Developing and Refining an Adaptive Token-Passing Strategy

Burkhard Englert * Larry Rudolph † Alex Shvartsman ‡

Abstract

Token rotation algorithms play an important role in distributed computing, to support such activities as mutual exclusion, round-robin scheduling, group membership and group communication protocols. Ring-based protocols maximize throughput in busy systems, but can incur a linear, in the number of processors, delay when a processor needs to obtain a token to perform an operation.

This paper synthesizes new algorithmic techniques for improving performance (responsiveness) of logical ring protocols. The parameterized technique preserves the safety properties of ring protocols and maintains high throughput in busy systems, while reducing the delay in lightly loaded systems from a linear to the logarithmic function in the number of processors. The development in this paper is done using term rewriting systems where our parameterized protocol is developed in a series of safety-preserving refinements of a basic specification.

1 Introduction

Developing adaptive, token-based protocols for mutual exclusion, group communication, and other distributed algorithms, is difficult because of the subtle interplay of correctness and performance concerns. We make use of an imperative-directive protocol design to develop a generic token-based protocol that can give rise to many implementations, each optimized for a unique set of performance characteristics but all having the same safety criteria. The protocols are developed from a high-level description in which the desired properties are basically self evident. A succession of refinements, each of which is easily shown to be as safe as the previous, are presented. Separating correctness from performance, makes it easier to devise complex sets of rules to improve performance without constantly worrying about how they impact correctness.

The final refined protocol results in a most responsive system and is described in just 8 rules. It is a token-based algorithm that combines passing a token around a ring of processors with a logarithmic, binary search strategy. Like the IEEE 802.4 protocol, ring-based schemes provide good throughput under varied loads, stable performance under increasing loads, and ensures fair access with deterministic guarantees. On the other hand, logarithmic, tree-based protocols, provide fair access on average for moderate loads and excellent response when the use is bursty but infrequent. Our adaptive scheme provides the best of both by combining a token rotation protocol that provides good and fair performance for loaded systems with a logarithmic token access for quick response under light to moderate loads.

The general target setting is that of a set of nodes communicating by message passing, where the messages may incur unpredictable delays. Some nodes may generate messages that need to be delivered to all nodes (broadcast/multicast) while maintaing consistent order for message deliveries. Alternatively, the nodes maintain replicated state and and use messages to change the state at the recipients. The state must remain consistent according to some application-specific criteria. Our overall goal is to develop techniques for structuring distributed application systems as abstract components that may be mapped to realistic distributed architectures. The components are designed to implement specific application properties. It is desireable for the properties of the constituent components to be “orthogonal” in the sense that the analysis of the resulting system can be derived from the analysis of the individual components.

Given application requirements, our emerging approach is based on developing the application system whose components use two qualitatively different communication modes:
(1) Components use “expensive” communication services; the object here is to use the guarantees of these services to show the correctness/safety properties.
(2) Components use “cheap” messaging without firm guarantees to shepherd the overall system so as to enhance its performance, e.g., if the system maintains a circulating token, the cheap messages may be used as hints designed to alter the token rotation schedule to divert the token to the nodes with greater needs.

The correctness of the system is shown by considering only the “expensive” messages, while the cheap messages do not affect correctness. In particular, the system remains correct even if no “cheap” message is ever sent. The “cheap” messages are used to show conditional performance property of the system. In general we would

* University of California Los Angeles, Dept. of Mathematics, Los Angeles, CA 90095-1555. Email: englert@math.ucla.edu.

† Massachusetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139. Email: rudolph@lcs.mit.edu.

‡ University of Connecticut, Computer Science and Eng., 191 Auditorium Road, U-155, Storrs, CT 06269. Email: aas@cse.uconn.edu.
The work of this author was in part supported by a NSF CAREER Award 9984778, NSF Grant 9988304 and a grant from AFOSR.

like to readson that if a certain number of "cheap" messages is sent, and the messages are delivered within certain time bounds, then the system can achieve desired performance goals, e.g., the overall distributed computation makes progress, or that it makes progress at a known pace.

In this paper we present one development that uses the approach we have now outlined that results in a family of adaptive token-passing algorithms. Our solution incorporates an interesting duality. When a node requires the token, it can either actively try to find the token, or the owner of a token can actively try to find which node requires it. In either case, combining a rotation strategy with a binary search reduces the number of messages. It achieves a responsiveness, defined as the maximum time between some service request and the satisfaction of some (possibly different) request, of $O(\log N)$ under all loads and is fair.

Our methodology and simple rules provide wide scope for many optimizations. As a consequence of the imperative-directive strategy, two types of message passing mechanisms can indeed be exploited. Messages associated with correctness concerns need to be delivered for the protocol to be correct and the correctness is directly related to the reliability of the messaging system. The messages associated with performance concerns are less important and a lower overhead, less reliable mechanism can be used.

1.1 Related work

Mutual exclusion protocols [10, 16, 18] are very important in enabling cooperative distributed computation. Token-based approaches, such as [3, 5, 7, 8, 9, 13], have been used in designing mutual exclusion and resource management protocols for a long time. These token-based approaches include ring-oriented solutions, which require up to a linear number of messages, and tree-oriented solutions that require the number of messages linear in the diameter of the graph.

