Implementing Atomic Data through Indirect Learning in Dynamic Networks*

Kishori M. Konwar*, Peter M. Musial‡, Nicolas C. Nicolaou *, Alex A. Shvartsman*,†,‡

*Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269, USA

†Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139, USA

‡ VeroModo. Inc, 11 Osborne Rd., Brookline, MA 02446 USA.

Abstract

Developing middleware services for dynamic distributed systems, e.g., ad-hoc networks, is a challenging task given that such services deal with dynamically changing membership and asynchronous communication. Algorithms developed for static settings are often not usable in such settings because they rely on (logical) all-to-all node connectivity through routing protocols, which may be unfeasible or prohibitively expensive to implement in highly dynamic settings. This paper explores the indirect learning*, via periodic gossip, approach to information dissemination within a dynamic, distributed data service implementing atomic read/write memory service. The indirect learning scheme is used to improve the liveness of the service in the settings with uncertain connectivity. The service is formally proved to guarantee atomicity in all executions. Conditional performance analysis of the new service is presented, where this analysis has the potential of being generalized to other similar dynamic algorithms. Under the assumption that the network is connected, and assuming reasonable timing conditions, the bounds on the duration of read/write operations of the new service are calculated. Finally, the paper proposes a deployment strategy where indirect learning leads to an improvement in communication costs relative to a previous solution that assumes all-to-all connectivity.*

1 Introduction

Distributed middleware services for dynamic systems must deal with communicating devices that may fail, join, or voluntarily leave the system, and experience arbitrary delays in message delivery. A common design approach in such settings is to have the participating network nodes periodically exchange their local state information with the goal of approximating the global state of the system and ensuring progress of local computation [4, 8, 16]. Performance of a service implemented in this way depends on the prompt update of the local state at each node, hence requiring (logical) all-to-all communication, which can be quite expensive. The communication cost associated with all-to-all communication can be reduced by minimizing the number of bits in the message [2], or by limiting the communication by assigning to each sender a proper subset of the nodes to communicate with [12]. Such methods can lead to good results in static environments, however their utility is diminished in highly dynamic networks. A weakness of all-to-all gossip is its reliance on the existence of point-to-point connectivity. This is an important limitation, since in dynamic systems such as ad-hoc and mobile networks, routing information is prohibitively expensive, where significant amount of power, memory, and communication are needed to keep the routing tables up to date [10, 18, 19, 20]. Furthermore, routing protocols provide a general solution and are oblivious to the data flows of specific applications, which results in unnecessary communication burden. On the other hand, in the absence of a routing service no predictable progress can be ensured in algorithms depending on all-to-all gossip.

In this paper we incorporate an indirect learning protocol within a distributed algorithm implementing atomic objects aimed at enhancing its effectiveness in dynamic networks. Our algorithm is based on the RAMBO [8, 16] algorithms and ensures atomicity in all executions while tolerating node departures, joins, failures, and message loss. Data objects are replicated to ensure survivability. To maintain consistency in the presence of small and transient changes, the algorithm uses *configuration* consisting of *quorums* of locations. To accommodate larger and more permanent changes, the algorithm supports *reconfiguration*, by which the configurations

*This work is supported in part by the NSF Grants 9988304, 0121277, and 0311368.

are modified. All decisions regarding the locally initiated operations on the replica are made by examining the local state. In order to update the local state and ensure operation liveness, RAMBO algorithms rely on point-to-point connectivity and use periodic all-to-all gossip to disseminate the state of replicas. Our goal is to enable progress of data access operations (reads and writes) as long as there are quorums in active configurations whose nodes are connected, either directly or indirectly, and without relying on routing protocols.

Contributions. We present an implementation of the RAMBO [16] service that incorporates an indirect learning mechanism designed to take advantage of the semantics of the internal data flow to effectively disseminate object replica information among participating nodes. We call the new algorithm ATILA (*atomicity through indirect learning algorithm*). The dynamic settings considered include mobile ad-hoc networks (MANETS), and we do not assume an underlying routing protocol or all-to-all direct connectivity.

ATILA implements indirect learning through gossip directly with neighboring service participants and achieves improvements in operation liveness, in dynamic settings, at the expense of higher memory consumption. Each node is required to maintain an estimate of the replica state at each service participant. This information is included in gossip messages. We present a general solution that is oblivious to the communication structure or existence of routing protocols. Our solution allows deployment optimizations that improve its performance; we discuss one such optimization.

We formally prove that ATILA implements atomic objects. The performance of read and write operations of the service is affected by the properties of the service deployment graph, where the edges are direct communication links between nodes. The more challenging and interesting part of this work is the probabilistic analysis that estimates the duration of read/write operations, and the analysis that uncovers the possible savings in cost per message bit. Of independent interest, we believe that our analysis approach can be generalized to other algorithms that use quorums.

