Integrated Real-Time Scheduling and Communication with Probabilistic Timing Assurances in Unreliable Distributed Systems

Fei Huang[⋆] Kai Han[⋆] Binoy Ravindran[⋆] E. D. Jensen[‡]

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

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

Abstract

We consider distributed real-time systems that operate under run-time uncertainties including those on execution times and communication delays, and subject to arbitrary node failures and message losses. We present an integrated real-time scheduling and communication algorithm called Real-Time Scheduling with Reliable Data Delivery (RTSRD) that provides probabilistic end-to-end assurances on distributed task timeliness behaviors in such systems. RTSRD considers distributed tasks with end-to-end timing requirements that are expressed using time/utility functions and the optimality criterion of maximizing the total accrued utility. The algorithm decomposes end-to-end time constraints into local time constraints, and uses local slack time for node-local real-time scheduling and node-to-node real-time communication. We analytically establish RTSRD's properties including probabilistic satisfaction of task time constraints. We also compare RTSRD with a prior algorithm called RTG-L for the same problem. Our comparisons show that RTSRD outperforms RTG-L in terms of timeliness assurances (stronger) and algorithm overhead (lower).

1. Introduction

We consider scheduling distributed real-time tasks in large-scale systems based on unreliable networks, e.g., those without a fixed network infrastructure or subject to interference, including mobile and wireless ad-hoc networks. Distributed real-time systems are subject to the run-time uncertainties including arbitrary message losses and node failures, which would make acknowledgement-mechanism-based communication inefficient. Despite of these communication uncertainties, distributed real-time systems desire strong end-to-end assurances on distributed task timeliness behaviors. Probabilistic timing assurances are often appropriate.

Over the last years, ad-hoc networks are widely used in many real-time applications, such as battlefields and earthquake response systems. While these applications remain diverse, one common point they all share is the requirement on the end-to-end time constraint. However, in addition to the message losses and node failures, the potential contention in MAC protocols (e.g., IEEE 802.11 and 802.15.4), the node mobility nature of the ad-hoc networks, and the interference between the transmitting nodes, all make it difficult to achieve reliable distributed real-time scheduling.

In this paper, we present an integrated real-time scheduling and communication algorithm, called *Real-Time Scheduling with Reliable Data Transmission* (RTSRD), which provides probabilistic end-to-end assurances on distributed task time constraints in unreliable distributed systems based on ad-hoc network infrastructures. It first decomposes a distributed task's end-to-end time constraint into local time constraints on each execution node. Then it executes its local scheduling algorithm for local subtask scheduling. After that a process for timely finding the task's next execution node is issued in order to continue task execution.

At its core of communication scheme, RTSRD makes use of our newly designed "Reliable Data Delivery" (RDD) mechanism, for discovering a task's execution node in ad-hoc networks. While a task is executing on one node, RTSRD reserves enough bandwidth between the current and next execution nodes. After the task completes on the current node, RTSRD executes real-time data delivery within a required time period. In addition, to deal with possible message losses, or to deliver extremely large data chunks, RTSRD simultaneously transmits data in multiple paths. With RTSRD, the algorithm achieves reliable distributed real-time scheduling in ad-hoc network based unreliable systems.

Our work builds upon our prior work in [1] that presents the RTG-L algorithm. The major difference between RTG-L and RTSRD is that the latter adopts RDD to achieve point-to-point real-time communication, while RTG-L uses gossip protocol, which can achieve good communication reliability but incur high message overhead. End-to-end real-time scheduling has been studied in the past (e.g., [2]–[5]), but these are mostly limited to infrastructure-fixed networks. End-to-end timing assurances in unreliable networks are considered in [1], [6], [7], but they do not deal with point-to-point real-time communication between current and next execution nodes, which is precisely what our work does.

The rest of the paper is organized as follows: In Section 2, we discuss models and algorithm objectives. Section 3 illustrates our RTSRD algorithm including real-time scheduling and communication strategies. In Section 4, we build the analytical model for RTSRD and extend the model of RTG-L for theoretical comparison. In Section 5, we report our simulation studies. We conclude the paper in Section 6.