An important example of cooperative distributed systems is provided by group communication services (GCS) [6]. GCSs enable processes of a distributed system to operate collectively as a group. Different GCSs offer different guarantees about the order and reliability of message delivery, see the survey [19]. Logical token ring approaches to establish message ordering and to maintain membership have been used in GCSs, e.g., [1].

Another related problem is called mutual search [4]. In this problem, a set of agents distributed in a network need to locate each other by querying the nodes of the network. Here the model can be interpreted as a set of clients (token requests) searching for resources (tokens). (See also the related problems given in the appendix in [4].)

1.2 Paper Organization

The structure of the rest of this paper is as follows. In Section 2 we give the basic definitions, and describe the model of computation and state the main safety property. In Section 3 we present the basic global protocol and its refinements leading to a local history protocol using token passing. In Section 4 we introduce explicit message passing modeling. In Section 5 we study the performance issues, define non-deterministic and binary search protocols, present simulation results and describe some immediate optimizations. We conclude in Section 6 with a discussion and future plans.

2 Definitions, and Description

In this section we describe our model of computation, the formal protocol specification notation, and the problem we will solve.

2.1 A Message-Passing System

We model the distributed computing setting where a finite set of processors communicate by means of message passing and share no common storage. The processors have unique identifiers from a finite set $\mathcal{P}$ and the communication network is a complete graph (e.g., we consider communication at the network layer level of abstraction). The processors and communication are asynchronous. In our safety proofs we assume complete asynchrony and make no assumptions about time or communication delays. To establish the performance results we involve operational reasoning about executions. In assessing the performance of our protocols, we assume that the communication occurs within bounded delays and that local events take negligible time.

2.2 Term Rewriting Systems

We use Term Rewriting Systems (TRS's) to define our protocol. We previously successfully used TRS's to model microarchitectures [2] and to specify cache coherence protocols [17]. A TRS $\mathcal{T} = (\Sigma, R)$ consists of a set of terms Σ and a set of rewriting rules $\mathcal{R}$. The terms represent system states and the rules specify state transitions. The general structure of rewriting rules is as follows:

$$s_1 \;\rightarrow\; s_2 \quad (\textbf{if}\, p(s_1))$$

where s_1 and s_2 are terms and $p(s_1)$ is an optional predicate about s_1.

A rule can be used to rewrite a term if its left-hand-side pattern matches the term or one of its subterms, and the corresponding predicate, if any, is true. If several rules are applicable, then any one of them may be applied. If no rule is applicable, then the term cannot be rewritten any further. A rewriting strategy can be used to specify which rule among the applicable rules should be applied at each rewriting step.
Notation: It is important to distinguish between variables and constants while pattern matching. A variable matches any expression while a constant matches only itself. We will follow the convention where variables and constants are represented by identifiers that consist of English and Greek letter, respectively. Identifiers with upper case letters are variables, but usually represent sets. We use '$-$' the wild-card term as a placeholder for any possible term. We use '$|$' as a connective to indicate that ordering does not matter (i.e., '$|$' is associative and commutative).

Additional notation used throughout this paper is given in Figure 1.

- The set of processors is $\mathcal{P}$.
- Given a cycle graph $(\mathcal{P}, \mathcal{E})$ and $x \in \mathcal{P}$, we define x^{+1} to be y such that $(x, y) \in \mathcal{E}$. In the same vein, x^{+2} is $(x^{+1})^{+1}$, x^{+n} is the n^{th} successor of x, and x^{-n} is such y that $y^{+n} = x$.
- The notation $\|_{p(j)} x(j)$ denotes an unordered list of terms $x(j)$ such that for each j the predicate $p(j)$ is satisfied. E.g., $\|_{j\in\{1,2,3\}} j$ stands for $1|2|3$.
- $\oplus$ is the append operator.
- We define ϕ_x to be a unique symbol for each $x \in \mathcal{P}$. It identifies x and it serves as the left identity for $\oplus$; we use it to denote an empty token request for x.
- We define τ_x to a unique symbol for each $x \in \mathcal{P}$. We use τ_x to denote a token trap set on behalf of processor x.
- Let A and B be (ordered) sequences over $\mathcal{C} \cup \mathcal{D}$. We define the relation '$\subset$' so that $A \subset B$ when A is a prefix sequence of B, and we define the relation '$\subset_{\mathcal{C}}$' such that $A \subset_{\mathcal{C}} B$ when A is a prefix sequence of B once they are projected onto the elements of $\mathcal{C}$.

Figure 1: Definitions and brief explanation of additional notation.

2.3 A High Level Description

Given a set of fully interconnected nodes, at any time, a node may wish to obtain an exclusive possession of a broadcast medium in order to multicast to all nodes, or to acquire exclusive access to some shared resource. We wish to develop a family of token protocols in which all the nodes broadcast messages or use an exclusive resource, in the same global order. In the following definition we formally state the safety property required of our protocols in terms of *histories*. A history is simply a sequence of events that are observed locally. A node may also receive a history in a message from another node, and it can append a local event to the history. Our systems will be described from a global point of view. Hence to ensure safety we need to ensure that all local events at a node are captured by our system description, our set of rules. We formalize this notion in the following definition:

Definition 1 *A protocol is said to* satisfy the prefix property *if each node's individual history is a prefix of the global history.*

3 Basic Protocols

A high level global specification is first presented. It is then refined to eliminate any global state or instantaneous actions.

