# A Distributed Scheduling Algorithm for Quality of Service Support in Multiaccess Networks

Craig Barrack Kai-Yeung Siu*

## Abstract

This paper presents a distributed scheduling algorithm for the support of QoS in multiaccess networks. Unlike most contention-based multiaccess protocols which offer no QoS guarantee and suffer the problems of fairness and low throughput at high load, our algorithm provides fairness and bandwidth reservation in an integrated services environment and at the same time achieves high throughput. Moreover, while most reservation-based multiaccess protocols require a centralized scheduler and a separate channel for arbitration, our algorithm is truly distributed in the sense that network nodes coordinate their transmissions only via headers in the packets. We derive theoretical bounds illustrating how our distributed algorithm approximates the optimal centralized algorithm. Simulation results are also presented to justify our claims.

## 1 Introduction

One of the most familiar problems in the field of data communication networks is the problem of allocating a multiaccess channel among a set of nodes competing for its use. Multiaccess media form the basis for local area networks, metropolitan area networks, satellite networks, and radio networks [4], [6], making the problem fundamental. In this *multiaccess communication problem*, we have $m$ nodes $1, 2, \ldots, m$, each connected to a multiaccess channel $C$. A data packet that arrives at node $i$ is inserted into a first-in-first-out (FIFO) queue at $i$, where the packet waits until it gains access to $C$, so that it can be transmitted to its destination. We assume that the time required to transmit a data packet is the same for all packets. Because the channel is shared, every node can read the data sent by every other node. However, if multiple nodes attempt to transmit packets simultaneously, none of the attempts succeed. We say that a distributed algorithm solves the multiaccess communication problem if it guarantees eventual transmission of all packets that enter the system.

In our multiaccess communication model, any communication among users takes place along the single channel $C$. In particular, there is no separate control channel, and no centralized scheduler with global knowledge coordinating the users. Knowledge about the state of the queues is distributed among the nodes of the system. Furthermore, we assume zero propagation delay, and that the node and channel are fault-free.[1]

Though nearly any strategy one might think of can be said to "solve the multiaccess communication problem," our concern is algorithm performance:

**Throughput** How much data is transmitted in a given interval of time?

**Delay** How long does a packet have to wait for transmission?

Strategies for addressing the multiaccess communication problem have traditionally fallen into two categories. Many proposed strategies have been *contention-based* schemes, in which nodes greedily attempt to send new packets immediately; for example, see [7], [10], [11]. Such strategies generally require backlogged nodes to attempt and reattempt transmission randomly to prevent deadlock. Under heavy loading, collisions among contending packets reduce system throughput and increase packet delay.

*C. Barrack is currently with Vertex Networks in Irvine, California. This work was done while C. Barrack was with MIT. Email: cbarrack@alum.mit.edu. K.-Y. Siu is with the d'Arbeloff Laboratory for Information Systems and Technology, Massachusetts Institute of Technology. Please send all future correspondence to K.-Y. Siu. Email: siu@sunny.mit.edu.

[1] We refer the reader to our technical report [2], which discusses extensions of our work when these assumptions are relaxed.

Other proposed strategies have been *reservation-based*, in which nodes receive reserved intervals for channel use; for example, see [3], [5]. Here, in order to reserve a time slot for use of the channel, a node needs to communicate its need to the remaining nodes. Generally, such systems alternate short *reservation intervals* giving each node an opportunity to reserve a time slot, with longer *data intervals* where actual packet transmissions take place. Unable to dynamically adjust to traffic conditions, existing reservation-based schemes typically follow a strict cyclic order in serving the users. In a good reservation-based scheme, if the channel is idle, only a small fraction of the time required to transmit a packet need be wasted. This small period of time is called an *idle minislot*.

This paper will present a new distributed scheduling algorithm for quality of service support in multiaccess networks. As we will see, our algorithm's performance is nearly identical to that of an optimal, centralized algorithm under moderately bursty traffic conditions. Furthermore, our algorithm provides fairness and bandwidth reservation in an integrated services environment, while achieving high throughput.