Related work. Dynamic distributed systems with an unknown and possibly unbounded number of participants that may join, voluntarily leave, and fail, are becoming increasingly common. Applicable to these settings problems include, for example, consensus [14] and maintenance of consistent memory [3].

Group communication services (GCS) [1] are important building blocks in distributed systems and can be used to implement shared memory abstractions. However, communication required for group maintenance limits the utility of common GCSs in dynamic environments such as MANETS. Dynamic node participation and node mobility force frequent group membership changes and group maintenance becomes an expensive task requiring high communication overhead and energy consumption [11].

The GEOQUORUMS approach of [3] uses stationary *focal points*, implemented by mobile nodes, to provide atomic shared read/write memory where consistency is maintained by using quorums of focal points. This approach requires *geocast* communication that can deliver messages to specific geographic locations. Earlier RAMBO service [16] was developed for dynamic overlay networks, where messages are routed automatically. Specification of RAMBO trades mathematical simplicity for practicality, and while its successive refinements [5, 6, 8, 9] improved usability of its implementation each still relies on the all-to-all connectivity.

Overlay networks allow transparent routing of messages atop diverse communication structures. Nodes communicate using virtual point-to-point channels with the help of routing protocols. Many routing algorithms for ad-hoc and mobile networks have been proposed [10, 18, 19, 20]. However, routing protocols have drawbacks: (i) Maintenance of overlay routes in systems where nodes join, migrate, depart, and fail, is expensive in terms of processing, memory consumption, and communication. As node membership and location varies the topology of the network may change and the new virtual routes have to be recalculated often in order to maintain integrity of the overlay network. (ii) Routing protocols are oblivious to the semantics of the communication among the participating nodes. Hence, there may be substantial redundancy in communication. In the networks that are sensitive to throughput, increased communication burden may have adverse effects on the performance of the routing algorithms themselves and on the message-passing applications.

Document structure. In Section 2 we present the model and definitions. We describe our algorithm in Section 3. The proof of atomicity is outlined in Section 4. Probabilistic performance analysis is presented in Section 5 and the deterministic analysis in Section 6. We conclude in Section 7. The complete specification of our algorithm and selected details of the correctness proof can be found in [13].

2 System Model and Definitions

We assume a message-passing model with asynchronous processors (i.e., *nodes*) that have unique identifiers from set I, which needs not be finite. Nodes may join, crash, and voluntarily leave the system. Nodes communicate via point-to-point, direct, asynchronous channels. A node can send a

message to another node if a direct link between the nodes exists. In the wireless setting a *virtual* point-to-point link between two nodes exists if these nodes reside within the effective transmission range of each other. In addition, the broadcasting node effectively sends a message to all nodes within its transmission range. The nodes and the point-to-point communication links form the *service deployment graph*.

Let C denote the set of *configuration identifiers*. For each $c \in C$ we define: *members*(c), a finite subset of node identifiers, *read-quorums*(c), a set of finite subsets of *members*(c), and *write-quorums*(c), a set of finite subsets of *members*(c). We require that for every $R \in \mathit{read\text{-}quorums}(c)$, and every $W \in \mathit{write\text{-}quorums}(c)$, $R \cap W \neq \emptyset$, and that for any $W_1, W_2 \in \mathit{write\text{-}quorums}(c)$, $W_1 \cap W_2 \neq \emptyset$. No intersection requirement is imposed on the sets of members or on the quorums from distinct configurations.

We define $C_\perp = C \cup \{\perp\}$ and $C_\pm = C \cup \{\perp, \pm\}$ to be the partially ordered sets, such that: $\perp < c$ and resp. $\perp < c < \pm$, for $c \in C$. We define the set ***CMap***, the set of configuration maps, as the set of mapping $\mathbb{N} \to C_\pm$. In any sequence in ***CMap***, the symbol $\perp$ represents an unknown configuration and $\pm$ represents obsolete configuration that has been removed. Finally, we define *update* to be a binary operation on $cm, cm' \in \mathit{CMap}$ that updates any element in cm with the corresponding element in cm' if that element is greater according to the partial order $C_\pm$.

3 The ATILA Algorithm

We now present the algorithm implementing a dynamic atomic object service using an indirect learning protocol. The algorithm is based on RAMBO [16] and its refinements in [8, 5], and we call the new algorithm ATILA. The service is defined for a single object — where atomicity is preserved under composition. The pseudocode implementing read and write operations of the algorithm appears in Figures 1.

