LRTG: Scheduling Distributed Real-Time Tasks in Unreliable and Untrustworthy Systems

Kai Han*, Binoy Ravindran*, and E. D. Jensen‡

*ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
{khan05,binoy}@vt.edu

‡The MITRE Corporation
Bedford, MA 01730, USA
jensen@mitre.org

Abstract

We consider scheduling distributed real-time tasks in unreliable (e.g., those with arbitrary node and network failures) and untrustworthy systems (e.g., those with Byzantine node behaviors). We present a distributed real-time scheduling algorithm called LRTG. The algorithm makes two novel contributions. First, LRTG uses gossip for reliably propagating task scheduling parameters and for discovering task execution nodes. Second, the algorithm guards against potential disruption of message propagation due to Byzantine attacks using a mechanism called LASIRC. By doing so, the algorithm provides assurances on task timeliness behaviors, despite system unreliability and untrustworthiness. Our performance evaluation shows LRTG's effectiveness.

I. Introduction

Some emerging, large-scale distributed real-time systems are envisioned to use unreliable network infrastructures— e.g., those without a fixed network infrastructure, including mobile and wireless networks. In such infrastructures, frequent and arbitrary message losses and node failures make directed communication inefficient. Furthermore, as the number of nodes increases, keeping track of routes or link states becomes less feasible. Despite such communication uncertainties, applications desire as strong assurances on end-to-end task timeliness behaviors as possible.

An example large-scale, distributed real-time system application context that motivates our work is the U.S. DoD's information age warfare transformation vision called Network-Centric Warfare (NCW) [1]. A ship may detect a threat in the airspace, or be warned of it by another ship, an aircraft, or a satellite. The threatened ship may be unable to prosecute the threat due to current limitations of its weapon systems (e.g., lack of suitable weapon). Consequently, prosecution of the threat may be assigned to a platform in the vicinity (e.g., another ship, or an aircraft), which may launch a weapon. Yet another platform in the vicinity may guide the weapon to the target until the engagement is complete. The pattern of interactions is peer-to-peer, departing from the hierarchical interaction pattern that has dominated traditional battle management/command and control (BM/C2) operations. Threat detection, identification, tracking, weapons assignment, and weapon guidance activities have time constraints, and the most important ones must be satisfied despite message losses and component failures, to achieve acceptable mission measures of effectiveness.

Further exacerbating the end-to-end real-time resource management challenge in such systems is the possibility of malicious node behaviors due to malicious insiders – infiltrated adversaries or traitors who are authenticated and know encryption codes, etc. Such behaviors may include disruption of communications, and transmission of false messages. Such malicious behaviors are referred to as Byzantine ones [2]. A node exhibiting Byzantine behaviors is called a Byzantine node. Byzantine nodes are more difficult to deal with — in any authentication process, they act just like other nodes [3]. Therefore, a "healthy" (i.e., legitimate) node cannot trust its peers — it does not know whether another one is a friend, a traitor, or an adversary. Further, Byzantine nodes may know how Byzantine nodes are detected, and may be intelligent in the sense that if they cannot protect themselves from being detected, they will not attack – i.e., behave maliciously. In this paper, we use "Byzantine nodes" and "Byzantine attackers", interchangeably.

In this paper, we present an integrated solution called *LASIRC-Aided Real-Time Gossip* (or LRTG) that provides probabilistic (end-to-end) timeliness assurances in such unreliable and untrustworthy systems. LRTG includes two parts: a gossip-based, Byzantine-tolerant message propagation model/mechanism called LASIRC, and a gossip-based distributed real-time scheduling algorithm called RTG. Gossip has its origins in replicated data management [4], was pioneered in [5] for reliable multicast in wired networks, has been

used to solve a variety of problems — examples include information dissemination [6] and reliable multicast [7]. End-to-end real-time scheduling has been studied in the past (e.g., [8], [9]), but these are limited to fixed (and reliable) network infrastructures. Support for timing assurances in unreliable systems are considered in [10], [11], but they do not provide end-to-end task timeliness assurances or consider untrustworthy systems.

Our work builds upon our prior work in [11], [12], which presents early version of RTG and LASIRC, respectively. However, LRTG significantly distinguishes itself from the above two—it completely changes the core component, gossip protocol, with peer nodes continuously sending messages instead of sending only once during message propagation processes. Besides, its gossip has much lower and controllable message overhead that RTG-DS does not provide. For the first time, LRTG gives LASIRC mechanism real-time properties which is essential in real-time scheduling. Besides, LRTG illustrates firm theoretical analysis for LASIRC, which is not presented before. Therefore, we argue that LRTG is not a simple integration of our former work, but a completely redesigned algorithm to deal with the more complicated scheduling environment—i.e., the unreliable and untrustworthy distributed systems.