3.1 Global Specification – System S

Consider the most basic scenario. At any time, there is a set of nodes with data to broadcast. For example, if d_x is the data to be broadcast by node x, then let the set Q consist of pairs (x, d_x) for each node x that wants to broadcast.

Our basic system, System S, starts with the set Q as defined above and empty history. Formally, Q is initialized to N pairs (x, ϕ_x), one for each node x, where ϕ_x is a distinguished no-message symbol for each node. There are two rules in System S: one that represents a node wishing to broadcast and another that records the fact that a node is broadcasting. The latter consists of a message from some node x being appended to the history log of all broadcast messages. That is, this rule removes some pair from Q and appends the data to a history log of messages, H. Figure 2 gives a purely symbolic, precise description. We note that rule 2 ensures that System S satisfies the prefix property. From now on we will use this fact to obtain the safety property for more refined systems.

3.2 Local Histories

Each node maintains its own local history record, P. We give a new specification, System S1, in which the global history record is copied to local history records. A node may perform this copy at any time – it is a performance issue as to exactly when the copy should happen. In fact the nodes can perform a copy in any order and at any time.

System S1 consists of three rules, see Figure 3, the first two are essentially the same as in System S. The third rule, which can be invoked at any time, copies the global history to some node's local, prefix history

Lemma 1 *S1 satisfies the prefix property.*

Proof sketch: Given two states in S1, C and D, such that there is a path from C to D, one can map each state in S1 to a state in S under a mapping M and it is easy to show that there is a path from M(C) to M(D) in S. Hence since S satisfies the prefix property, S1 does. The mapping is trivial, just ignore the values of P. □

Exactly how and when a node updates its prefix history can be established either with additional rules or by refining the current rules. Although this will limit the number of possible states that the system can enter, it will not violate the correctness properties.

3.3 Token Passing

The broadcast rule can be restricted to the time when a node has the token. The transitions and reachable states of such a system, like the one represented as System Token in Figure 4, are clearly a proper subset of the transitions and

System S

0	*Initial state* (Q, H)	$:$	$(\|_{x\in\mathcal{P}}(x,\phi_x),$	$\emptyset)$
1	$(Q\|(x,d_x),\quad -)$	$\rightarrow$	$(Q\|(x,d_x \oplus new_x),$	$-)$
2	$(Q\|(x,d_x),\quad H)$	$\rightarrow$	$(Q,$	$H \oplus d_x)$

Figure 2: System S, The base, abstract protocol. When a node wishes to broadcast, it adds a new datum to Q. H is the global history of the broadcasts. It is an ordered set.

System S1

0	*Initial state* (Q, H, P)	$:$	$(\|_{x\in\mathcal{P}}(x,\phi_x)$	$,\quad \emptyset\ ,$	$\|_{x\in\mathcal{P}}(x,\emptyset))$
1	$(Q\|(x,d_x)\ ,\ -\ ,\quad -)$	$\rightarrow$	$(Q\|(x,d_x \oplus new_x)\ ,$	$-\ ,$	$-)$
2	$(Q\|(x,d_x)\ ,H\ ,\quad -)$	$\rightarrow$	$(Q$	$,H \oplus d_x\ ,$	$-)$
3	$(-\quad ,H\ ,P\|(y,-))$	$\rightarrow$	$(-$	$,\quad H\ ,$	$P\|(y,H))$

Figure 3: System S1, The set P is the collection of all the local history prefix variables. It contains pairs (i, H_i), which can represent the local prefix history H_i for node i. The first two rules are from System S, but with an extra parameter field.

reachable states of System S1. Thus, it is easy to show that it too satisfies the prefix property.

Lemma 2 *System* Token *satisfies the prefix property.*

4 Converting to Message Passing

A somewhat less abstract protocol that includes message passing and does not have global state will now be defined. The components of global variables are distributed among the nodes. We shall use the notation in which a global variable is a set of its distributed state at each node.

Rather than maintaining the history as a global variable, it is passed around the nodes as part of some message. When the message arrives at a node, that node can use the message to update its prefix history. We also must translate the single rule that instantaneously changes state at two different nodes, into the more realistic two rules: one for sending and another for receiving messages. Each node has some input set I and output set O that are maintained using a rule that models message passing. The specification for System Message-Passing is given in Figure 5.

It is easy to show correctness by using the idea of drained states. Moreover, it is easy to map a state in System Message-Passing back to a state in System Token by getting rid of states in which the transmission sets are empty, by stopping any new transmissions and letting everything drain. We also map the maximal H_x in System Message-Passing to H in Token.

Lemma 3 *System* Message-Passing *satisfies the prefix property.*

In the next section we address the token passing strategies and associated performance issues.