2. Models and algorithm objectives

2.1. Task model

Distributed tasks execute in local and remote objects by location-independent invocations. A task completes it execution by invoking a set of object operations that are specified when the task is initially created. The portion of a task executing an object operation is called a *subtask*. Thus, a task can be viewed as being composed of a concatenation of subtasks. A task's initial subtask is called its *root* and its most recent subtask is called its *head*. A task's head is the only subtask that is active. A subtask begins with a remote invocation, and ends with another possible invocation.

Upon creation, the number of a task's objects that need remote invocations are known, while the invocation sequence are assumed to be unknown. Execution time estimates of the subtasks of a task are known when the task arrives at the respective nodes. Normally, the application is comprised of a set of tasks, denoted as $\mathbf{T} = \{T_1, T_2, T_3, \ldots\}$.

We assume that each object operation required by the task is UNIQUELY hosted by one system node, and this assumption can be easily extended to a node group providing the same service.

2.2. Timeliness model and utility accrual scheduling

Each task's time constraint is specified using a time/utility function (or TUF) [8]. Figure 1 shows downward "step" TUFs.

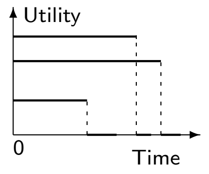


Figure 1. Step TUFs

A TUF decouples importance and urgency of a task—i.e., urgency is measured as a deadline on the X-axis, and importance is denoted by utility on the Y-axis. This decoupling is a key property of TUFs, as a task's urgency is typically orthogonal to its relative importance—e.g., the most urgent task can be the least important, and vice versa; the most urgent can be the most important, and vice versa.

A task T_i's TUF is denoted as $U_i(t)$. Classical deadline is unit-valued—i.e., $U_i(t) = \{0, 1\}$, since importance is not considered. Downward step TUFs generalize classical deadlines where $U_i(t) = \{0, \{n\}\}$. 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 $U_i(t) = 0, \forall t \notin [I_i, X_i], \forall i$.

If a task has not completed by its termination time, a failure-exception is raised, and exception handlers are released for aborting all partially executed subtasks (for releasing system resources). The handlers' time constraints are also specified using TUFs.

2.3. System model

The system consists of a set of nodes, denoted as $N = \{n_1, n_2, n_3, \ldots\}$. It adopts ad-hoc network structure, where each node acts as a router in communication processes. A basic unicast routing protocol called "Destination Sequenced Distance Vector" is assumed to be available for packet transmission between nodes. MAC-layer packet scheduling is assumed to be done IEEE 802.11. We assume that node clocks are synchronized using an algorithm such as [9].

Nodes may dynamically join or leave the network. Nodes may fail by crashing, links may fail transiently or permanently, and messages may be lost, all arbitrarily.

2.4. Objectives

Our goal is to design an algorithm that can schedule tasks with probabilistic termination-time satisfactions in the presence of message losses and node/link failures —i.e., establish probabilistically satisfied end-to-end timing assurance for a task. Further, we desire to maximize total task accrued utility, and minimize the number of aborted tasks.

3. The RTSRD algorithm

In this section, we will first describe the RTSRD algorithm starting from the scheduling algorithm, and then discuss the core communication scheme.

3.1. Building local TUF

RTSRD decomposes the task's end-to-end TUF based on the execution time estimates of the subtasks 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 subtasks (including the local subtask on n_j) be Er_i, the total remaining slack time be Sr_i, the number of remaining subtasks (including the local subtask on n_j) be Nr_i, and the execution time of the local subtask be $Er_{i,j}$. RTSRD computes a local slack time $LS_{i,j}$ for T_i as $LS_{i,j} = \frac{Sr_i}{Nr_i - 1}$, if $Nr_i > 1$; $LS_{i,j} = Sr_i$, if $0 \leqslant Nr_i \leqslant 1$. The algorithm equally divides the total remaining slack time to give the remaining subtasks a fair chance to complete their execution.

The local slack is used to compute a local termination time for the subtask. 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 RTSRD to test for schedule feasibility, while constructing local subtask schedules.