The rest of the paper is organized as follows: In Section II, we discuss models and algorithm objectives. Section III illustrates our gossip-based task scheduling strategies. We discuss possible Byzantine attacks in gossip protocols in Section IV. We then present our Byzantine attacker detectors in Section V. Sections VI describe the LASIRC model and the mechanism. We present and analyze LRTG algorithm in Section VII and VIII, respectively.In Section IX, we report on our experimental (simulation) studies. We conclude the paper in Section X.

II. Models and Algorithm Objectives

A. Task Model

We model a distributed task as being composed of a sequence of subtasks or *sections*, where a section constitutes the portion of the task's execution on a node. If the task models a series of nested, remote method invocations, then a section constitutes a maximal length sequence of contiguous method executions on a node. If the task models a series of chained, publication and subscription events, then a section constitutes the execution of a subscription/publication service on a node.

We call the initial section of a task as the task's *root*. The node hosting the root is called the task's *root node*. A task's most recent section — i.e., the task's current execution locus — is called the task's *head*, and the node hosting the *head* is called the task's *head node*.

We assume that the number of sections of a task is known. Also, the execution time estimates of the sections are also assumed to be known.

The application is thus comprised of a set of tasks, denoted $\mathbf{T} = \{T_1, T_2, \ldots\}$.

B. Timeliness Model and Utility Accrual Scheduling

Each task's time constraint is specified using a time/utility function (or TUF) [13]. A TUF specifies the utility of completing a task as a function of its completion time. Figure 1 shows example step TUFs.

A task T_i's TUF is denoted as $U_i(t)$. In this paper, we focus on downward step TUFs, and denote the maximum, constant utility of a TUF $U_i()$, simply as U_i. Each TUF has an initial time I_i, which is the earliest time for which the TUF is defined, and a termination time X_i, which, for a downward step TUF, is its discontinuity point. $U_i(t) > 0, \forall t \in [I_i, X_i]$ and $U_i(t) = 0, \forall t \notin [I_i, X_i], \forall i$.

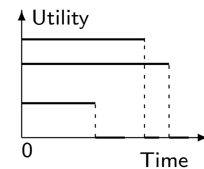


Fig. 1. Step TUFs

C. System Model

The system consists of a set of processing components, generically referred to as *nodes*, denoted $N = \{n_1, n_2, \ldots n_\alpha\}$. Nodes may dynamically join or leave the network, thus forming an ad hoc network as in [1]. Node clocks are synchronized using an algorithm such as [14]. We assume that the network communication delay follows some non-negative probability distribution—e.g., the Gamma distribution as in [15]. Nodes may fail by crashing, and messages may be lost, both arbitrarily. Besides, some nodes may exhibit Byzantine behaviors, as indicated in Section I and IV.

D. Objectives

Our goal is to schedule tasks with probabilistic termination-time satisfactions — i.e., establish probabilistically satisfied end-to-end timing assurance for a task. Further, we desire to maximize the sum of the tasks' attained utilities, and minimize the number of tasks that miss termination times.

III. LRTG Rationale

Since nodes may crash and messages may be lost, task scheduling must (dynamically) recognize and

reconcile with system dynamics. For example, once a task completes at a node, referred to as the task's *current* head node, the next node to which task execution progresses, referred to as the task's *next* head node, may crash, or may become unreachable. Thus, we take the approach of the current head node dynamically "discovering" the next head node, once execution completes at the current head node.

We consider a gossip-style protocol for this discovery, where the current head node randomly selects a set of peer nodes and multicasts task scheduling parameters in REQUEST (REQ) messages, for a finite number of synchronous gossip rounds. Upon receiving a REQ message, a peer node repeats the gossip process, if it is not the next head node. Otherwise, it determines the feasibility of executing the next task section and replies back with its decision in REPLY (REP) messages. The current head node waits for a decision, but adds a deadline. If it does not receive a reply within that deadline, it will consider the next head node as crashed or unreachable and the task is regarded as failed.

Gossip-based algorithms offer a scalable, robust, fault-tolerant, and probabilistically-reliable message propagation design paradigm for large-scale, unreliable systems — informed nodes simply "gossip" to randomly selected targets, without requiring any confirmation regarding message reception. Despite these attractive features, these algorithms incur relatively high message overheads. However, for message propagation in large-scale, unreliable networks, this problem is not that serious. First, the randomness nature of gossip reduces the (lower-layer) overhead for gathering, storing, and updating massive amounts of information in a vast network—e.g., gossip makes nodes update their link state tables less frequently. Second, gossip is robust against a class of Byzantine attacks (e.g., black hole attacks [2]).

IV. Byzantine Attacks in Gossip Protocols

A. Byzantine Attack Types

If a node receives but does not forward gossip messages, the behavior is referred to as a *Black Hole* attack. In addition, two or more attackers may collude together to form a larger "*Black Hole*". We call these two attacks as *Black Hole Class (BHC)* attacks.