5 Performance Issues

Often there are several competing criteria for optimization. Traditional protocols integrate performance with correctness making it difficult to modify a protocol to improve its performance without violating some correctness criteria. Our approach is to separate the two concerns. The basic protocol just described says nothing about performance. It assumes that the token can be passed to any node and at any time, leaving wide scope for implementation. This section addresses performance issues such as the waiting time and number of messages.

A node that requires the token, should get it as soon as possible. It is a non-responsive system that will allow a node to wait for a long time to receive an unused token.

Definition 2 *The* Responsiveness *of a system is the maximum time period during which at least one node requires the token and until the token is given to some node that requires it.*

Note that responsiveness is not the same as average waiting time. For example, when all nodes simultaneously require the token, the responsiveness is $O(1)$, whereas the average delay can be $O(N)$ since at least half the nodes will have to wait $N/2$ cycles before they get the token. Also note that the time period is not from the time a node requires the token to the time the same node receives the token, but rather from the time a node requires it to the time at which some node gets it that also required it.

In reasoning about the performance of our protocols, we associate the cost of zero time with rules that affect only the local state of a node, and some constant time cost with the rules that result in message passing, i.e., the rules that add data to any pair in O, the output set.

When considering a specific system, its performance and responsiveness, we restrict the conditions under which certain rules may be applied. These conditions always involve

System Token

0	*Initial state* (Q, H, P, T)	:	$(\Vert_{x\in\mathcal{P}}(x, \phi_x) \quad , \emptyset \quad , \Vert_{x\in\mathcal{P}}(x, \emptyset) \quad , x)$
1	$(Q\vert(x, d_x) \, , - \, , - \quad , -)$	$\rightarrow$	$(Q\vert(x, d_x \oplus new_x) \, , - \quad , - \quad , -)$
2	$(Q\vert(x, d_x) \, , H \, , P\vert(x, -) \, , x)$	$\rightarrow$	$(Q\vert(x, \phi_x) \quad , H \oplus d_x \, , P\vert(x, H \oplus d_x) \, , y)$

Figure 4: System Token, The second rule is a combination of rules 2 and 3 of System S1. A token passing mechanism is represented by restricting the update to the global history record. Node x can append its data only when it has the token, which is represented by the fourth (rightmost) field T.

System Message-Passing

0	*Initial state* (Q, P, T, I, O)	$: \; (\Vert_{x\in\mathcal{P}}(x, \phi_x), \Vert_{x\in\mathcal{P}}(x, \emptyset), x, \emptyset, \emptyset)$
1	$(Q\vert(x, d_x), -, -, -, -)$	$\rightarrow (Q\vert(x, d_x \oplus new_x), -, -, -, -)$
2	$(-, -, -, I, O\vert(x, (y, m)))$	$\rightarrow (-, -, -, I\vert(y, (x, m)), O)$
3	$(Q\vert(x, d_x), P\vert(x, H), x, -, O)$	$\rightarrow (Q, P\vert(x, H \oplus d_x), \bot, -, O\vert(x, (y, H \oplus d_x)))$
4	$(-, P\vert(x, -), \bot, I\vert(x, (y, H)), -)$	$\rightarrow (-, P\vert(x, H), x, I, -)$
3'	$(Q\vert(x, d_x), P\vert(x, H), x, -, O)$	$\rightarrow (Q, P\vert(x, H \oplus d_x), \bot, -, O\vert(x, (y, H \oplus d_x)))$ (**where** $y = x^{+1}$)

Figure 5: System Message-Passing, The history H is no longer explicitly in the global state, but is passed around as a message. The two new parameters, I and O, represent message passing between nodes. Each node has an input set and an output set. The output set O has for a node x its output, $(x, (y, m))$, that represents x sending a message m to node y. The input set I is similar, $(x, (y, m))$ means that x has received a message from y containing data m. We use the special distinguished symbol $\bot$ to represent the case where the token is in transit and thus no node has the token. To guarantee a circular token rotation, we are able to replace rule 3 by rule $3'$. This will be used to make certain performance guarantees.

only the local state (and thus can be viewed as specific implementation details). Clearly any resulting behavior of the constrained system is also a possible behavior of the unconstrained system.

We now slightly modify System Message-Passing, replacing rule 3 with rule $3'$, Figure 5, to obtain a responsiveness of $O(N)$ for N nodes.

Lemma 4 *System* Message-Passing *with Rule* $3'$ *has a responsiveness of* $O(N)$ *for* N *nodes.*

Proof sketch: We assume a cycle graph on $\mathcal{P}$. The claim follows since rule $3'$ ensures that y must be the successor of x. This yields a responsiveness of $O(N)$. When no node requires the token, it simply cycles around the nodes. As soon as a node does require the token, it will take at most N message delays until the token arrives. □

5.1 Non-deterministic Search

It is easy to see that the best responsiveness of the system is obtained when the token is always passed to the node that requires it. The problem is how does the node with the token know which node requires the token, or, alternatively, how does a node know who currently has the token.

Rather than have the token cycle over all the nodes, it is possible to send request messages to all the nodes, asking for the token. Each node maintains a set of traps, W, indicating who wants the token. When a node receives the token, it will forward it to whomever wants it. The rules for this specification, System Search, are given in Figure 6.