3.2. Constructing local schedule

RTSRD constructs local schedules of subtasks, with the goal of maximizing the total accrued utility, and minimizing the number of local termination times that are violated. The problem of maximizing total accrued utility itself is NP-hard [10]. Thus,

Algorithm 1: Local RTSRD Scheduling Algorithm [Local_SCHEDULE( )]

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

RTSRD considers heuristics in constructing local subtask schedules—Potential Utility Density (PUD). A subtask's PUD is the utility that can be accrued by executing the subtask, per unit of execution time. For a subtask S_i, at a scheduling event that occurs at time t, its PUD is given by $PUD_i(t) = U_i(t + LEr_i(t)) = LEri(t)$, where $LEr_i(t)$ is S_i's remaining (local) execution time at t. Thus, a subtask's PUD measures its return on "investment". RTSRD constructs local subtask schedules at two scheduling events: 1) the arrival of a message that signals the release of the subtask for execution on the node; and 2) completion of the subtask's execution.

RTSRD constructs local schedules as follows. The algorithm sorts all subtasks in the ready queue in the descending order of their PUDs. The sorted subtasks are then examined, highest PUD first, and inserted into a tentative schedule. The tentative schedule is sorted in the ascending order of the subtask termination times, to minimize termination time misses (since deadline ordering is optimal for that objective), and tested for feasibility. A schedule is said to be feasible, if the predicted completion time of each subtask in the schedule does not exceed its local termination time. (This feasibility testing is similar to that in [10].) If the schedule is infeasible, the subtask is removed from the schedule. The process is repeated until all subtasks in the ready queue are examined, while preserving the invariant of schedule feasibility. The subtask with the earliest termination time in the resulting schedule is then selected for execution. The algorithm is shown in Algorithm 1. It is worth mentioning that when a node is underloaded, this algorithm is capable of scheduling all the subtasks in that node to meet their time constraints [1].

3.3. Discovering the next head node

Underlying routing protocol. RTSRD makes use of routing protocols to discover and maintain

Table 1. N1's simplified forwarding table

Destination	Next Hop	Hops	Sequence Number
N1	N1	0	710
N2	N3	2	392
N3	N3	1	676
N4	N3	2	128
N5	N3	3	350

data delivery paths. Because deliveries are real-time, it expects path-finding time to be as short as possible.

Some well-known routing protocols, such as Ad-hoc On-demand Distance Vector (or AODV) and Dynamic Source Routing (or DSR), are reactive in that they form a path on-demand when a transmitting node requests one. Reactive protocols are not suitable when considered by RTSRD, because path-finding time might be very long, especially when several intermediate nodes exist in the path — i.e., long path-finding time might cause data delivery violate its time constraint.

RTSRD therefore adopts a proactive routing protocol called "Destination-Sequenced Distance Vector" (or DSDV). DSDV achieves proactiveness by letting nodes periodically discover and maintain paths. With DSDV, when a packet needs to be delivered, the path is already known and can be immediately used. Table 1 shows one DSDV node's (N1) simplified forwarding table.

Each node maintains a forwarding table for all reachable destinations. The table contains the next hop, number of hops and sequence number for each destination. A sequence number shows the freshness of a path, and is used to help nodes distinguish stale paths from the new ones and thus avoid formation of path loops.

Nodes broadcast routing updates periodically or at the time the network topology changes. Each path is tagged with a Time-to-Live (TTL) to indicate its freshness. Before a real-time data delivery starts, RTSRD will reduce the TTL of the path between the source and destination nodes, in order to enhance path maintenance frequency. In this way, it remains a reliable path when delivery begins.

RTSRD bandwidth reservation. Based on prediction on the possible size of real-time data, RTSRD executes bandwidth reservation (or BR) before the real data delivery (or DD) begins. In this way, when a real-time task completes and data is ready for delivery, it can immediately transmit data with desired sending rate. RRTD's basic strategy is shown in Figure 2.

RTSRD uses an existing wireless routing protocol (DSDV in this paper) to provide immediate data

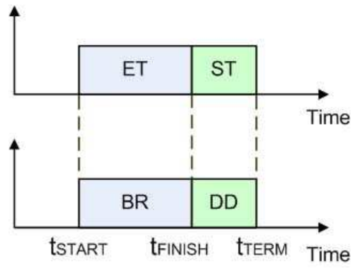


Figure 2. RTSRD strategy

delivery path. It also needs to reserve enough bandwidth before data is delivered, and achieves bandwidth reservation by controlling its and its neighbors' packet sending and receiving rates. In addition, it delivers data through multiple paths, in order to robustly transmit data, or separately transmit large data chunks.