A *Message-Faking (MF)* attack is more harmful, since it propagates incorrect information, trying to mislead others to make a wrong decision. For instance, a node may send a REQ message requesting a remote service. If an MF attacker replies with a "NO" instead of the correct answer "YES", the sender will incorrectly abort a waiting task. If the attacker replies a "YES" for a "NO", the sender has to keep the ineligible task and hold all its locked resources, thereby delaying the execution of eligible tasks, which degrades timeliness optimality.

B. Gossip Message Structures

A REQ message contains a task's end-to-end scheduling parameters, the original sender's (the current head node) identifier (ID), the selected target node identifier (ID), the requested service ID (SID), the set of gossip fan out numbers (F_r), and the number of gossip rounds (R). A REP message contains the original sender's (the next head node) ID, the selected target node's ID, SID, the set of F_r, R, and the scheduling decision ("YES" or "NO").

C. Byzantine Attacks in Message Propagation

BHC attackers stop forwarding messages, trying to slow or even cease message propagation. They are easy to deal with, since gossip is robust to message losses.

For MF attacks in REQ message propagation, if an MF attacker spreads a fake SID, it can activate an irrelevant receiver to gossip back, and thus increase the message overhead in the network. If the attacker spreads a fake REP message, it may tempt the sender to give wrong responses. An important feature of gossip is that every node will eventually receive the (attacker's fake) messages, with a high probability. If the real next head node gets a fake REP message, it may easily identify the attacker (by comparison with the original sender's ID). Thus, "intelligent" attackers will not initiate such attacks in REQ message propagation.

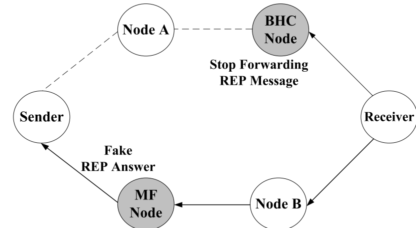


Fig. 2. Byzantine Attacks in REP Message Propagation

Figure 2 shows BHC and MF attacks in REQ and REP propagation, respectively. Unlike in REQ message propagation, an "intelligent" attacker can fake REP messages in REP message propagation without being identified by others. Though the real next head node can finally receive fake REP messages, it cannot identify the initiator (the original

sender is itself). It is quite possible that the direct sender is an intermediate node, which is a victim cheated by the MF attacker.

V. Byzantine Attacker Detection

Gossip-based algorithms are robust to BHC attacks. In this section, we focus on designing MF Attacker Detectors (MFADs).

MFAD utilizes MF attackers' feature of faking SID in REQ message propagation, or faking the answer in REP message propagation. To initiate MF detection, a healthy node (a node that is not an MF attacker) must run its own MFAD before others.

Algorithm 1: Initiating MF Attacker Detector

1. Node i puts its own SID in "REQ" messages;
2. Node i sets $R = 1$;
3. At 1st gossip round: Broadcast "REQ" messages;
4. After 1st gossip round: Check SID in every received "REQ" message;
5. **if** *SID is changed* **then**
6. Identify this message sender as an MF attacker;

To detect MF attackers in REQ message propagation, the initiator broadcasts REQ messages, but includes its own SID and sets the number of gossip rounds R to 1. MF attackers will change this SID. Since $R = 1$, an attacker must broadcast fake REQ messages to all other nodes. Therefore, the initiating MFAD can easily identify an MF attacker by checking the SID in every received REQ message. For activated MFADs on other healthy nodes, since they receive the initial REQ message, they can identify MF attackers by comparing every received REQ with the one from the initiator. MFADs are described in Algorithms 1 and 2.

Algorithm 2: Activated MF Attacker Detector

1. At the first gossip round: Receive "REQ" messages from the initiating MFAD;
2. At the second gossip round: Broadcast "REQ" messages once;
3. After the second gossip round: Compare service ID in every received "REQ" message with the one in the original message;
4. **if** *the service ID is changed* **then**
5. Identify this message sender as an MF attacker;

To detect MF attackers in REP message propagation, the initiator broadcasts REP messages, includes its preset answer (e.g., "YES"), and sets $R = 1$. MF attackers will reverse the answer (e.g., "NO"). Since $R = 1$, an attacker must broadcast fake REP messages to all other nodes. Similar to REQ MFADs, both initiating and activated MFAD can easily identify an MF attacker. Since these two detection processes are similar, we do not separately show REP MFADs.

If an activated MFAD does not receive the original REQ or REP message from the initiator, it cannot identify any MF attacker later. If an MFAD does not receive a one-time-broadcast REQ or REP message from another node, it cannot regard that node as an MF attacker. Therefore, the effectiveness of MFADs depends on the message loss ratio.

VI. LASIRC Model and Mechanism

MFADs cannot exhaust MF attackers if the message loss ratio is larger than zero, which is common in unreliable networks. Since gossip is robust to message losses and node failures, it is relatively easy to deal with hiding BHC attackers. However, gossip cannot handle hiding MF attackers, which is more dangerous. Therefore, it is necessary to design a gossip-based propagation model/mechanism to defend MF attacks.