To obtain a responsiveness of $O(N)$ for N nodes for System Search, we modify the System with the following restrictions: Rule 4, in which the token can be given to some arbitrary node is disabled (by imposing a precondition **false**) and replaced by rule $3'$ from the modified System Message-Passing. Rules 5 and 6 are restricted so that they send messages to their cyclic neighbors (by requiring that $y = x^{+1}$ and $u = x^{+1}$ respectively).

Lemma 5 *The modified System* Search *has a responsiveness of* $O(N)$ *for* N *nodes.*

Proof sketch: The restrictions force the requests to traverse the nodes in a cyclic order. It is now easy to see that within N message delays, the token will be found and send to the requester. □

5.2 Binary Search

The circular token rotation restriction of System Search obtains linear responsiveness, which is inherent in the sequential search strategy. A responsiveness of $O(\log n)$ is possible by doing a parallel search on the ordered nodes; a node recursively asks two other nodes to search for the token. However, such a search requires an unreasonable number of messages, in the worst case it takes $\Theta(n)$ messages for each node that requires the token and there can be many bottlenecks as well. It is somewhat surprising that a more efficient protocol results from combining the two modified protocols. That is, the token flows around a ring. Whenever a node wants the token, it sends special "gimme" messages

System Search

0	*Initial state* (Q, P, T, I, O, W)	$: \ (\|_{x\in\mathcal{P}}(x, \phi_x), \|_{x\in\mathcal{P}}(x, \emptyset), x, \emptyset, \emptyset, \emptyset)$
1	$(Q\|(x, d_x), -, -, -, -, -)$	$\rightarrow (Q\|(x, d_x \oplus new_x), -, -, -, -, -)$
2	$(-, -, -, I, O\|(x, (y, m)), -)$	$\rightarrow (-, -, -, I\|(y, (x, m)), O, -)$
3	$(-, P\|(x, -), \perp, I\|(x, (y, H)), -, -)$	$\rightarrow (-, P\|(x, H), x, I, -, -)$
4	$(Q\|(x, d_x), P\|(x, H), x, -, O, -)$	$\rightarrow (Q, P\|(x, H \oplus d_x), \perp, -, O\|(x, (y, H \oplus d_x)), -)$
5	$(-, -, -, -, O, W)$	$\rightarrow (-, -, -, -, O\|(x, (y, \tau_x)), W\|(x, \tau_x))$
6	$(-, -, -, I\|(x, (y, \tau_z)), -, W)$	$\rightarrow (-, -, -, I, O\|(x, (u, \tau_z)), W\|(x, \tau_z))$
7	$(-, P\|(x, H), x, -, O, W\|(x, \tau_y))$	$\rightarrow (-, P\|(x, H), \perp, -, O\|(x, (y, H)), W))$

Figure 6: System Search, non-deterministic token search: Rule 5: Generate interest (set trap τ_x for node x). Set a trap and send a message to some other node to trap and return the message. Rule 6: If asked, set a trap locally for the token and continue to ask some other node. Rule 7: If have trap and token then remove trap and send token.

to the node directly across the (logical) ring. A node receiving a special message either returns the token, if it has it, or it lays a local "trap" and sends a "gimme" message halfway around either in the clockwise or counter-clockwise direction depending on whether the token has been at the node before or after it being at the requesting node. The token continues to flow around the ring again from where it was first intercepted.

Figure 7 gives the precise rules – we call this specification System BinarySearch (Figure 8 illustrates rule 6 of the system).

There are several desirable properties of our binary search protocol.

Theorem 1 *System* BinarySearch *satisfies the prefix property.*

Proof sketch: Follows by mapping states to the less restricted protocol. □

Lemma 6 *In System* BinarySearch *each token request is forwarded $O(\log N)$ times for N nodes.*

Proof sketch: If the token does not move, i.e., Rule 4 is not invoked, then within $\log N$ message cycles, a search message will reach the node that has the token. If the token is moving then it will bump into a trap within $\log N$ message cycles. □

Theorem 2 *System* BinarySearch *with the additional requirement that the token traps are locally stored and processed in* FIFO *order, has a responsiveness of $O(\log N)$ for N nodes.*

Proof sketch: This follows from the constraint of System BinarySearch and Lemma 6. □

Theorem 3 *System Binary Search is* $\log N$ *fair. That is, during the time when some node x wants the token, time t_0, and gets it, time t_1, no one node gets the token more than $\log N$ times, and there are no more than N possessions of the token by other nodes.*

Proof sketch: We need to show that some node cannot get the token more than $\log N$ times while some other node wants it. Suppose the token is at node x, that node r continually requests the token, and that node w also requests the token. Within $\log N$ message steps, the request from node w will arrive at node x by the binary search. Assuming that nodes handle messages and satisfy requests in FIFO order, then when node r returns the token, it will be sent to node w.

Similarly, if all nodes search for the token, then node w's request will be satisfied within $N + \log N$ message steps. □

It is also possible to have nodes keep their requests local and have the token find which node wants it. That is, if a node has the token, it sends out probing messages to see which node wants the token. Once again, if the token or some dummy request cycles around the nodes, then a binary search can be performed requiring only logarithmic number of messages rather than linear. Finally, it is possible to combine both schemes.