IEEE 802.11 provides a CSMA/CA-based mechanism to allow nodes access wireless medium. To avoid the "hidden terminal" problem, before data transmission, the source node sends "Request to Send" (or RTS), and the destination node replies "Clear to Send" (or CTS). Every other node receiving RTS/CTS should remain in silence during the transmission period. With RTS/CTS, a node is not allowed to transmit whenever:

1) It is receiving data;
2) One of its neighbors is receiving data (due to the reception of a CTS);
3) One of its neighbors is transmitting data to a node that is neither another neighbor nor the node itself (due to the reception of a RTS).

The available bandwidth AB_i for a node i to transmit data is calculated as follows:

$$AB_i = EB_i - \left(b_i + \sum_{j \in \aleph_i} b_j + \sum_{j \in \aleph_i, k \neq \aleph_i^+} b_{jk} \right) \quad (1)$$

where b_i/b_j is the receiving bandwidth used by node i/j, b_{jk} is the traffic from node j to k, and $\aleph_i/\aleph_i^+$ is the set of neighbors of node i excluding/including itself. In real systems, poor link quality and the interference between nodes makes only a portion of the total bandwidth is usable. Therefore, here we use the total effective bandwidth EB_i for available bandwidth computation.

Denote the required bandwidth for real-time data delivery as r. In delivery processes, nodes not only need to reserve r bandwidth, but also need to consider the extra bandwidth which they use to remain in silence due to the reception of RTS/CTS. Table

Table 2. Data delivery in a 4-hop path

	A	B	C	D	E
Hop1	S	R	CTS	-	-
Hop2	RTS	S	R	CTS	-
Hop3	-	RTS	S	R	CTS
Hop4	-	-	RTS	S	R

Algorithm 2: Computing Required Bandwidth

```
1 if $i = source/destination$ then
2     if $destination/source$ in $neighbors$ then
3         $r \leq AB_i$;
4     else
5         $r \leq AB_i/2$;
6 else if $i \in \aleph_{source/destination}$ then
7     $r \leq AB_i/3$;
8 else
9     $r \leq AB_i/4$;
```

2 shows bandwidth consumption of nodes in data delivery path, where S and R are data sender and receiver, respectively. Note that S may be the source node (A) or intermediate nodes (B, C, D), and R may be the destination node (E) or intermediate nodes as well. We show how to compute reserved bandwidth for node i in Algorithm 2.

Multi-path Data Delivery. Message losses and node failures are frequent in some ad-hoc networks. To achieve reliable real-time data delivery, RTSRD adopts multi-path delivery mechanism (the number of paths is application-specific), as shown in Figure 3(a).

If the required bandwidth exceeds a node i's total effective bandwidth EB_i, multi-path delivery can help to distribute the delivery work load to different paths, as shown in Figure 3(b).

4. Theoretical analysis of RTSRD

In this section, we are going to theoretically analyze RTSRD from the perspective of timeliness and overhead.

4.1. Probabilistic timeliness assurance

Suppose that all the nodes in the distributed system are underloaded. The probabilistic end-to-end assurances on distributed task timeliness behaviors can be modeled by the following reliability analysis.

Given a packet loss ratio $plr_{i,j}$ and a bandwidth reservation failure ratio $bfr_{i,j}$, which refer to the failure probability of packet delivery and failure probability of bandwidth reservation, respectively, during the i^{th}-hop transmission in the j^{th} path, we can carry out the success probability r_j of packet transmission in the j^{th}-path as

$$r_j = \prod_{i,j} (1 - plr_{i,j} - bfr_{i,j}), \qquad (2)$$

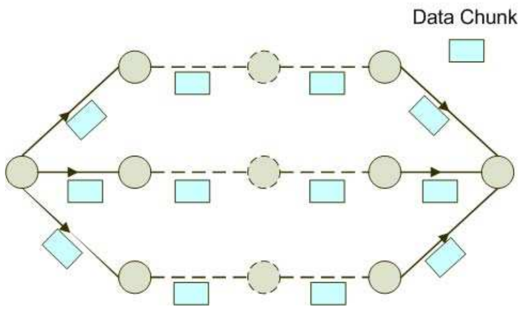
where n_h represents the number of hops in the path from current head node to next head node.