In order to ensure fault tolerance, object data is replicated at several nodes. The algorithm uses *quorum configurations* to maintain consistency. Configurations can be modified on-the-fly through *reconfiguration*. Main parts of the algorithm deal with communication with replicas during read and write operations, and the removal of the obsolete configurations using *configuration upgrade* operations.

Participant Information. Each participant maintains the *value* and the associated *tag* of the object being replicated. The *tag*s are used to totally order write operations with respect to each other and all read operations with respect to the writes — this forms the basis for the proof of atomicity (Section 4). Each node maintains a set of node identifiers, *world*, representing the nodes that are locally known to have joined the service, and the configuration information stored in variable $\mathit{configs}$ of type CMap.

Each node uses *phase numbers* to logically timestamp the messages it sends to other nodes indicating the "freshness" of the state conveyed in the messages. The phase number of a node is incremented following an "important" event at a node, such as the start of a new phase of a read or a write, or a configuration upgrade operation. Most importantly, phase numbers are used to implement indirect learning as discussed later in this section. Each node i maintains a matrix of phase numbers, *pNums*, where rows and columns are indexed by node identifiers, hence its size is $|\mathit{world}| \times |\mathit{world}|$. The variable $\mathit{pNums}[i][j]$ represents the most recent phase information known to i about another participating node j. This means that i has learned the replica information known to j when j's phase number was equal to $\mathit{pNums}[i][j]$. The variable $\mathit{pNums}[j][k]$, for some $j, k \in \mathit{world}$ and $i \neq j$, represents the most recent phase number known to i about the phase of node k that is known to j. Each of these variables reflects the latest information locally known at a node, but not necessarily the most up-to-date global information.

Each node maintains two records to store information about the ongoing operations. Record op is used to keep track of the phases of read and write operations. Fields of op are initialized when a new phase of a read or write operation is initiated: $\mathit{op\text{-}configs}$ records the value of $\mathit{configs}$, $\mathit{op\text{-}Nums}$ records the value of pNums, and $\mathit{op\text{-}acc}$, initially $\emptyset$, records the identifiers of the nodes that contain adequately current information regarding i's state. Similarly, record upg is used to track the configuration upgrade operation, where the fields $\mathit{upg\text{-}configs}$, $\mathit{upg\text{-}Nums}$ and $\mathit{upg\text{-}acc}$ are defined analogously to the fields of op record. In addition, the upg record contains field $\mathit{upg\text{-}target}$, an the index of the configuration being upgraded. (The phases of read, write, and configuration operations are discussed later in this section).

Information Propagation and Indirect Learning. Periodically, and following certain events, any non-failed participant of the service sends state messages to all nodes found in its local world. These messages include sender's current values of: tag, val, $\mathit{configs}$, world, and pNums. Although a node attempts to send messages to all nodes in its world, only the messages addressed to the nodes with a direct connection may be delivered, all other messages may be lost.

We now narrate the update process based on an example of a message exchange between two non-failed service participants, say i and j. When i receives message from j it compares values of variables comprising its state against the

- **RW-Start:** Node i resets its local structures pertaining to the read/write operations, such as: $op\text{-}configs$, $op\text{-}Nums$. Also, it notes that a read or a write operation was initiated.
- **RW-Phase-1a:** Node i increments its local phase number and updates the $pNums$ set with the new information. A snapshot of the information stored in $configs$ and $pNums$ is recorded in $op\text{-}configs$ and $op\text{-}pNums$. At this point node i sets out to query configurations found in $op\text{-}configs$ for the most recent tag and $value$ information. Next, i sends $\langle RW1a, tag, val, configs, world, pNums\rangle$ message to all known participants of the service, i.e. $world$.
- **RW-Phase-1b:** Upon receipt of a $\langle RW1a, t, v, c, w, pn\rangle$ message from i, node j compares its local knowledge (local state values) with the information included in the message. For instance if its local tag is strictly smaller than t, then it updates its tag with t and $value$ with v. Also, it updates its $configs$, $world$, and $pNums$. Next, j replies to i with $\langle RW1b, tag, val, configs, world, pNums\rangle$.
- **RW-Phase-1c:** Upon receipt of a $m = \langle RW1b, t, v, c, w, pn\rangle$ message from j, node i updates its state based on comparison of the values of its local state with the related information found in the message. If $m.c$ contains configurations previously unknown to i, then the current phase is restarted.
- **RW-Phase-2a:** Node i compares $m.pn$ and $op\text{-}pNums$ to check if at least one read quorum of each configuration found in $op\text{-}configs$ has an adequately recent state information of i (i.e. has at least learned the phase number of i from **RW-Phase-1a**) If so then the first phase is complete – i is now in the position of the highest tag. At this point node i sets out to propagate to the members of configurations found in $op\text{-}configs$ the most recent tag and $value$ information. Node i increments its phase number and updates its $pNums$ with the new information, it also records current values of $configs$ and $pNums$ in $op\text{-}configs$ and $op\text{-}pNums$. Next, i broadcasts $\langle RW2a, tag, val, configs, world, pNums\rangle$ message where tag and $value$ depend on whether it is a read or a write operation: in the case of a read, they are just equal to the local tag and $value$; in the case of a write, they are a newly chosen tag, and v, the value to write.
- **RW-Phase-2b:** If node j receives a $\langle RW2a, t, v, c, w, pn\rangle$ message from i, it updates its state accordingly, and responds to i with $\langle RW2b, tag, val, configs, world, pNums\rangle$.
- **RW-Phase-2c:** Same as **RW-Phase-1c**.
- **RW-Done:** If node i can determine that at least one write quorum of *all* configurations in $op\text{-}configs$ has an adequately recent state information of i (i.e. has at least learned the phase number of i from **RW-Phase-2a**), then the read or write operation is complete and the tag is marked confirmed. If it is a read operation, node i returns its current value to client. Node i marks that the operation is now terminated. At this point new read/write operation may be initiated at node i.