5.3 Simulation results

We have conducted an empirical study by implemeneting a simulation of System Binary Search and the regular token rotation protocol and observing their behavior under various load scenarios. Figure 9 and Figure 10 show some of the results. In each simulation 1000 rounds were performed, i.e., the token visited each node at least 1000 times. In Figure 9 we see the results of a simulation where we fixed the load; on average, every 10 time units one of the nodes in the system makes a request. The curves show, that using a regular ring algorithm, the average responsiveness approaches 10, the average distance between nodes on the ring that make a request. Using System Binary Search, the average responsiveness is bounded by $\log n$, where n is the number of processors. This is further illustrated in Figure 10. Here we decrease the load and fix the number of processors ($n = 100$). Using System Binary Search, the average responsiveness approaches $\log n$ from below. For the regular ring algorithm the average responsiveness approaches $n/2(= 50)$.

System BinarySearch

0	*Initial state* (Q, P, T, I, O, W)	$: \ (\Vert_{x\in\mathcal{P}}(x, \phi_x), \Vert_{x\in\mathcal{P}}(x, \emptyset), x, \emptyset, \emptyset, \emptyset)$
1	$(Q\mid(x, d_x), -, -, -, -, -)$	$\rightarrow (Q\mid(x, d_x \oplus new_x), -, -, -, -, -)$
2	$(-, -, -, I, O\mid(x, (y, m)), -)$	$\rightarrow (-, -, -, I\mid(y, (x, m)), O, -)$
3	$(-, P\mid(x, -), \bot, I\mid(x, (y, H)), -, -)$	$\rightarrow (-, P\mid(x, H), x, I, -, -)$
4	$(Q\mid(x, d_x), P\mid(x, H), x, -, O, -)$	$\rightarrow (Q, P\mid(x, H \oplus d_x), \bot, -, O\mid(x, (x^{+1}, H \oplus d_x)), -)$
5	$(-, P\mid(x, H), -, -, O, W)$	$\rightarrow (-, P\mid(x, H), -, -, O\mid(x, (y, N, H, \tau_x)), W\mid(x, \tau_x))$
6	$(-, P\mid(x, H), -, I\mid(x, (y, n, H_z, \tau_z)), O, W)$	$\rightarrow (-, P\mid(x, H), -, I, O\mid(x, (u, \frac{n}{2}, H_z, \tau_z)), W\mid(x, \tau_z))$ where $u = x^{-n/2}$ if $H \subset_C H_z$, and $u = x^{+n/2}$ if $H_z \subset_C H$
7	$(-, P\mid(x, H), x, -, O, W\mid(x, \tau_y))$	$\rightarrow (-, P\mid(x, H), \bot, -, O\mid(x, (\hat{y}, H)), W))$
8	$(Q\mid(x, d_x), P\mid(x, -), \bot, I\mid(x, (\hat{y}, H)), O, -)$	$\rightarrow (Q, P\mid(x, H \oplus d_x), \bot, I, O\mid(x, (y, H \oplus d_x)), -)$

Figure 7: System BinarySearch, this is a combination of a circular token rotation and a binary search for the token. The first four rules are the same as in System Search. The more complicated search messages contain the local prefix history of the requesting node. Rule 6 compares what the requesting node has seen with what x has seen to decide how to continue the search. ($A \subset_C B$ is the prefix comparison of histories when the histories are projected onto the circular token ring rotation events.) Rule 7 sends the token to the requester indicated by the trap τ_y; the decorated $\hat{y}$ indicates that the token is to be returned upon use. Rule 8 gets the token in response to a search request and then immediately returns it to the sender.

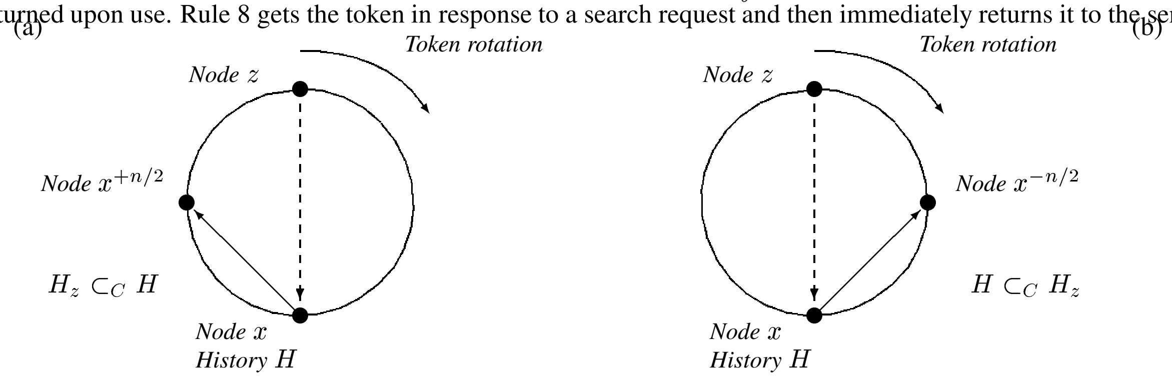

Figure 8: Illustration of Rule 6 of System BinarySearch: (a) the history H_z of the requester at the time of request is a prefix of the history H at node x (when projected onto the token circulation events) – search continues clockwise; (b) the history H at node x is a prefix of the history H_z of the requester at the time of request – search continues counter-clockwise.