A. LASIRC Model

As described in Section IV-B, it is easy to identify an MF attacker that fakes SID in REQ message propagation. In addition, another MF attack increases communication overhead, but it does little harm to the real REQ message propagation. Therefore, we focus on modeling node behaviors in REP message propagation. We first introduce new definitions for nodes participating in REP message propagation:

Definition 1 (Healthy Node)**.** *A node that is not an MF attacker.*

Definition 2 (Host (H))**.** *A node that has received one or more REP messages.*

Definition 3 (Launcher (L))**.** *The initiating sender of the REP messages.*

Definition 4 (Attacker (A))**.** *An MF attacker that tries to spread a fake answer.*

Definition 5 (Susceptible (S))**.** *A healthy node that has not received any REP message.*

Definition 6 (Infective (I))**.** *A healthy host that has a fake answer.*

Definition 7 (Removed (R))**.** *A healthy host that knows the correct answer, or always sends correct REP messages.*

Definition 8 (Consumer (C))**.** *The initiating sender of REQ messages.*

We refer to this model as the LASIRC model, integrating the six actor's acronyms from Definitions 3–8.

B. LASIRC Mechanism

Algorithm 3: Launcher

```
1 Initialize REP message rep;
2 GOSSIP(rep);
```

We now describe the LASIRC mechanism. Algorithm 3 shows how a launcher works. A launcher is a next head node. It holds the correct answer, so it cannot be infected by an MF attacker. The GOSSIP() procedure invoked in line 2 of Algorithm 3 is shown in Algorithm 4.

An MF attacker is activated when receiving a REP message. If the sender is another attacker, it follows the answer. Otherwise, it reverses the answer (e.g., "NO" for a "YES"). Algorithm 5 describes MF attackers.

Algorithm 4: Gossip Emission [GOSSIP() Function]

```
1 On gossiping a message msg:
2   while R != 0 do
3       Every gossip round Γ, randomly select F targets;
4       R = R – 1;
5       for each m ∈ [1, . . ., F ] do
6           SEND(target_i, msg);
```

Algorithm 5: MF Attacker

```
1 On receiving a REP message rep:
2   if the sender is not an MF attacker then reverse
    the answer in rep;
3   GOSSIP(rep);
```

Algorithm 6 shows healthy node behaviors. A susceptible turns into a removed if its first received message is an antibiotic (from a removed or an identified attacker), or turns into an infective if that message is a virus (from an infective or an MF attacker).

Consumer behavior in LASIRC is shown in Algorithm 7. A consumer behaves like other healthy nodes during gossip rounds. After gossip finishes, if it still cannot identify itself as a removed (knowing the correct answer), it will count the number of the same answers in received REP messages. If the consumer is optimistic, it will regard that MF attackers occupy less than half of the total number of nodes. Thus, it will select the answer in most received REP messages. Otherwise, the consumer is pessimistic, and it will select the answer in less received REP messages.

Algorithm 6: Susceptible, Infective, and Removed

```
1 On receiving the first REP message rep:
2   GOSSIP(rep); //Become a Removed if rep is a
    Vaccine; become an Infective if rep is a Virus
3 On receiving another REP message with the same
  message ID:
4   if the sender has sent REP message before then
5       if the answer changes then
6           adopt answer in new REP message;
            //Identify the sender has changed from an
            infective to a removed
7           reverse the answer in rep;
8           GOSSIP(rep); //Change from an infective
            to a removed
9   if the sender is an identified MF attacker then
10      if the answer in the first rep is the same as the
        one in this attacker's message then
11          reverse the answer in rep;
12          GOSSIP(rep); //Change from an infective
            to a removed
```

Algorithm 7: Consumer

```
1 In gossip: Act as other healthy nodes do in Algorithm 6
2 After gossip finishes:
3   if the consumer has not identified itself as a removed
    then
4       select the answer in most(less) received REP
        messages; //Optimistic (Pessimistic) consumer
```

VII. The LRTG Algorithm

The LRTG algorithm follows three steps to schedule tasks in unreliable distributed systems with possible Byzantine attacks. These steps include 1) building local TUF, 2) constructing local schedule, and 3) determining a task's next head node.

A. Building Local TUF

LRTG decomposes a task's end-to-end TUF into local TUFs for the task sections, based on the execution time estimates of the sections and the task's termination time. Let a task T_i arrive at a node n_j at time t. Let T_i's total execution time of all the remaining task sections (including the local section on n_j) be Er_i, the total remaining slack time be Sr_i, the number of remaining task sections (including the local section on n_j) be Nr_i, and the execution time estimate of the local section be $Er_{i,j}$. LRTG computes a local slack time $LS_{i,j}$ for T_i :

$$LS_{i,j} = \begin{cases} \frac{Sr_i}{Nr_i - 1} & Nr_i > 1 \\ Sr_i & 0 \leqslant Nr_i \leqslant 1 \end{cases} \quad (1)$$

The algorithm computes $LS_{i,j}$ in a way that allows the remaining task sections to have a fair chance to complete. The network communication delay incurred by LRTG for the gossip rounds must be limited to at most $LS_{i,j}$.