Considering multi-path transmission in RTSRD, we can express the overall packet reachability R_{pck} as

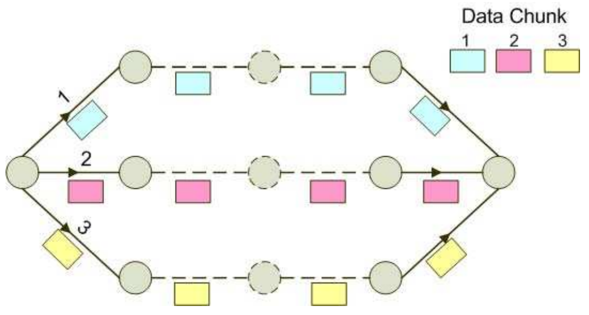
$$R_{pck} = 1 - \prod_{j} (1 - r_j). \qquad (3)$$

To reduce the complexity of analysis, in the following discussion we will assume that the number of hops in each path is equal to n_h with $n_h = max\{i\}$. To help understand, we can think of the existence of some virtual nodes with zero packet loss ratio, if one path has less nodes than n_h. Furthermore, to estimate the timeliness assurance under the bound of worst-case scenario, it is assumed that all the packet loss ratios are in the same value plr with $plr = max\{plr_{i,j}\}$, and all the bandwidth reservation failure ratios are in the same value bfr with $bfr = max\{bfr_{i,j}\}$. Accordingly, we can derive that

$$R_{pck} = 1 - (1 - r)^{n_r}, \qquad (4)$$



(a) Multi-path for reliable delivery



(b) Multi-path for large data delivery

Figure 3. Multi-path real-time data delivery

where n_r represents the number of paths, and $r = (1 - plr - bfr)^{n_h}$.

Assume that a data message consists of n_{pck} packets. Because there is no Automatic Repeat-reQuest (ARQ) mechanism inside RTSRD, the failure reception of any packet at the destination node, i.e., next head node, will lead to the failure of corresponding subtask. If the current and next head nodes are underloaded, a subtask k's probability P_k to satisfy its time constraint is given by

$$P_k = R_{pck}^{n_{pck}} = [1 - (1 - r)^{n_r}]^{n_{pck}}. \qquad (5)$$

Since a distributed task is completed on the successful execution of all the subtasks, the probability P_T for task T to successfully complete its execution through m distributed subtasks can be written as

$$P_T = \prod_{1 \leqslant k \leqslant m-1} P_k. \qquad (6)$$

In light of the above equation, it is ready to achieve the success probability P_{TS} for a whole task set TS as

$$P_{TS} = \prod_{T \in \ TS} P_T. \qquad (7)$$

If we assume all the subtasks in the task set have the same packet loss ratio, the same number of hops per path, and the same number of paths, we can further simplify P_T and P_{TS} as

$$P_T = P_k^{m-1}, P_{TS} = P_T^{n_{task}},$$

where n_{task} is the number of tasks in the task set. It is worth noticing that our simplification is based on the worst case scenario, which can help us analyze the bounded timeliness behavior in such distributed real-time system.

4.2. Message overhead properties

Based on the same worst case assumption in the previous analysis, we can derive the message overhead in terms of various ad-hoc networks parameters, such as packet loss ratio, number of hops and number of routes.

Theorem 1. *When bandwidth is reserved, the number of message overhead to execute one task, which includes the data packets generated and delivered in order to transmit the data message, is in linear relation to the number of routes n_r and the number of packets.*

Proof: For one subtask k, the message overhead N_k is given by

$$N_k = \frac{1 - (1 - plr)^{n_h}}{plr} \times n_r \times n_{pck}. \qquad (8)$$

In light of (8), we can further derive the message overhead N_T to execute one task is

$$N_T = \frac{1 - P_k^{m-1}}{1 - P_k} \times N_k. \qquad (9)$$

From (8) and (9), we can prove that N_T is in linear relation to n_r and n_{pck}. □

Theorem 2. *The message overhead to execute one task set is linear with the number of tasks n_{task} in the task set.*