Description of the phases of the read and write protocols.

information included in the message. Assume that i receives message $m = \langle tag, val, configs, world, pNums\rangle$ from j. If $m.tag \geq tag$ then i updates its tag with $m.tag$ and the value with $m.val$. Next, i includes in its $world$ any new identifiers found in $m.world$. For each new node identifier, matrix $pNums$ is extended with a new column and a new row, initialized to zeros. Also, i sets its $configs$ to $update(configs, m.configs)$.

The last step updates the phase information, where i compares its phase matrix with the one in the sender's message. This update captures the indirect learning process. For all $k, \ell \in m.world$, if $m.pNums[k][\ell] > pNums[k][\ell]$, then j knows that k has learned about a higher phase number of ℓ. Therefore, whenever $m.pNums[k][\ell] > pNums[k][\ell]$ then i assigns $pNums[k][\ell] \leftarrow m.pNums[k][\ell]$.

Observe that all bookkeeping information (except for value) is monotonically growing with each update, i.e., a tag is updated only when the arriving tag is larger, nodes are only added to the $world$ set, and the phase number information is updated if the incoming phase number information is more recent than what i is aware of. Therefore, if some node k learns that i's phase number is p, then k has learned of a tag (resp. value) of the replica that is at least as recent as when i's phase number was p. Phase numbers are updated either following a receipt of a message directly from k or indirectly from some other node. Thus if i is performing some operation and p is its current phase number then if $pNums[k][i] \geq p$, then i can deduce that k learned the information that is at least as recent as the information communicated by i to its $world$ in phase p. (Finally, if the service deployment graph is connected and the network is reasonably well-behaved, then eventually i will (indirectly) learn that k (indirectly) learned the information disseminated by i.)

Joining. Nodes join the service by sending a join request to the nodes provided by the user ("seeds"). Our well-formedness assumption is that when the set of seed nodes is empty, the node processing the join request is the "creator" of a new object. If an active participant of the service receives a join request it will add sender's identifier to its local *world* set and reply with a state message. The joinee becomes operational (*active*), when a response message to the join-request is received.

Read and Write Operations. The read and the write operations are conducted in two phases (see Figure 1): **RW-Phase-1**, or *query* phase, is identical for both operations. In this phase the replica owners are queried in regard to the most recent *tag* and the associated *value*. **RW-Phase-2**, or *propagation* phase. During this phase, the replica information is propagated to the replica owners. In case of the read operation the replica information discovered during **RW-Phase-1** is propagated. In case of the write operation the new *tag* and the associated *value* are propagated to the replica owners,

where the *tag* is strictly grater than the one discovered during preceding **RW-Phase-1**. The termination point of each phase is determined only after the node conducting this operation can certify that at least one quorum of replica owners from each active quorum set has responded (directly or indirectly) to its latest phase information.

Reconfiguration and Configuration Upgrade. Reconfiguration process has three stages. First, a new configuration is introduced by some active service participant. Second, the proposed configuration is installed, this is handled by an external service, called *Recon*, as in [16]. Finally, the obsolete configurations are removed using the *configuration upgrade* operation. Liveness of the service is ensured as long as old configurations remain operational until they are removed.

Configuration upgrade operation is implemented in two phases, which are similar to phases of read operation. The goal of the query phase is to obtain the most recent replica information from the appropriate quorums of all active configurations with index smaller than that of the configuration being upgraded. In the propagation phase the newly discovered replica information is propagated to the configuration being upgraded. The formal specification using Input/Output Automata notation [17] appears in [13].