The local termination time for a task T_i is given by $LX_{i,j} = t + Er_{i,j} + LS_{i,j}$. The local termination

time is used by the algorithm to test for schedule feasibility, while constructing local task section schedules.

B. Constructing Local Schedule

Scheduling sections on a node to maximize the total accrued utility is NP-hard [16]. Thus, we consider heuristics called Potential Utility Density (or PUD). For a section S_i at time t, its PUD is given by $PUD_i(t) = U_i/El_{ri}(t)$, where $El_{ri}(t)$ is S_i's remaining local execution time at t. Thus, a section's PUD measures its return on "time investment." LRTG constructs local schedules at two scheduling events: 1) a message arrival that signals the release of a section (or a section's handler) for execution; and 2) completion of a section's execution.

When invoked, the algorithm first sorts all sections in the non-increasing order of their PUDs. The sorted sections are then examined, highest PUD first, and inserted into a tentative schedule. The tentative schedule is maintained in the order of non-decreasing section termination times, and tested for feasibility. A schedule is said to be feasible, if the predicted completion time of each section in the schedule does not exceed its local termination time. If the schedule is infeasible, the inserted section is removed. The process is repeated until all sections are examined. The section with the earliest termination time first executes. Algorithm 8 describes this procedure.

Algorithm 8: Local LRTG Scheduling Algorithm [Local_SCHEDULE( )]

```
1 Create an empty schedule φ;
2 Let t be the time of the scheduling event;
3 Sort sections in ready queue according to PUDs;
4 for each section in decreasing PUD order do
5     Insert section in φ at its termination time position
      (maintaining φ's increasing termination-time
      order);
6     if schedule is infeasible then
7         Remove section from φ;
8 Select earliest-deadline section from σ for execution;
```

C. Determining the Next Head Node

Definition 9. *Gossip Round r: Denotes the rth gossip time interval, at the beginning of which nodes send messages. All messages are assumed to arrive at their destination nodes when the round r ends, with a high probability.*

We assume that the message delay follows a non-negative distribution, e.g., the Gamma distribution, as in [15]. Many distributions have infinite tails, and therefore, to determine the length of a gossip round, application users need to decide a termination time point t_{end}, after which message arrivals can be ignored. This is done by determining a threshold on the message arrival ratio, which is referred to as Θ. For instance, if $\Theta = 98\%$, we can determine the relative t_{end} in a given distribution. The length of a gossip round is then equal to the time interval between the round start time point (the value is often 0) and t_{end} in the distribution function.

Definition 10. *I_r : Denotes the number of total informed nodes at the end of gossip round r.*

Definition 11. *U_r : Denotes the number of uninformed nodes at the end of gossip round r.*

Definition 12. *Fan out, F_r: The number of messages a node sends to target nodes at the beginning of round r.*

Let N denote the total number of nodes in the system. We compute the expected number of uninformed nodes at the end of gossip round r as:

$$U_r = U_{r-1} \times (1 - \frac{F_r}{N-1})^{I_{r-1}} \quad (2)$$

When $F_r \ll N - 1$, we have:

$$U_r = U_{r-1} \times \exp\left(\frac{-F_r \times I_{r-1}}{N-1}\right) \quad (3)$$

The fan out and the number of messages issued during gossip round r (M_r), are given by:

$$F_r = \frac{N-1}{I_{r-1}} \times \ln\left(\frac{U_{r-1}}{U_r}\right) \quad (4)$$

$$M_r = F_r \times I_{r-1} = (N-1) \times \ln\left(\frac{U_{r-1}}{U_r}\right) \quad (5)$$

Different from gossip protocols with fixed fan out number at each round, here, F_r can be adjusted on an application-specific basis.

In gossip protocols, a message is supposed to be sent at the beginning of a round, and arrive at its destination before the end of the same round (with a high probability). This message should not be counted in the next round. Thus, the number of messages existing at the same time is much less than the total number of messages. In addition, randomly selecting gossip targets uniformly distributes the messages across the network, and thereby reduces the likelihood for network congestion.

VIII. Algorithm Analysis

Let Δt denote the time length of a gossip round, and ℓ_1, ℓ_2, and ℓ_3 denote the number of successful contacts made by one infective, attacker, and host per time unit, respectively. We have:

$$\ell_1 = \frac{\alpha\beta F}{\Delta t}, \quad \ell_2 = \frac{\alpha\beta F(1-\eta)}{\Delta t}, \quad \ell_3 = \frac{\alpha F}{\Delta t}$$

where α is the message loss rate, β indicates whether a contact message is the first arrival, η is the immunity rate, and F is the gossip fan out number.