5.4 Optimization considerations

In this section we briefly discuss some of the numerous possible optimization and refinement alternatives for our non-deterministic and binary search protocols. First of all, we have utilized unbounded-size sets and histories for simplicity of presentation. It is easy to bound the size of the sets by throttling the requests and messages issued at the nodes. For the ring protocols the histories can be bounded by introducing the notion of a round and using round counters.

The protocol given by System BinarySearch can be viewed as a *delegated search*. Here, the search message, once issued by the requesting node, finds its way to the token with the help of other nodes. Nodes remember the search requests they receive from other nodes in the form of traps that are stored locally. To reduce the storage cost of the protocol we want to garbage-collect the obsolete traps. This gives rise to two different algorithms:

Token-rotation clean up. Here the token itself is used to remove old traps during subsequent token rotation rounds. The token can carry enough information, e.g., round number, information about the satisfaction of a search request, to enable the nodes to clean up.

Inverse token clean up. The token after, it was found, is not passed directly to the requesting node, but rather traces the trail left behind by the search messages. In doing so, it removes the traps *en route* to the requester.

Another modification to the protocol would be the *directed search*, where search messages do not migrate through the ring but instead are always returned to the searching node informing it whether the token was found or not. This increases (doubles) the number of search messages from $\log N$ to $2 \log N$ in the worst case, however if the token reaches the searching node during the search due to its normal rotation, the search can be stopped, thus saving some number of messages.

In yet another further extension, nodes ensure that they have only one request (one "gimme" message) outstanding. All other requests that arrive are delayed until the first request has been satisfied. This reduces the number of "gimme" messages to be no more than the number of token passing messages.

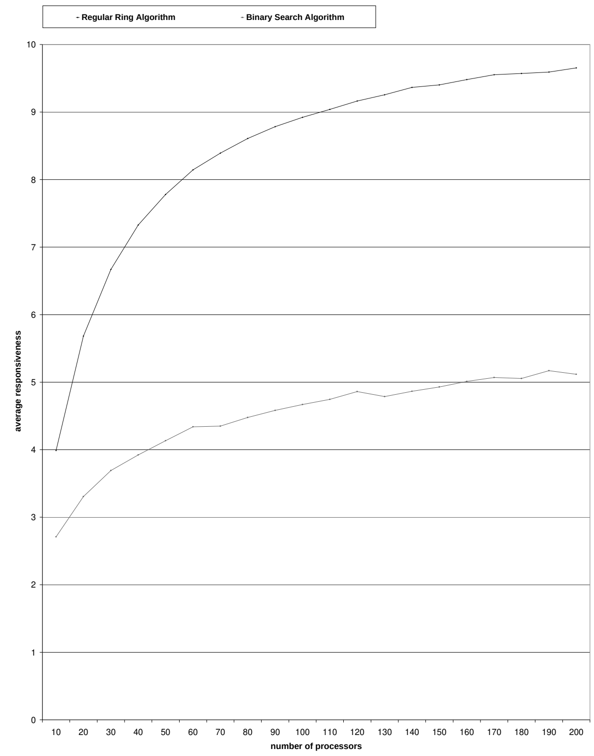

Figure 9: Performance with fixed load

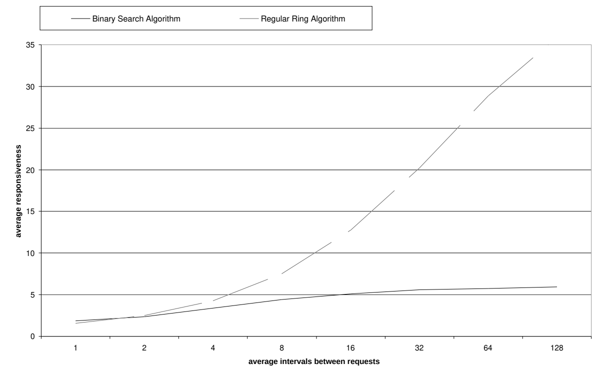

Figure 10: Performance with fixed number of processors

Finally, we note that the speed of token passing around the cycle can be varied according to the demand – very slow when only a few nodes require the token and much faster when there is high demand.

6 Conclusions and Future Work

The methodology followed in this paper provides wide scope for performance optimizations and adaptive tuning. In particular, adaptive algorithms can often be immediately derived from the abstract specifications in a methodical way that preserves correctness. The methodology in this paper is based on the refinements to specifications stated as transition systems. In this paper we use the Term Rewriting Systems (TRS) [2] framework to formally express specifications. We note that we could have done the same, maybe not as succinctly as presented here, using Temporal Logic of Actions (TLA) [11], or Input/Output Automata (IOA) [12, 14].

We have specified a family of token passing protocols. One interesting observation is that messages are used in two different ways. One is to pass the token (potentially with the message history). Another is to push the system along certain performance optimized trajectories. The first requires that messages arrive correctly, or at least with a high probability as well as a mechanism to resend in case of failure (not considered in this paper). As is usually the case, raising the likelihood of success is more expensive in terms of several different resources. The second can be accomplished using "lightweight" messages, since they do not impact system correctness, only its performance.