4 Proof of Atomic Consistency

In this section we formally show that ATILA implements atomic objects by applying necessary refinements on the safety proofs of RAMBO [8]. The challenge is to show that atomicity is ensured when indirect mechanism is used. We present only parts of the RAMBO [8] proof framework that need modification. (The omitted details are covered in [13].)

We consider *well-formed* executions of the algorithm for each active participant, i, where: i follows the protocols for joining and reconfiguration, i initiates only one operation at a time, and i waits for appropriate acknowledgments before proceeding.

Let α be an arbitrary well-formed execution of the algorithm, and let π_1 and π_2 be two read or write operations that occur at i and j respectively — non-failed participants of ATILA. Additionally, we assume that π_1 completes before π_2 begins in α. When ordering of operations is not important, we use π to denote an arbitrary read or a write operation.

For every π, the query-fix (resp. prop-fix) event occurs immediately after the *query* (resp. *prop*) phase of π completes. Therefore, query-fix point occurs at the point when node i determines that at least one read quorum of each configuration in $op\text{-}configs$ has a sufficiently recent state information of i, which happens in phase **RW-Phase-2a** (Figure 1). A similar relation exists between prop-fix and **RW-Done**.

Next we introduce history variables. The $query\text{-}cmap(\pi)$ is a mapping: $\mathbb{N} \rightarrow C_{\pm}$, initially undefined. It is set in the query-fix step of π, to the value of $op\text{-}configs$ in the pre-state, and the variable $prop\text{-}cmap(\pi)$ that is defined analogously for the propagation phase of operation π. The query-phase-start(π), initially undefined, is defined in the query-fix step of π, to be the unique earlier event at which the collection of query results was started and not subsequently restarted (the last time $op\text{-}acc$ set is assigned $\emptyset$). This is either in **RW-Start** step of a read or a write operation, or in **RW-Phase-1c** step. The event prop-phase-start(π) is defined analogously, but with respect to the propagation phase.

For every read and write operation π at i, we define the history variable $tag(\pi)$ to be the value of tag_i when the query-fix event occurs for π at node i. If π is a read operation then $tag(\pi)$ is the largest tag that node i encounters during the query phase. If π is a write operation, $tag(\pi)$ is the new tag that is chosen by i for performing the write.

Finally for any operation π we define the history variable $R(\pi, k)$, for $k \in \mathbb{N}$, as a subset of I, initially undefined. It is set in the query-fix step of π, for each k such that $query\text{-}cmap(\pi)(k) \in C$, to an arbitrary $R \in read\text{-}quorums(c(k))$ such that $R \subseteq op\text{-}acc$ in the pre-state, where $c(k) \in C$. Similarly we define $W(\pi, k)$, for $k \in \mathbb{N}$, to be a subset of I, initially undefined and set during the prop-fix step of π, for each k such that $prop\text{-}cmap(\pi)(k) \in C$, to an arbitrary $W \in write\text{-}quorums(c(k))$ such that $W \subseteq op\text{-}acc$ in the pre-state.

Phase guarantees. Results presented in this section account for the effects of query and propagation phases of read/write and configuration upgrade operations. We show that if i initiates a phase of a read/write or a configuration upgrade operation and if there exists a specific sequence of message exchanges that starts and ends at i, then if that phase terminates, i is in possession of the most recent tag and its value cannot be smaller than what i knew at the start of the phase. Moreover, we show that configuration information and value of the tag at each node that participated in the examined communication sequence has specific properties. Our claims are based on the following observation: A node sends the most recent state information that includes its configuration information, value and tag, and phase information of all service participants. By the specification of the algorithm, the receiver of this message can only increase its tag and increment the phase information in any cell of its phase number matrix. Also, the configuration information is updated only with a more recent one. This means that nodes may learn

about configuration information, tag, and phase information of other participants indirectly.

Note that in ATILA, the case $j = i$ is treated uniformly with the case where $j \neq i$. Next, we consider how the *tag* information is propagated in the query phase of the read and the write operation. Since the flow of information in the propagation phase is analogous to that in the query phase, we compress two lemmas into one.

