007.

[13] G.C. Polyzos and M.L. Molle. Performance analysis of finite nonhomogeneous population tree conflict resolution algorithms using constant size window access. *IEEE Trans. Commun.*, 35(11):1124–1138, 1987.

[14] D. Towsley and P.O. Vales. Announced arrival random access protocols. *IEEE Trans. Commun.*, COM-35(5):513–521, May 1987.

[15] B. S. Tsybakov and V.A. Mikhailov. Free synchronous packet access in a broadcast channel with feedback. *Problemy Peredachi Informatsii*, 14(4):32–59, 1978.

[16] Y. Yu and G. B. Giannakis. SICTA: a 0.693 contention tree algorithm using successive interference cancellation. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Miami (USA)*, pages 1908–1916, March 2005.

[17] Y. Yu and G. B. Giannakis. High-throughput random access using successive interference cancellation in a tree algorithm. *IEEE Trans. Inf. Theory*, 53(12):4628–4639, Dec. 2007.