## 2 The *GlobalTime* algorithm

Our framework for the multiaccess communication problem as presented in Section 1 is on-line and distributed; users must decide in real time whether to attempt packet transmissions, and must do so without the coordinative assistance of a centralized scheduler. To gain insight into this problem, we examine the related offline, centralized problem. It turns out that the multiaccess communication problem in its offline, centralized incarnation is just the machine-job scheduling problem, a well-researched topic in the dynamic programming and discrete optimization literature. It is known that the optimal solution to the machine-job scheduling problem is that completing the jobs in order of increasing *release times* – the time a job becomes available – minimizes worst case waiting time [1]. This result provides motivation for a multiaccess communication algorithm based on machine-job scheduling principles.

### 2.1 Inversions

Machine-job scheduling principles suggest that the more closely the sequence of transmissions generated by a distributed algorithm approximates first-come-first-served (FCFS), the better the algorithm performs with respect to worst case delay. To quantify this notion, we introduce a new concept: the inversion.

**Definition 1** *Given $n$ packets $P_1, P_2, \ldots, P_n$, where packet $P_i$ arrived in the system at time $a_i$ and was transmitted at time $t_i$, we say two packets $P_i$ and $P_j$ are* **inverted** *if $t_i < t_j$ but $a_i > a_j$.*

We apply the concept of inversion to measure worst case delay, a performance metric that is sensitive to choice of algorithm.

Many existing algorithms generate $\Omega(n^2)$ inversions in the worst case, where $n$ is the number of packet transmissions. For example, it is not difficult to prove that the popular gated limited service or "round robin" protocol produces an output with $\Omega(n^2)$ inversions in the worst case. Observe that $\Omega(n^2)$ inversions signifies that on average, each packet is displaced from its optimal FCFS position by a length proportional to that of the entire transmission sequence. We can do better.

### 2.2 The algorithm

An algorithm that can guarantee a constant number of inversions per packet on average is highly desirable. In this section, we produce a new algorithm called *GlobalTime*. As we will see, the *GlobalTime* algorithm generates a sequence of transmissions with fewer than $mn$ inversions in the worst case — at most $m$ per packet — where $m$ is the number of users in the system.

The *GlobalTime* algorithm works as follows:

1. Each node $1, 2, \ldots, m$ has a clock. The nodes' clocks are synchronized and are initialized to 0.

2. When a packet $P$ arrives in the system, it is immediately timestamped with the current time. This timestamp records the arrival time of $P$.

3. Each node $i$ has a set of $m$ states *known-time*$[j]$, one for each node $j$. All the *known-time* states are initialized to 0.

4. At the beginning of a time slot, node $i$ determines the value of $j$ such that *known-time*$[j] \leq$ *known-time*$[k]$ for all $k$. If $i = j$, then node $i$ has gained access to the channel. (Ties are broken according to predetermined rules.)

5. On its turn, node $i$ transmits the first data packet on its queue. In addition to actual data, the transmitted frame also contains piggybacked information — namely, the *timestamp of the next packet* in node $i$'s queue. However, if the transmission has exhausted node $i$'s queue, the algorithm instead inserts the *current time* in this extra field. Finally, if on node $i$'s turn, $i$ has no packets to send at all, then $i$ sends a dummy packet (augmented with the current time, as before).

6. When a packet $P$ is being transmitted from node $i$, all nodes read the timestamp piggybacked on $P$, and then update their local value of *known-time*$[i]$.

The crucial idea behind the *GlobalTime* algorithm is that on their turn, users not only transmit the usual data, but also broadcast the arrival time of the packet waiting next in queue. Because all users maintain a table of the most recent *known-time* declaration by each other user, transmission can proceed in nearly FCFS order.

A perceived problem with piggybacking *known-time* declarations on data packets is that if a node has no data to send initially, it might never get an opportunity to tell the other users when it *does* have a packet to transmit. The *GlobalTime* algorithm avoids this problem by broadcasting the current time in lieu of a real packet's arrival time if a user has no next-in-line packet on its turn. Furthermore, this dummy timestamp is treated no differently from a real timestamp during the execution of *GlobalTime*, ensuring that the idle user will eventually get another turn. On the other hand, because the dummy timestamp carries the current time, we are guaranteed that all real data packets presently in the system will be cleared out before the idle user's next turn.