Let γ_1, γ_2, and γ_3 denote the number of successful immunized nodes made by one removed, launcher, and attacker per time unit, respectively. We have:

$$\gamma_1 = \frac{\alpha\beta F}{\Delta t}, \quad \gamma_2 = \frac{\alpha F}{\Delta t}, \quad \gamma_3 = \frac{\alpha\beta F\eta}{\Delta t}$$

Let N_1 and N_2 denote the number of attackers and healthy nodes, respectively. Let S_n, I_n, and R_n denote the number of susceptibles, infectives, and removed after gossip round n, respectively. Let $\underline{I}_n$ and $\underline{S}_n$ denote the number of activated and non-activated attackers, respectively. The discrete-time deterministic LASIRC model is:

$$S_{n+1} = S_n \times \left(1 - \frac{\ell_1 \Delta t I_{+} \ell_2 \Delta t \underline{I}_n + \gamma_1 \Delta t R_n + \gamma_2 \Delta t + \gamma_3 \Delta t \underline{I}_n}{N}\right) \tag{6}$$

$$\underline{S}_{n+1} = \underline{S}_n \left(1 - \frac{\ell_3 \Delta t \left(I_n + R_n + \underline{I}_n\right)}{N}\right) \tag{7}$$

$$I_{n+1} = I_n \left(1 - \frac{\gamma_2 \Delta t + \gamma_3 \Delta t \underline{I}_n}{N}\right) + \left(\frac{\ell_1 \Delta t I_n S_n + \ell_2 \Delta t \underline{I}_n S_n}{N}\right) \tag{8}$$

$$\underline{I}_{n+1} = \underline{I}_n + \left(\frac{\ell_3 \Delta t (I_n + R_n + \underline{I}_n) \underline{S}_n}{N}\right) \tag{9}$$

$$R_{n+1} = R_n + \frac{1}{N} \times \Big(\gamma_1 \Delta t R_n S_n + \gamma_2 \Delta t S_n + \gamma_3 \Delta t \underline{I}_n S_n + \gamma_2 \Delta t I_n + \gamma_3 \Delta t I_n \underline{I}_n\Big) \tag{10}$$

It is clear that $N_1 = \underline{S}_n + \underline{I}_n$, $N_2 = S_n + I_n + R_n$, and $N = N_1 + N_2$ for all time.

Theorem 1. $\lim_{n\to\infty} \underline{S}_n = 0$*,* $\lim_{n\to\infty} \underline{I}_n = N_1$*.*

Proof: Let $g(\underline{S})$ denote the right side of $\underline{S}_{n+1}$ in (7):

$$g(\underline{S}) = \underline{S}\left(1 - \ell_2 \Delta t \frac{\underline{I} + I + R}{N}\right) \tag{11}$$

Note that:

$$g'(\underline{S}) = 1 - \ell_2 \Delta t \left(\frac{N_1 + I + R}{N}\right). \tag{12}$$

Thus, $g'(\underline{S}) < 1$ or $g(\underline{S}) < \underline{S}$ for $\underline{S} \in (0, N_1]$. It follows that $\underline{S}_n$ is a strictly decreasing sequence bounded below by zero and must approach a fixed point of g on $[0, N_1]$. The only fixed point of g on $[0, N_1]$ is $g(0) = 0$; hence, $\lim_{n\to\infty} \underline{S}_n = 0$. Since $\underline{I}_n = N_1 - \underline{S}_n$, we have $\lim_{n\to\infty} \underline{I}_n = N_1$. ∎

Theorem 2. *$lim_{n\to\infty} S_n = 0$, $lim_{n\to\infty} I_n = 0$, $lim_{n\to\infty} R_n = N_2$.*

Proof: From Definition 5, $lim_{n\to\infty} S_n = 0$. Let $h(I)$ denote the right side of I_n in (8):

$$h(I) = I\left(1 - \frac{\gamma_2 \Delta t + \gamma_3 \Delta t \underline{I}}{N}\right) + \frac{\ell_1 \Delta t S I}{N} + \frac{\ell_2 \Delta t S \underline{I}}{N}. \tag{13}$$

Note that:

$$h'(I) = 1 + \frac{\ell \Delta t (S - I)}{N} - \frac{\gamma_2 \Delta t}{N} - \frac{\gamma_3 \Delta t \underline{I}}{N} - \frac{\ell_2 \Delta t \underline{I}}{N} \tag{14}$$

$$h''(I) = -\frac{\ell \Delta t}{N} \tag{15}$$

Case 1. When $I = 0$, if $h'(I) > 1$, then $h(I) > I$. It follows that I_n is initially an increasing sequence. Since $h''(I) < 0$, $h'(I)$ strictly decreases to be 1 and then less than 1. Thus, $h(I)$ stops increasing at some point on $[0, N_2]$, and then strictly decreases to a fixed point on $[0, N_2]$. Suppose, this fixed point is larger than 0. Then, R_n will increase to infinity, which is a contradiction. Thus, fixed point is 0.