Lemma 4.1 *Suppose that a* query-fix$_i$ *(resp.* prop-fix$_i$*) event for a read or write operation π occurs in α. Let $k, k' \in \mathbb{N}$. Suppose query-cmap$(\pi)(k) \in C$ and $j \in R(\pi, k)$ (resp. prop-cmap$(\pi)(k) \in C$ and $j \in W(\pi, k)$). Then there exists a sequence of identifiers $\langle \iota_1, ..., \iota_n \rangle$ where for all $1 \leq h \leq n$ each $\iota_h \in I$, and the corresponding message sequence $\left\langle m_{\iota_1,\iota_2}, \ldots, m_{\iota_{\hat{h}},\iota_{\hat{h}+1}}, \ldots, m_{\iota_{n-1},\iota_n} \right\rangle$, where $\iota_1 = \iota_n = i$ and that there is $\iota_{\hat{h}} = j$, for some $1 < \hat{h} < n$. Such that: (i) The message m_{ι_1,ι_2} is sent after the* query-phase-start(π) *(resp.* prop-phase-start(π)*) event. (ii) Each message $m_{\iota_h,\iota_{h+1}}$ is sent after m_{ι_{h-1},ι_h} is received. (iii) The message m_{ι_{n-1},ι_n} is received before the* query-fix *(resp.* prop-fix *) event of π. (iv) If t is the value of the tag_j in any state before j sends $m_{\iota_{\hat{h}},\iota_{\hat{h}+1}}$, then: (a) $tag(\pi) \geq t$. (b) If π is a write operation then $tag(\pi) > t$. (v) If configs$(\ell)_j \neq \perp$ for all $\ell \leq k'$ (resp. $\ell < k'$) in any state before j send $m_{\iota_{\hat{h}},\iota_{\hat{h}+1}}$, then query-cmap$(\pi)(\ell) \in C$ (resp. prop-cmap$(\pi)(\ell) \in C$) for some $\ell \geq k'$.*

The remaining task is to show that the tag and the $configs$ information is propagated correctly during the query and propagation phases of the configuration upgrade operation. Meaning that at each node participating in the corresponding message sequence the following holds: (i) the tag information is non-decreasing and it at least as large as tag_i at the beginning of the operation, and (ii) that all configurations with identifier equal to and smaller than $upg\text{-}target_i$ are not $\perp$. In the interest of concise presentation, we will forgo of the detailed explanation as it is analogous to Lemma 4.1, where the departures are traceable to the difference between specification of the configuration upgrade operation and the read and the write operation.

Atomicity. We show atomicity using the framework of Lemma 13.16 in [15]. Recall that α is an arbitrary, good execution of the algorithm. It suffices to show that in α if all invoked the read and write operations complete, then these operations can be partially ordered by an ordering $\prec$ and the following properties are satisfied. *(P1)*: $\prec$ totally orders all write operations in α. *(P2)*: $\prec$ orders every read operation in α with respect to every write operation in α. *(P3)*: for each read operation, if there is no preceding write operation in $\prec$, then the initial value is returned by this read; else, the read operation returns the value of the unique write operation immediately preceding it in $\prec$. *(P4)*: if some operation, π_1, completes before another operation, π_2, begins in α, then π_2 does not precede π_1 in $\prec$. If such ordering $\prec$ can be constructed for α, then the algorithm guarantees atomicity.

We define $\prec$ in terms of the lexicographic order on tags of operations π. Observe that *(P1)* to *(P3)* are essentially immediate. Lemma 4.1 stated above and the additional lemmas presented in [16, 8, 5], which describe the behavior of configuration upgrade operation and read and write operations in any execution, are used to establish the monotonically increasing order on tags with respect to non-concurrent read or write operations. Based on the tags we define a partial order on operations and verify that property *(P4)* is enforced. Therefore, it follows immediately that the tags induce a partial order $\prec$ that meets the necessary and sufficient requirements for atomic consistency. Hence, the main result:

Theorem 4.2 ATILA *implements atomic read/write objects.*

5 Conditional Analysis of Operation Latency

In this section we examine the operation latency under similar timing assumptions as in the analysis of operations in RAMBO presented in [16, 8, 5, 7]. The novelty of our analysis as compared to the type of analysis done in [16, 8, 5, 7] is that here we use a more realistic assumption on the duration of message delivery. The previous analysis assumed that all messages were delivered within a fixed time interval; instead we assume a probability distribution on the delivery time of messages with finite variance.

ATILA is specified as a nondeterministic algorithm for asynchronous environments with arbitrary message delays and node crashes, departures, and new nodes joining. For the purpose of analysis, we restrict asynchrony, resolve the nondeterminism of the algorithm, and impose constraints sufficient to guarantee that the universe is connected.