What is interesting about the *GlobalTime* algorithm is the way we exploit all the information available in the multiaccess channel architecture. In particular, after node $i$ has won the competition for channel access, the other nodes do not simply wait passively for the next round of competition. Instead, all nodes are active at all rounds — if not transmitting actual data, then "snooping the bus" and performing local update operations.

### 2.3 Worst case inversions

In this section, we prove the result we alluded to earlier, that the *GlobalTime* algorithm indeed addresses the issue of eliminating inversions.

**Theorem 1** *In a sequence of $n$ transmissions generated by the* GlobalTime *algorithm, a packet $P$ with arrival time $a_P$ will be transmitted after at most $m-1$ packets with later arrival times.*

**Corollary 2** *A sequence of $n$ transmissions generated by the* GlobalTime *algorithm has $O(mn)$ inversions in the worst case.*

**Proof:** First, we define some notation. Let $a_P$ be the arrival time of packet $P$ at node $X$, and let $\tau_P$ be the value of *known-time*$[X]$ at the time packet $P$ is transmitted. Now consider a packet $i$ that arrives at node $B$. By step 5 of the *GlobalTime* algorithm, clearly $\tau_i \leq a_i$. Next, consider a packet $j$ that is transmitted before packet $i$, but arrived at node $A \neq B$ at time $a_j > a_i$. By step 4 of the *GlobalTime* algorithm, we conclude $\tau_j < \tau_i$.

We now argue that there cannot exist another packet $k$ at node $A$, transmitted between the transmissions of packets $j$ and $i$, such that $a_k > a_i$. If true, the theorem and its corollary follow immediately. Assume for the purpose of contradiction that $a_k > a_i$, but $k$ is transmitted before $i$. If the extra field in packet $j$ contained the arrival time of the next packet in node $A$'s queue, then

$$\tau_k = a_k > a_i \geq \tau_i \ ,$$

which, by step 4 of the *GlobalTime* algorithm, is a contradiction. On the other hand, if the extra field in packet $j$ contained the current time $c$, then

$$\tau_k = c > a_j > a_i \geq \tau_i \ ,$$

again a contradiction. □

### 2.4 Throughput

Another performance-related question about the *GlobalTime* algorithm is the maximum traffic intensity it can sustain while ensuring bounded queues. For example, it is well known that a traditional TDM protocol has a throughput that in the worst case can be $1/m$, where $m$ is the number of users [4].

The presence of dummy packets in the *GlobalTime* algorithm causes one to wonder if a similar low-throughput phenomenon can occur here, if the arriving

traffic is sufficiently tweaked to force the algorithm's worst case behavior. In this section, we argue that this is not the case.

First, we consider a condition satisfied by the *GlobalTime* algorithm, expressed in Lemma 1.

**Lemma 1** *In the* GlobalTime *algorithm, if* known-time$[A] = k$*, and the timestamp* $k$ *is a dummy timestamp recording the last time* $A$ *had a turn, then all packets with arrival time less than* $k$ *will be transmitted before* $A$*'s next turn.*

**Proof:** At the time *known-time*$[A] = k$ is recorded, $k$ must be the largest value in the common *known-time* table. Therefore, all users will get at least one opportunity before $A$'s next turn. It follows that all head-of-line packets waiting at time $k$ will be transmitted before $A$'s next turn. By induction on packet position in queue, any other packet waiting at time $k$ will be announced by the piggybacked timestamp on the packet preceding it before $A$'s next turn. We conclude that all packets in the system at time $k$ will be transmitted before $A$'s next turn. □

To summarize, the *GlobalTime* algorithm satisfies the condition that if user $A$ is idle on its turn, then all packets that had arrived *before* $A$*'s previous idle turn* must have already been transmitted. We now can prove an essential corollary of Lemma 1.