Proof: From (9), we can calculate the message overhead N_{TS} to execute one task set by the following equation

$$N_{TS} = N_T \times n_{task}. \qquad (10)$$

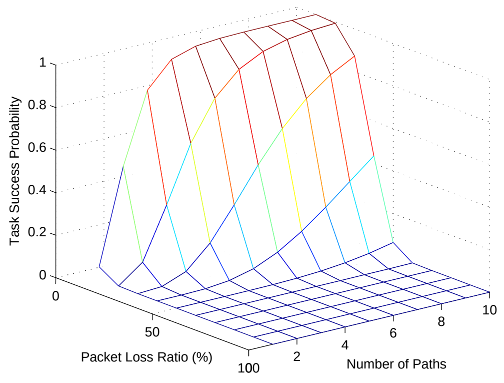
Thus, we can conclude the proof. □

Based on the analytical study about timeliness and message overhead, we can quantitatively illustrate the theoretical performance of RTSRD in Figure 4(a) and Figure 4(b). Suppose that the bandwidth reservations are all successful. For those two plots, we set the number of hops as 3, the number of packets for one message as 1, the number of subtasks as 10. As shown in those two plots, when the number of paths increases and packet loss ratio is low, it is easy to understand the task success probability will increase while the message overhead significantly steps up. However, when packet loss ratio is high, even more paths is added into the scheme, the packet still cannot survive through the first hops in paths, resulting from which the task success probability and message overhead doesn't change too much. Similarly, when the number of paths is small, the success probability and message overhead does not change obviously with the variety of packet loss ratio.

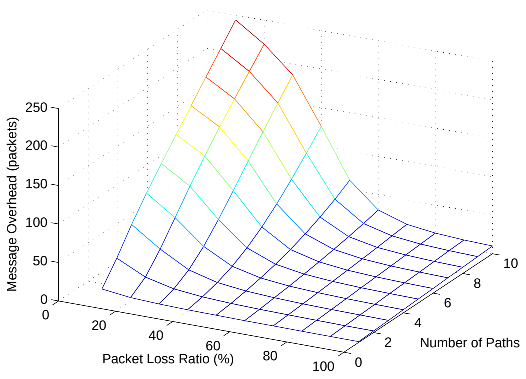
4.3. RTSRD vs. RTG-L

To understand the performance of RTSRD intensively, we now compare it with RTG-L algorithm which shows a satisfactory result in [5]. First, we will build the analytical model for the timeliness property and message overhead of RTG-L with practical parameters in applications, such as packet loss ratio, number of relaying hops, and number of gossip rounds.

Timeliness property of RTG-L. Let I_r be the number of newly informed nodes after gossip round



(a) Task success probability



(b) Message overhead

Figure 4. RTSRD

r, and U_r be the number of uninformed nodes at the end of gossip round r. On the initial condition before gossip starts, $I_0 = 1$, and $U_0 = N-1$ where N is the number of nodes in the ad-hoc network. Iteratively, we can have

$$U_r = U_{r-1} \times [1 - \frac{F_r \cdot (1-plr)^{n_{rh}}}{N-1}]^{I_{r-1}}; \tag{11}$$

$$I_{r-1} = U_{r-2} - U_{r-1}, \tag{12}$$

where plr is the packet loss ratio in each relaying hop (assuming it is equal for each hop), n_{rh} is the number of relaying hops in each fan-out path. Resulting from that, the success probability of subtask k can be carried out as

$$P_k = (1 - \frac{U_{fr}}{N-1})^{2n_{pck}}, \tag{13}$$

where fr represents the final round, and n_{pck} is the number of packets segmented from one message. It is worth mentioning that the exponent $2n_{pck}$ in (13) contains 2 as a factor because there are two gossip procedures involved, which are for locating the next head node, and for returning the result, respectively. Then, the success probability of one task is

$$P_T = P_k^{m-1}, \tag{14}$$

where m is the number of distributed subtasks belonging to the task. If we have n_{task} tasks in the task set, the success probability of the task set can be obtained by

$$P_{TS} = P_T^{n_{task}}. \tag{15}$$

Message overhead of RTG-L. To complete subtask k, the incurred message overhead is

$$N_k = [1 + (1 - \frac{U_{fr}}{N-1})^{n_{pck}}] \times \sum_{r=1}^{fr}[F_r \times I_{r-1} \times n_{pck} \times \sum_{i=0}^{n_{rh}-1}(1-plr)^i]. \tag{16}$$

Furthermore, we can work out the success probability to execute one task as

$$N_T = \begin{cases} N_k \times \sum_{i=0}^{m-2} P_k^i & m > 1; \\ 0 & m \leqslant 1. \end{cases} \tag{17}$$

So for one task set, the success probability can be mathematically expressed as

$$N_{TS} = N_T \times n_{task}. \tag{18}$$

In light of the above theoretical models, we can accurately compare the performance of our proposed RTSRD with previous RTG-L. For the convenience of illustration, we will discuss such theoretical results together with the simulation studies in Section 5.