Case 2. When $I = 0$, if $h'(I) \leq 1$, then $h(I) \leq I$. Since $h''(I) < 0$, I_n is a strictly decreasing sequence to the fixed point zero. There is no infection.

From *Cases 1* and *2*, $lim_{n\to\infty} I_n = 0$. Thus, $lim_{n\to\infty} R_n = N_2$. ∎

Theorem 3. *If all nodes are underloaded and therefore, the section schedules constructed at all nodes are feasible (Section VII-B), then LRTG probabilistically bounds task termination time satisfactions.*

Proof: Let a task execute through $m+1$ nodes. Since each node is underloaded, each task section can be successfully finished. The probability of satisfying task termination times is the probability of successfully completing all communication processes. Determining the next head node includes two steps. First, the head node gossips REQ messages. Second, the next head node gossips REP messages back.

The message REQ propagation follows the SI (Susceptible-Infective) model in epidemiology [17]. Let a node that receives a REQ message be an infectious individual. From the SI model:

$$\widetilde{S}_{n+1} = \widetilde{S}_n - \lambda_n \Delta t \widetilde{S}_n \tag{16}$$

$$\widetilde{I}_{n+1} = \widetilde{I}_n + \lambda_n \Delta t \widetilde{S}_n \tag{17}$$

where $\widetilde{S}_n$ and $\widetilde{I}_n$ are the number of non-infectious individuals and infectious individuals at round n, respectively, and λ_n is the force of infection [17].

The second step follows the LASIRC model. The number of removeds is:

$$R_{n+1} = R_n + \frac{1}{N}\Big(\gamma_1 \Delta t R_n S_n + \gamma_2 \Delta t S_n + \gamma_3 \Delta t \underline{I}_n S_n + \gamma_2 \Delta t I_n + \gamma_3 \Delta t I_n \underline{I}_n\Big) \tag{18}$$

Let p be the largest number of rounds that a head node must wait for a reply. Let p_1 and p_2 be the number of rounds needed to notify the next head node and receive a correct reply, respectively ($p_1 + p_2 = p$). A section k's probability to satisfy its time constraint is:

$$p_{s_k} = \frac{\widetilde{I}_{p_1}}{N} \frac{R_{p_2}}{N_2} \tag{19}$$

Clearly, $p_1 \geq 0$ and $p_2 \geq 0$. Thus, the probability P_{Sd} for a task d to successfully complete through $m+1$ head nodes, and that for a task set D, P_{SD}, is given by:

$$P_{S_d} = \prod_{1 \leq k \leq m} p_{s_k} \qquad P_{S_D} = \prod_{d \in D} P_{S_d} \tag{20}$$

■

IX. Experimental Studies

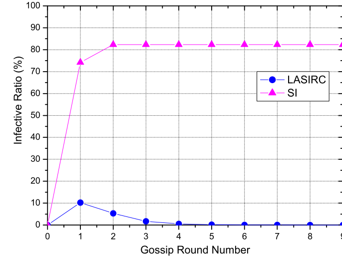
A. Effectiveness of LASIRC Mechanism

In this simulation study, we focused on Byzantine-tolerant properties. We simulated the LASIRC mechanism in a 100-node system, in which every node is reachable to others.

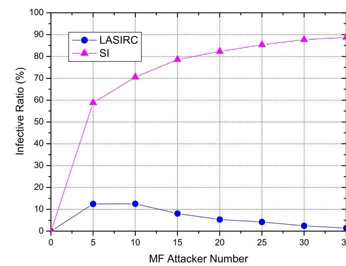
LASIRC is robust to BHC attacks. For MF attacks, we compared LASIRC with the Susceptible-Infective (SI) mechanism, which represents the common gossip protocol without considering MF attackers. MFADs cannot detect all MF attackers if MLR is larger than 0. Therefore, we need the LASIRC mechanism to deal with the remaining MF attackers after MFAD detection. The number of rounds (R) and the fan out number (F) was 10. Since LASIRC is combined with LRTG, our attained result also holds under different F_r patterns and larger systems.

Figure 3 shows Infective Ratio (IR; the ratio of number of infectives to the number of healthy nodes) along with the MF attacker number (N) and R, respectively. In Figure 3(a), We observe that as R increases, at the first round, LASIRC's IR moderately increases from 0 to 12.84%, while SI's IR dramatically increases from 0 to 74.52%. This is because, LASIRC MFADs enable nodes to identify a number of MF attackers. In comparison, SI does not have MFADs, and nodes immediately get infected. In later rounds, we observe that LASIRC's IR quickly decreases to near zero ($IR = 0.01\%$ when $R = 9$), while SI's IR almost remains unchanged ($IR = 82.35\%$ when $R = 9$). With Algorithms 6 and 7, LASIRC turns infectives into removeds. SI does not have such algorithms, so its number of infectives keeps increasing till there is no susceptible.