Assumptions. Assume α is an admissible timed execution and α' a finite prefix of α. Let $\ell time(\alpha')$ denote the time of the last event in α'. Let α be a *timed admissible execution* then we say that α is an α'-$normal$ execution if no message sent in α after α' is lost, and if a message is sent at time t in α, it is delivered within bounded time (unknown to the participants). Moreover, we assume that each node sends messages at the first possible time and at regular intervals of d thereafter, as measured by the local clock, and each node will immediately send messages to all of its immediate neighbors following: receipt of a join request, new configu-

ration is discovered, and a message that indicates that phase information of any node has changed. Also, the non-send and locally controlled events occur just once, and are assumed to be instantaneous.

In the quorum-based algorithms operational liveness depends on all members of some quorum set remaining active. Let us denote by $t(c)$ the time at the end of the installation of configuration c, a time can always be specified by using the well-known axioms of time passage actions [15]. Also, let c' denote the next configuration that has been installed after c. Finally, we assume that configurations remain operational in order for reconfiguration to terminate, reconfigurations are not too frequent (say τd separated), only nodes that have joined the service may become part of the next configuration, and the deployment graph remains at least weakly connected.

Analysis. Now we provide analysis that estimates the duration of a read (resp. write) operation when reconfiguration is present. To make this estimate more realistic we provide minimum timing restrictions on spacing of certain events in the system and delays on message delivery. For the purpose of analysis, we assume certain probability distribution on the message delivery time. Unfortunately, such probability distribution may be difficult to determine for a complex algorithm like ATILA. However, under conditions when the rate at which nodes join, leave, or fail and the reconfiguration of the system, is not very high, we may estimate the mean or the standard deviation of message delivery delay.

In the analysis that follows, we consider a subgraph of the service deployment graphs induced by members of active configurations. Let D represent the diameter of this graph. Now, consider some non-failed quorum member, j, such that the length of the communication path between i and j is D. Note that new nodes may join the service at any time and at any active participant. If a new node joined only at j and is included as a member of a configuration installed next, D will increase. Therefore, we are interested in estimating the time required to complete a single phase of the read (resp. write) operation in a situation when new nodes join the service and become members of new configuration during the following reconfiguration attempt. Omitted details, are presented in [13].

Suppose that the mean time required for a message delivery between any two nodes is λ_A with finite variance ${\sigma_A}^2$ and the mean time of a new member being inducted into the quorum is λ_B and with finite variance ${\sigma_B}^2$. Also, we assume that $\lambda_A < \lambda_B$, i.e., on an average it takes less time for a message to reach its destination than it takes for the service to reconfigure, e.g., see RAMBO algorithms in [5, 8, 16].

The communication distance between i and some other node j is measured in terms of the length of the shortest path between i and j in the communication graph assuming each edge to be of unit length. Therefore, the delivery time of a point-to-point message is $\lambda_A = \frac{\lambda_B}{k}$, where λ_B is the average time needed to install a new configuration and typically, $1 \leq k \leq 12$. Since messages are propagated faster than new configurations are installed, read/write operations take $\frac{D\lambda_B}{\lambda_B - \lambda_A} = \frac{kD\lambda_A}{k\lambda_A - \lambda_A} = \frac{kD}{k-1}$ to complete with high probability (*whp*). We say that an event $\mathcal{E}$ occurs with whp to mean that $\Pr[\mathcal{E}] = 1 - O(n^{-\alpha})$ for some constant $\alpha > 0$.

The deterministic upper bound. Under assumptions stated above we consider the following worst case scenario. Let i be the node that initiates a read or a write operation. At the start of the operation, let j be the node farthest from i, this distance is at most the diameter of the service deployment graph at the time when i initiates its operation, this is referred to as the second pointer. Soon after i initiates its operation, new nodes join the service. The first new node connects to j and each new node may join at the last node that joined the service. In essence the nodes that joined the service form a path. By the *recon* spacing assumption a new node may become a member of the next configuration at least $12d$ time after it joined the service.

Theorem 5.1 *Let α be a α'-normal execution of the* ATILA *that satisfies 12d reconfiguration spacing then a read/write operation takes $O(N)$ time to complete since its invocation, where N is the number of nodes present at the time of invocation of the operation and $\tau > \epsilon N$, for some constant ϵ.*

6 Analysis of communication cost in ATILA

Now, we describe a scenario where the message bit cost complexity of ATILA is less than that of RAMBO and yet the necessary redundancy in the case of direct link failure is provided. The message bit cost complexity is the total cost of sending the individual bits across communication links. The RAMBO algorithm involves point-to-point perpetual dissemination of information which eventually helps to infer liveness of the protocol. However, such approach is obviously wasteful when nodes are separated by long geographical distances. A more reasonable solution to the above problem is to minimize communication over long distances, hence reducing the total message bit cost.