4.4. Possible improvements for RTG-L

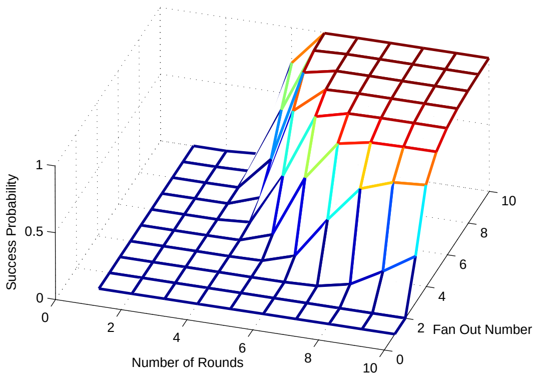
Based on the analytical model of RTG-L, we can explore the possible improvements for RTG-L. As in previous discussion, RTG-L normally incurs heavy message overhead, though its reliability is good. We are wishing to lower down the message overhead of RTG-L while keep its reliability performance. For that purpose, one additional metric is defined as P_T/N_T, which is the task success probability over the message overhead. Such metric can describe how much gain on reliability can be achieved from each packet cost in average.

In our study, we find two parameters strongly affect the message overhead of RTG-L. They are the number of rounds r and the fan out F_r, i.e., the number of message copies a node sends out when a gossip round starts. It is interesting to exploit the optimal adjustments of those parameters so as to maximize the performance of RTG-L in terms of P_T/N_T.

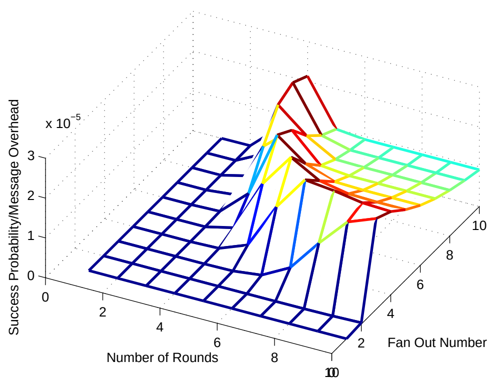
Let packet loss ratio $plr = 0.1$, the number of relaying hops $n_{rh} = 3$, the number of distributed subtasks $m = 3$, the number of packet segmented from one message $n_{pck} = 3$ and network size $N = 1000$. According to the above scenario, Figure 5(a) illustrates that the task success probability will generally increase with the augment of gossip rounds and fan out number. When the number of gossip rounds and the fan out are both low, e.g., $r = 4$ and $F_r = 4$, the task success probability is close to 0. When $r = 10$ and $F_r = 10$, the task success probability reaches 0.997, which is the zenith in the plot.

Now let us apply a new metric P_T/N_T. As shown in Figure 5(b), P_T/N_T has a wavelike profile, which means the top values of P_T/N_T are surrounding ($r = 10$, $F_r = 10$) at some radius, while not happen at the center part itself where maximum reliability is. Due the time constraint of real-time system, we prefer less gossip rounds if the performance of RTG-L can be satisfactory. In this example, we find the preferred value of P_T/N_T is at $r = 5$ and $F_r = 8$ where task success probability $P_T = 0.938$ which is just 0.059 off the best P_T. Remarkably, it saves over 48,218 packets in the theoretical model.

From the discussion in this section, the future theoretical optimization on RTG-L would be quite promising and can help the trade-off between message overhead and reliability.