**Corollary 2** *(Lagging Window Property) Let* $t_{\text{idle}}$ *be any time at which user* $A$ *is idle on its turn, and let* $t_{\text{idle}}$ *be the* $k$*th idle slot in the transmission sequence of length* $n$*. Furthermore, let* $t_{\text{lag}}$ *be the time of the* $(k–m)$*th idle slot, where* $m$ *is the number of users in the system. Then at time* $t_{\text{idle}}$*, all packets that arrived in the system before time* $t_{\text{lag}}$ *must have been transmitted.*

**Proof:** Some user $X$ must have generated two of the $m+1$ idle slots between $t_{\text{lag}}$ and $t_{\text{idle}}$, by the pigeonhole principle. Applying Lemma 1 to user $X$ proves the corollary. □

The lagging window property can be used to show that the *GlobalTime* algorithm has a maximum stable throughput of nearly 100%. The intuition is that if the expected time between consecutive idle slots in the transmission sequence is high, then by definition, the throughput is high. On the other hand, if the expected time between consecutive idle slots in the transmission sequence is low, then the expected lagging window size $E[t_{\text{idle}} - t_{\text{lag}}]$ is small; therefore, by Corollary 2 the throughput is again nearly optimal. We refer the reader to our technical report [2] for a detailed proof.

## 3 Simulations

In performing simulations, we ran the *GlobalTime* algorithm on the same set of input traffic as three other canonical reservation-based schemes: gated limited service, gated unlimited service, and exhaustive. All three are TDM-like protocols — that is, users receive turns in cyclic order. When it is node $i$'s turn to transmit, $i$ sends all its packets in both a gated unlimited and an exhaustive service discipline, but only the first of its waiting packets in a gated limited service protocol. The only difference between gated unlimited service and exhaustive is whether or not packets that arrive during node $i$'s turn get served during that same turn; in an exhaustive protocol, they are. For the sake of comparison, the four distributed algorithms were compared against a single queue discipline, in which all data packets enter one FIFO queue, and are served in order.

We concerned ourselves primarily with worst case delay and delay variance, though for the sake of space, we focus on worst case delay here.

### 3.1 Simulating bursty traffic

In practice, real data traffic patterns exhibit bursts of activity followed by long idle periods. In our bursty traffic model, time is divided into busy periods and idle periods independently for each user. During busy periods, arrivals are Poisson distributed with mean arrival rate equal to service rate; during idle periods, no traffic arrives. The length of a busy period is an exponential random variable with given mean burst size. Similarly, the length of an idle period is exponentially distributed.

In this section, we discuss the results of applying bursty traffic to the *GlobalTime* algorithm. In the first simulation, depicted in Figure 1, worst case delay is plotted against system arrival rate for the *GlobalTime* algorithm and three alternative distributed protocols. The simulation is run with 128 nodes and $2^{15}$ time slots. The mean burst size is set to 8. For example, if the channel is 2 Gbps, and the time to transmit a single packet is 2 $\mu$s, then a mean burst size of 8 corresponds to 4 KB.

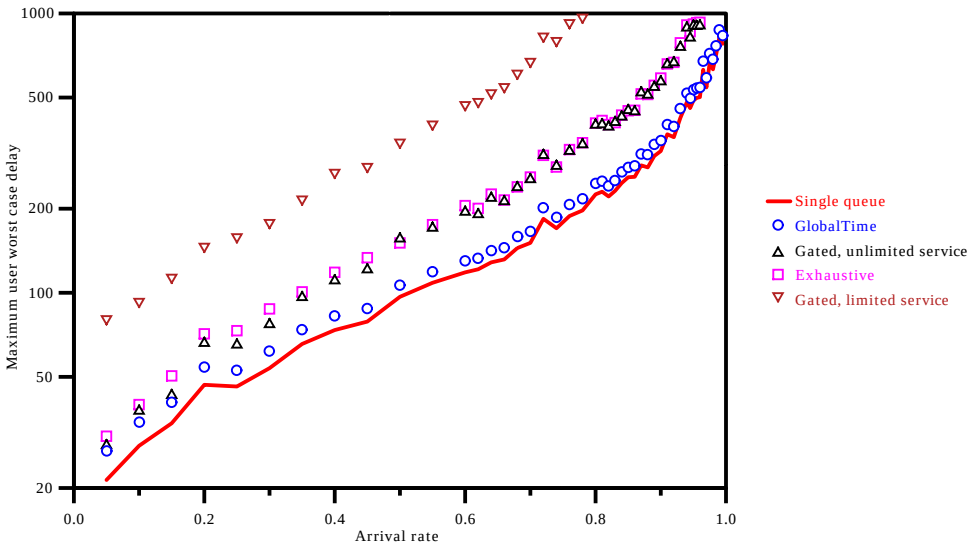


**Figure 1: Worst case delay plotted against arrival rate in each of 5 algorithms.**

Figure 1 demonstrates the superiority of the *GlobalTime* algorithm over reservation-based alternatives with respect to worst case delay. For utilization greater than 30%, the *GlobalTime* algorithm improves the delay bound over the next best strategy by at least 25%. At 80% utilization, this improvement percentage increases to at least 40%. Observe that in Figure 1, the *GlobalTime* curve closely follows the ideal single queue curve over all arrival rates. Indeed, the *GlobalTime* system performs as well as a single server in controlling worst case packet delay.