Consider the following grouping. Let the participants of the service be divided into m disjoint groups $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_m$ based on their proximity in terms of cost/reliability of communication among the nodes. For each group $\mathcal{G}_\iota$ we define

a non-empty subset, $\mathcal{L}_\iota$, to which we refer as the *representatives* of the group. Within a group nodes communicate using the all-to-all gossip protocol, however only groups' representatives may communicate with other representatives in the different groups. In this setting the indirect learning protocol allows a reduction of message bit cost complexity. Note that, the set of representatives may be agreed upon using an arbitrary consensus service, and handled in a similar fashion as ATILA applies reconfigurations. Also, in this deployment correctness is vacuously satisfied — we only impose a communication policy that restricts certain nodes from sending messages to certain other nodes.

We compare the communication cost complexities of the RAMBO and ATILA and show that the use of indirect gossip can lead to substantial cost savings. To simplify the analysis we assume that each group has size $|\mathcal{G}_\iota| = g$ and contains $|\mathcal{L}_\iota| = \ell$ representatives. The following equation compares the communication bit complexity per a single round of gossip in ATILA, left hand side, and RAMBO, right hand side.

$$g^2 m(\Delta + \delta N) + \ell \tfrac{m(m-1)}{2}\big(\Delta + \delta(N^2 + N)\big) + \ell(g - \ell)m\big(\Delta + \delta(N^2 + N)\big) \leq N^2(\Delta + \delta N) = O(N^3)$$

where Δ represents the constant size of the constant message components and δ is the size of a node identifier. The omitted analysis details are presented in [13].

7 Conclusions

We provide an algorithm that implements atomic read/write objects where the participating nodes communicate with their direct neighbors only, thus obviating the need for a global routing protocol. The indirect learning approach, as presented in this work, has the potential of making more robust other algorithms that, for example, employ all-to-all gossip as means for information exchange. The algorithmic development presented here is formally proved to guarantee atomicity in all executions. The indirect learning protocol allows operations to progress as long as the underlying network remains connected. We also presented a novel analysis of the operational latency under reasonable assumptions about the message delivery time. Lastly, we considered scenarios where our algorithm helps reduce messaging costs.

References

[1] Special issue on group communication services. *Communications of the ACM*, 39(4), 1996.

[2] J.-C. Bermond, L. Gargano, A. A. Rescigno, and U. Vaccaro. Fast gossiping by short messages. In *Automata, Languages and Programming*, pages 135–146, 1995.

[3] S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. Geoquorums: Implementing atomic memory in ad hoc networks. In *Proc. of 17th International Symposium on Distributed Computing*, pages 306–320, 2003.

[4] S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks, 2003.

[5] C. Georgiou, P. Musial, and A. Shvartsman. Long-lived RAMBO: Trading knowledge for communication. In *Proc. of 11th Colloq. on Structural Information and Communication Complexity*, pages 185–196, 2004.

[6] C. Georgiou, P. Musial, and A. Shvartsman. Developing a consistent domain-oriented distributed object service. In *Proc. 4th IEEE Int-l Symposium on Network Computing and Applications*, pages 149–158, 2005.

[7] S. Gilbert. RAMBO II: Rapidly reconfigurable atomic memory for dynamic networks. Master's thesis, MIT, 2003.

[8] S. Gilbert, N. Lynch, and A. Shvartsman. RAMBO II: Rapidly reconfigurable atomic memory for dynamic networks. In *Proc. of International Conference on Dependable Systems and Networks*, pages 259–268, 2003.

[9] V. Gramoli, P. Musiał, and A. Shvartsman. Operation liveness in a dynamic distributed atomic data service with efficient gossip management. In *Proc. 18th International Conference on Parallel and Distributed Computing Systems*, 2005.

[10] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Kluwer Academic*, 1996.

[11] I. Keidar, J. B. Sussman, K. Marzullo, and D. Dolev. Moshe: A group membership service for wans. *ACM Trans. Comput. Syst.*, 20(3):191–238, 2002.

[12] S. Khuller, Y. Kim, and Y. Wan. On generalized gossiping and broadcasting, 2003.

[13] K. Konwar, P. Musial, N. Nicolaou, and A. Shvartsman. Implementing atomic data through indirect learning in dynamic networks. Technical Report MIT-CSAIL-TR-2006-070, MIT, 2006.

[14] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[15] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[16] N. Lynch and A. Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In *Proc. of 16th International Symposium on Distributed Computing*, pages 173–190, 2002.

[17] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. Technical report, MIT, 1987.

[18] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proc. of IEEE INFOCOM*, 1997.

[19] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. In *Proc. of ACM SIGCOMM*, 1994.

[20] C. E. Perkins and E. M. Royer. Ad hoc on-demand distance vector routing. In *Proc. of IEEE WMCSA*, 1999.