Token-passing protocols, as sometimes considered in the literature, trade token access efficiency for loads at certain nodes, e.g., as is the case with fixed tree-based topologies where fast access comes at the cost of high loads at the roots. Our token-passing scheme is able to capture in a single protocol the good load balancing and fairness property of the ring, with the efficiency of logarithmic access. This is accomplished in a fluid way, where the token-possessing nodes can temporarily act as virtual roots of a token-distribution tree.

Our future plans include making the protocols more dynamic with respect to the nodes comprising the network. It is possible to modify the protocol to handle nodes that asynchronously leave and join the group. The search mechanism needs to know those nodes that are halfway, 1/4 way, etc., around the cycle. An approximation may be sufficient, and so nodes need only a set of a logarithmic number of neighbors. Note also that by combining token traversal with searching, the protocol already has a way of handling failures. If a node x with the token fails, then nothing will happen until some other node y needs the token, at which point it will quickly discover that the token holder has failed (provided a time-out based detection is available). Node y will then get in touch with x^{-1} and x^{+1}. These two nodes can then determine if x is really dead and if the token was at x. If so, they can generate a new token.

Acknowledgments: We thank Greg Malewicz for several valuable observations and comments.

References

[1] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal and P. Ciarfella, "The Totem single-ring ordering and membership protocol," ACM Transactions on Computer Systems, vol. 13, no. 4, November 1995, pp. 311-342.

[2] Arvind and X. Shen, "Design and Verification of Processors Using Term Rewriting Systems", CSG Memo 419, LCS, MIT; to appear in *IEEE Micro*, 1999.

[3] S. Banerjee and P. K. Chrysanthis, "A New Token Passing Distributed Mutual Exclusion Algorithm ", In Proc. of the 16th International Conference on Distributed Computing Systems, pp. 717–724, Hong Kong, May 27-30, 1996.

[4] H. Buhram, M. Franklin, J.A. Garay, J.-H. Hoepman, J. Tromp and P. Vitanyi, "Mutual Search", J. ACM, vol. 46, no. 4, pp. 517-536, 1999.

[5] Y-I. Chang, M. Singhal, and M. T. Liu. "A dynamic token-based distributed mutual exclusion algorithm", In Proc. 10th IEEE Int'l Phoenix Conf. on Computers and Communications, pp. 240-246, 1991.

[6] *Communications of the ACM*, special section on group communications, vol. 39, no. 4, 1996.

[7] F. Commoner, A.W. Holt, S. Even and A. Pnueli, "Marked directed graphs", JCSS, vol. 5, no. 6, pp. 511-523, 1971.

[8] J.-M. Helary and A. Mostefaoui, "A $O(\log_2 n)$ fault-tolerant distributed mutual exclusion algorithm based on open-cube structure", Research Report 2041, INRIA, September 1993.

[9] J.-M. Helary, A. Mostefaoui and M. Raynal, "A General Scheme for Token- and Tree-based Distributed Mutual Exclusion Algorithms," Research Report 1692, INRIA-IRISA, May 1992. Also in IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 11, pp. 1185-1196, 1994.

[10] L. Lamport, "Time, clocks, and the ordering of events in a distributed system", Comm. ACM, vol. 21, no. 7, pp. 558-565, 1978.

[11] L. Lamport, "The Temporal Logic of Actions", *ACM Transactions on Programming Languages and Systems*, 16(3):872-923, 1994.

[12] N.A. Lynch, *Distributed Algorithms*, Morgan Kaufman Publishers, San Mateo, CA, 1996.

[13] N.A. Lynch and M.R. Tuttle, "Hierarchical Correctness Proofs for Distributed Algorithms", in Proc of the 6th Annual ACM Symp. on Principles of Distributed Computing, pp. 137-151, 1987.

[14] N.A. Lynch and M.R. Tuttle, "An Introduction to Input/Output Automata", *CWI Quarterly*, vol.2, no. 3, pp. 219-246, 1989.

[15] G. Peterson and M. Fischer, "Economical Solutions for the Critical Section Problem in a Distributed System", in Proc of the 9th Annual ACM Symp. on Theory of Computing, pp. 91-97, 1977.

[16] M. Raynal, Algorithms for Mutual Exclusion, MIT Press, 1986.

[17] X. Shen, Arvind and L. Rudolph "CACHET: An Adaptive Cache Coherence Protocol for Distributed Shared-Memory Systems", In proceedings of the 13th ACM SIGARCH International Conference on Supercomputing, June 1999, Rhodes, Greece, 1999.

[18] M. Singhal, "A Taxonomy of Distributed Mutual Exclusion", Journal of Parallel and Distributed Computing 18(1), pp. 94-101, 1993.

[19] R. Vitenberg, I. Keidar, G.V. Chockler and D. Dolev, "Group Communication Specifications: A Comprehensive Study", MIT Technical Report MIT-LCS-TR-790, September 1999. URL http://theory.lcs.mit.edu/~idish/ftp/gcs-survey-tr.ps.