In a second simulation, we investigate the behavior of the *GlobalTime* algorithm as the burstiness of the input stream varies. For this simulation, we again employ 128 nodes for $2^{15}$ time slots, and we fix channel utilization at 80%. The results of this simulation are plotted in Figure 2.

Table 1 summarizes some of the results of these and other simulations.

# 4 Extending *GlobalTime*

In our discussion so far, we have ignored the human negotiations associated with a multiaccess network, during which customers pay for a fraction of the total bandwidth and receive throughput and delay assurances.

Although in a policed environment the basic *GlobalTime* algorithm maintains quality of service, users do not always "follow the rules" in practice. If one user floods the system with his own packets, all other users will suffer. In this section, we propose an extension of the *GlobalTime* algorithm that safeguards performance guarantees even in the presence of misbehaving users.

## 4.1 Virtual timestamps

Let user $A$ have a reserved rate $\lambda_A$, and let $T_A = 1/\lambda_A$. We can interpret $T_A$ as the expected period between packet arrivals at user $A$, assuming packets enter $A$'s queue at the reserved rate. In the extended *GlobalTime* algorithm, we will regulate the traffic transmitted from node $A$. We imagine *credits* periodically arriving at $A$, one every $T_A$ time units. The incoming credits are stored in a *credit bucket* up to a maximum of $W$; any credits that arrive when the bucket is already full are discarded. Furthermore, we allow the bucket to contain an arbitrarily negative number of credits, signifying a credit shortage. The idea behind this credit scheme is that incoming credits will *validate* waiting packets [8], [9]; without a credit to validate its presence in the system, a packet receives lower priority for transmission.

The circulation of credits is a theoretical fiction in the extended *GlobalTime* algorithm. In reality, user $A$ maintains an additional state $N^A_{\text{credit}}$, where

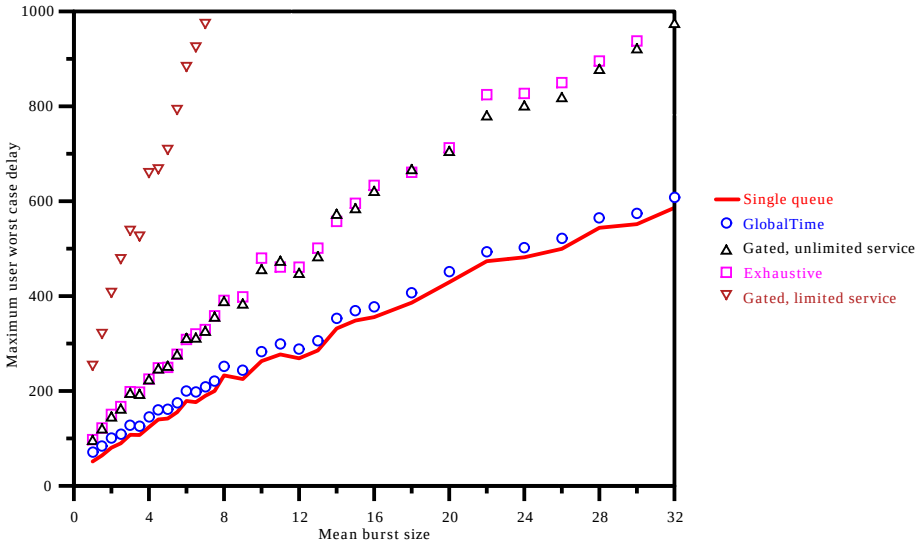


**Figure 2: Worst case delay plotted against mean burst size of arriving traffic.**

| Traffic type | | Improvement in delay standard deviation | Improvement in worst case delay |
|---|---|---|---|
| Poisson | $\lambda = 0.6$ | $< 1\%$ | 4% |
| | $\lambda = 0.8$ | $< 1\%$ | 13% |
| Mean burst size = 8 | $\lambda = 0.6$ | 27% | 41% |
| | $\lambda = 0.8$ | 32% | 47% |