In figure 3(b), we observe that SI's IR dramatically increases as N increases ($IR = 88.76\%$ when $N = 35$). However, LASIRC's IR slightly increases and then decreases to almost zero ($IR = 1.07\%$ when $N = 35$). This shows the effectiveness of LASIRC MFADs — if there are more MF nodes, MFADs identify more MF attackers. This is why LASIRC's IR decreases when N increases.



(a) Infective Ratio (20 Attackers)



(b) Infective Ratio(2nd Round)

Fig. 3. Infective Ratio under LASIRC/SI Model

B. Overall Performance Evaluation

We evaluated LRTG's overall performance in a 700-node unreliable system, and compared it with the gossip-based algorithm RTG-DS in [11]. RTG-DS does not change F_r during gossip. Besides, it does not use the LASIRC mechanism.

In the previous experiments, we verified that LRTG is robust to message losses, node failures, and BHC attacks. We focused on dealing with MF attacks in this section. We set tasks to execute through four nodes. In Figure 4, we observe that when the number of MF attackers increases, LRTG's SR gently decreases. At the same time, RTG-DS's SR decreases in a much more dramatic way — if there are 15 MF attackers in the system, RTG-DS's SR drops to 20.57%, while LRTG's SR is still above 90%. Thus, even if MF attackers only occupy a small ratio in a large-scale system, it can have a significant impact on scheduling performance. However, with LASIRC, LRTG can effectively deal with MF attackers.

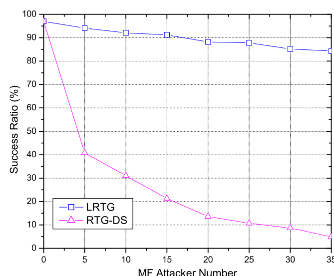


Fig. 4. Comparison between LRTG and RTG-DS

X. Conclusions

LRTG uses gossip for reliably propagating task scheduling parameters and for discovering task execution nodes, despite message losses and node failures, with acceptable message overheads. The algorithm's LASIRC mechanism guards against potential disruption of message propagation due to Byzantine attacks including BHC and application-level message-faking attacks. We show that LRTG provides probabilistic assurances on timeliness behaviors. Our experiments shows its effectiveness.

References

[1] J. A. Freebersyser et al., "Realizing the network-centric warfare vision: network technology challenges and guidelines," in *MILCOM*, October 2001, pp. 267–271.

[2] A. J. Ganesh et al., "Peer-to-peer membership management for gossip-based protocols," *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 139–149, 2003.

[3] H. C. Li, A. Clement, et al., "Bar gossip," in *7th OSDI*, November 2006, pp. 191–204.

[4] A. Demers, D. Greene, et al., "Epidemic algorithms for replicated database maintenance," in *PODC*, 1987, pp. 1–12.

[5] K. P. Birman, M. Hayden, et al., "Bimodal multicast," *ACM TOCS*, vol. 17, no. 2, pp. 41–88, 1999.

[6] W. R. Heinzelman et al., "Adaptive protocols for information dissemination in wireless sensor networks," in *MobiCom*, 1999, pp. 174–185.

[7] J. Luo et al., "Route driven gossip: Probabilistic reliable multicast in ad hoc networks," in *INFOCOM*, 2003, pp. 2229 – 2239.

[8] J. Sun, *Fixed-Priority End-To-End Scheduling in Distributed Real-Time Systems*, Ph.D. thesis, UIUC, 1997.

[9] T. Abdelzaher et al., "A feasible region for meeting aperiodic end-to-end deadlines in resource pipelines," in *ICDCS*, 2004, pp. 436–445.

[10] B. S. Manoj et al., "Real-time traffic support for ad hoc wireless networks," in *IEEE ICON*, 2002, pp. 335 – 340.

[11] K. Han et al., "Exploiting slack for scheduling dependent, distributable real-time threads in mobile ad hoc networks," in *RTNS*, March 2007.

[12] K. Han et al., "Byzantine-tolerant point-to-point information propagation in untrustworthy and unreliable networks," in *NBiS*, March 2007.

[13] E. D. Jensen et al., "A time-driven scheduling model for real-time systems," in *RTSS*, December 1985, pp. 112–122.

[14] K. Romer, "Time synchronization in ad hoc networks," in *MobiHoc*, 2001, pp. 173–182.

[15] S. Verma and W. Ooi, "Controlling gossip protocol infection pattern using adaptive fanout," in *ICDCS*, 2005.

[16] R. K. Clark, *Scheduling Dependent Real-Time Activities*, Ph.D. thesis, CMU, 1990, CMU-CS-90-155.

[17] L. J.S. Allen and A. M. Burgin, "Comparison of deterministic and stochastic sis and sir models in discrete time," *Mathematical Biosciences*, vol. 163, no. 1, pp. 1–33, January 1994.