**Table 1: Summary of some simulation results from our work.**

$$-\infty < N^A_{\text{credit}} < W \ .$$

The value of $N^A_{\text{credit}}$ is incremented every $T_A$ time units, up to a maximum of $W$. Moreover, whenever a packet arrives at node $A$, $N^A_{\text{credit}}$ is decremented. In addition to $N^A_{\text{credit}}$, user $A$ maintains a state $t^A_{\text{next}}$, which records the time at which the next virtual credit is set to arrive. Of course, $t^A_{\text{next}}$ is initialized to 0 and is increased by $T_A$ whenever $t^A_{\text{next}}$ is equal to the current time; furthermore, whenever $t^A_{\text{next}}$ is increased, $N^A_{\text{credit}}$ is incremented. The point of all this discussion about implementation is that only two new states, the real number $t_{\text{next}}$ and the integer $N_{\text{credit}}$, are required to maintain the fictional credit bucket scheme locally at each node.

When a packet $P$ arrives at $A$, $N^A_{\text{credit}}$ is decremented, as indicated earlier. Then $P$ receives a virtual timestamp according to the following criterion:

1. If $N^A_{\text{credit}} \geq 0$, then set $P$'s timestamp equal to the current time.

2. If $N^A_{\text{credit}} < 0$, then set $P$'s timestamp equal to $t^A_{\text{next}} - T_A\left(1 + N^A_{\text{credit}}\right)$.

What is a virtual timestamp? Like a real timestamp, it is recorded immediately after a packet arrives in the system. In Case 1, at packet $P$'s arrival time, we have $N^A_{\text{credit}} \geq 1$, indicating that $P$ can be validated by a waiting credit instantaneously upon $P$'s arrival. Therefore, Case 1 sets $P$'s virtual timestamp to $P$'s arrival time, as in the original *GlobalTime* algorithm.

In Case 2, at packet $P$'s arrival time, we have $N^A_{\text{credit}} \leq 0$, expressing that the credit bucket is currently empty. The fact that $P$ cannot be validated immediately by a waiting credit means that $P$ entered $A$'s queue too early. In other words, packet $P$ arrived in $A$'s queue faster than expected, given $A$'s reserved rate. With what time should we stamp packet $P$? Case 2 stamps packet $P$ with the time $P$ *should have arrived* in the system, so as to conform with $A$'s reserved rate. The idea here is that if $P$ arrives too early, we measure $P$'s delay not from its actual arrival time, but from its *virtual arrival time*: the time at which $P$'s corresponding credit arrived at $A$.[2]

In the original *GlobalTime* algorithm, if on user $A$'s turn, $A$'s queue has no next-in-line packet, a dummy timestamp representing the current time is sent. In the extended *GlobalTime* algorithm, dummy timestamps are generated similarly. If $A$ has no next-in-line packet to declare on its turn and $N^A_{\text{credit}} \geq 0$, then $A$ sends the current time as in original *GlobalTime*. On the other hand, if $A$ has no next-in-line packet and $N^A_{\text{credit}} < 0$, then as before, $A$ pretends as if it has a packet that just arrived. Such a packet could not be immediately validated, because the credit bucket is empty, and therefore would be stamped with the time at which the associated credit is scheduled to arrive. Thus, in this case, a dummy timestamp of $t^A_{\text{next}} - T_A\left(1 + N^A_{\text{credit}}\right)$ is sent.

The original *GlobalTime* algorithm minimized the deviation of a transmission sequence from the optimal sequence. Because we have not touched the basic *GlobalTime* framework, this minimization of inversions must remain true, though the optimal sequence is now different. We still want a sequence that minimizes the worst case delay of any packet. However, we now measure delay starting not from a packet's real arrival time, but from its virtual arrival time, the time at which the packet is validated by a credit. With respect to this new optimal sequence, the $O(mn)$ inversions bound holds as before.

### 4.2 User lockout

If we run the original *GlobalTime* algorithm using virtual timestamps in lieu of real timestamps, we may encounter a throughput anomaly. Although node $B$ is backlogged, node $A$ — with no waiting packets — gets a turn every time slot. What could cause such an anomaly? If node $A$ has $N^A_{\text{credit}} > 0$, the next arriving packet will be validated immediately. Therefore, if on its turn $A$ has no packet to transmit, then a dummy timestamp with the current time is sent. In the original *GlobalTime* algorithm, this strategy posed no problem because if the current time was sent, then *known-time*$[i]$ < *known-time*$[A]$ for all other users $i$ immediately after $A$'s turn; that is, the current time served as an upper bound on the values in the *known-time* table. The problem in the extended *GlobalTime* algorithm is that virtual timestamps can exceed the current time. Therefore, even after *known-time*$[A]$ is updated with the current time, it could still be the case that *known-time*$[A]$ < *known-time*$[B]$, and $A$ will receive another turn. In the worst case, this throughput anomaly will continue until the current time "catches up" with the validation time of $B$'s head-of-line packet; meanwhile, valuable channel bandwidth is wasted.

In this section, we suggest a simple solution to the channel utilization problem. If on user $A$'s turn, $A$ has no next-in-line packet and sends a dummy timestamp, then the algorithm *freezes* user $A$ for a period of time $\Delta$.

**Definition 2** *A user $A$ has been* **frozen** *during an execution of the extended* GlobalTime *algorithm if the protocol ignores $A$ when it uses the* known-time *table to decide which user's turn will be next.*

The idea is that if a user has no packets in its queue, it must wait at least time $\Delta$ before getting another turn. Once $A$ has been unfrozen, its virtual timestamp again is active in the $\min_i\{$*known-time*$[i]\}$ phase of the *GlobalTime* algorithm. Because $A$ is frozen for some time, user $B$ is not locked out of the protocol.

How large should we make $\Delta$? The tradeoff between utilization and delay makes the choice situation-specific. The larger we make $\Delta$, the greater the worst case number of inversions in the transmission sequence. The smaller we make $\Delta$, however, the lower the channel utilization in the worst case.

We make an important observation about the choice of $\Delta$.

**Theorem 3** *Let $\beta$ be the length of an idle minislot. Then if $\Delta \leq (m-1)\beta$, user lockout can occur in the worst case.*

**Proof:** If $\Delta \leq (m-1)\beta$, then the freeze time $\Delta$ is not large enough to prevent an execution in which $m-$

[2] The idea of virtual timestamp has been employed in the literature on packet scheduling; see the survey paper [12].

1 of the users each obtain opportunities to transmit one after the other in cyclic fashion, but never have any packets to send. The remaining user may have a backlogged queue, but will be locked out. □

### 4.3 The extended algorithm

To summarize, the extended *GlobalTime* algorithm works as follows:

1. Each node $1, 2, \ldots, m$ has a clock. The nodes' clocks are synchronized and are initialized to 0.

2. Each node $i$ has a predefined reserved rate $\lambda_i$. A state $t^i_{\text{next}}$ is initialized to 0, and is increased by $T_i = 1/\lambda_i$ whenever the clock reads $t^i_{\text{next}}$. Moreover, each node $i$ has an integer state $N^i_{\text{credit}}$, initialized to 0, and incremented whenever the clock reads $t^i_{\text{next}}$.

3. When a packet $P$ arrives in node $i$'s queue, $N^i_{\text{credit}}$ is decremented. Then $P$ is immediately timestamped according to the following criterion:

   - If $N^i_{\text{credit}} \geq 0$, then set $P$'s timestamp equal to the current time.
   - If $N^i_{\text{credit}} < 0$, then set $P$'s timestamp equal to $t^i_{\text{next}} - T_i\left(1 + N^i_{\text{credit}}\right)$.

4. Each node $i$ has a set of $m$ states *known-time*$[j]$, one for each node $j$. Furthermore, each node $i$ has a set of $m$ states *unfreeze-time*$[j]$, one for each node $j$. All the *known-time* and *unfreeze-time* states are initialized to 0.

5. At the beginning of a time slot, let $\mathcal{F}$ be the set of nodes $k$ such that *unfreeze-time*$[k]$ is no greater than the current time. Next, node $i$ determines the value of $j \in \mathcal{F}$ such that *known-time*$[j] \leq$ *known-time*$[k]$ for all $k \in \mathcal{F}$. If $i = j$, then node $i$ has gained access to the channel. (Ties are broken according to predetermined rules. Moreover, if $\mathcal{F} = \emptyset$, then predetermined rules decide which user gains access to the channel.)

6. On its turn, node $i$ transmits the first data packet in its queue. In addition to actual data, the transmitted frame also contains piggybacked information — namely, the *timestamp of the next packet* in node $i$'s queue (if it exists). The transmitted frame also contains a *freeze* bit set to 0.

7. However, if the transmission has exhausted node $i$'s queue, the algorithm instead inserts the current time in this extra field if $N^i_{\text{credit}} \geq 0$, and inserts $t^i_{\text{next}} - T_i\left(1 + N^i_{\text{credit}}\right)$ otherwise. On the other hand, if on node $i$'s turn, $i$ has no packets to send at all, then $i$ sends a dummy packet (augmented with either the current time or $t^i_{\text{next}} - T_i\left(1 + N^i_{\text{credit}}\right)$, as before). In either case, the transmitted frame also contains a *freeze* bit set to 1.

8. When a packet $P$ is being transmitted from node $i$, all nodes read the timestamp piggybacked on $P$, and then update their local value of *known-time*$[i]$. Furthermore, if the *freeze* bit is set to 1, then all nodes update *unfreeze-time*$[i]$ to be $\Delta$ added to the current time, where $\Delta$ is the globally defined *freeze period*.

## 5 Conclusion

In this paper, we have presented a new distributed scheduling algorithm, *GlobalTime*, that supports QoS in multiaccess networks. Our algorithm eliminates the need for a centralized scheduler and a separate control channel by coordinating transmissions via packet headers. Furthermore, the *GlobalTime* algorithm exploits the full power of the multiaccess channel architecture, by actually using the ability of nodes to hear all the information being transmitted on the channel. In fact, our algorithm provides fairness and bandwidth reservation in an integrated services environment, while also achieving high throughput. With moderately bursty traffic, *GlobalTime* has a worst case delay nearly identical to that produced by the optimal, centralized algorithm.

## References


[1] Bachem, A. et al. *Mathematical Programming: The State of the Art*. New York: Springer-Verlag. 1983.

[2] Barrack, C. and Siu, K.-Y. "A Distributed Scheduling Algorithm for Quality of Service Support in Multiaccess Networks." Technical Report. MIT Laboratory for Information Systems and Technology. June 1998.

[3] Barry, R. and others. "All-Optical Network Consortium — Ultrafast TDM Networks." *IEEE Journal on Selected Areas in Communications*, 14:999–1013. 1996.

[4] Bertsekas, D. and Gallager, R. *Data Networks*. New Jersey: Prentice Hall, 271–353. 1992.

[5] Cooper, R. "Queues Served In Cyclic Order: Waiting Times." *Bell Systems Journal,* 49:399–413. 1970.

[6] Gallager, R. "A Perspective on Multiaccess Channels." *IEEE Transactions on Information Theory*, 31:124–42. 1985.

[7] Hajek, B. and Van Loon, T. "Decentralized Dynamic Control of a Multiaccess Broadcast Channel." *IEEE Transactions on Automatic Control*, 27:559–568. 1982.

[8] Kam, A., Siu, K.-Y., and others. "A Cell Switching WDM Broadcast LAN with Bandwidth Guarantee and Fair Access." *IEEE/OSA Journal of Lightwave Technology*, 16:2265–2280. 1998.

[9] Kam, A. and Siu, K.-Y. "Linear Complexity Algorithms for QoS Support in Input-Queued Switches with No Speedup." *IEEE Journal on Selected Areas in Communications*, 17:1040–1056. 1999.

[10] Metcalfe, R. and Boggs, D. "Ethernet: Distributed Packet Switching for Local Computer Networks." *Communications of the ACM*, 395–404. 1976.

[11] Rivest, R. "Network Control By Bayesian Broadcast." Technical Report. MIT Laboratory for Computer Science. 1985.

[12] Zhang, H. "Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks." *Proceedings of the IEEE*, 83:1–23. 1995.