(a) Task success probability



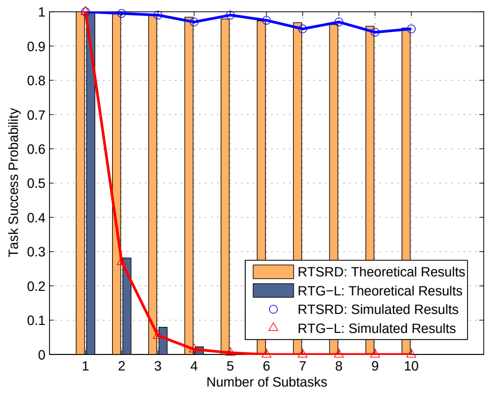
(b) Success probability/Message overhead

Figure 5. Possible improvements of RTG-L

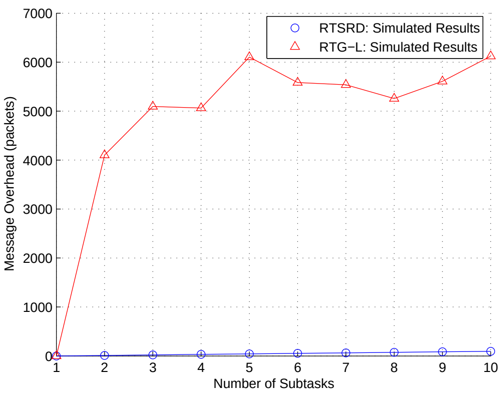
5. Simulation results of RTSRD

In this section, we present our simulation studies on RTSRD in terms of timeliness and message overhead under different task and network conditions, such as packet loss ratio, number of paths and number of hops. Some comparisons between theoretical results and simulated results are given to demonstrate the correctness of our analytical models for RTSRD and RTG-L. Furthermore, we also compare the performance of RTSRD with that of RTG-L.

Now we compare RTSRD and RTG-L in terms of reliability and message overhead. Because the bandwidth reservation failure ratio in the model of RTSRD plays an analog role with packet loss ratio, we can easily derive its influence on the system's behaviors from that of packet loss ratio. In terms of that, we suppose that the bandwidth reservations are all successful in the following discussions. The number of simulations is 200 in total. We consider such settings as the number of packets being 1 under the packet loss ratio of 0.1. Specifically, for RTSRD, we set up the number of hops as 3, the number of paths as 4. For RTG-L, we consider a network size of 1000 nodes with the fan out number as 4, the number of relaying hops as 3, and 6 gossip rounds. Figure 6(a) illustrates when more subtasks are required to be executed, the success probability will decrease for both RTSRD and RTG-L, which is because even if one subtask fails the whole task will claim a failure. However,RTSRD can sustain a significant success probability with just small reduction while RTG-L is very sensitive to the change in the number of subtasks. For example, when the number of subtasks becomes 3, RTSRD's success probability is still 0.989 while RTG-L's is unacceptably cut down to 0.054 in simulation. Beyond that, RTG-L's success probability is approaching zero while RTSRD keeps above 0.95. We should notice when there is only one subtask in the task which means current head node can do the work without invoking

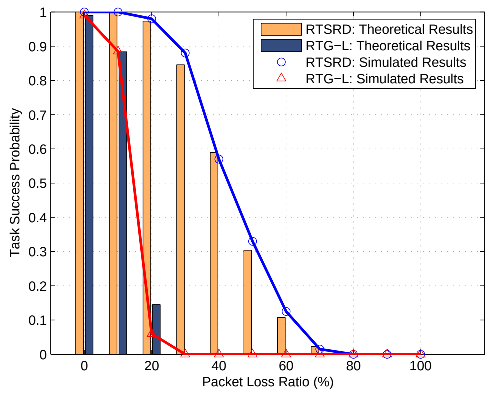


(a) Task success probability

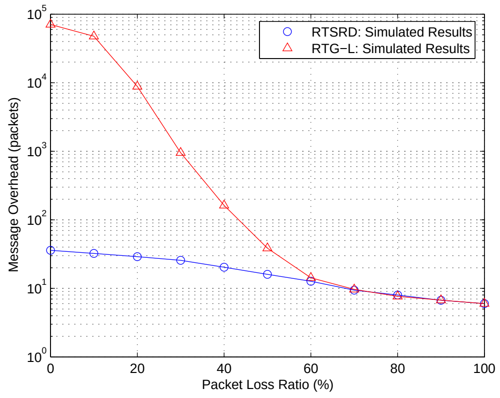


(b) Message overhead

Figure 6. RTSRD *vs.* RTG-L when the number of subtasks changes



(a) Task success probability



(b) Message overhead

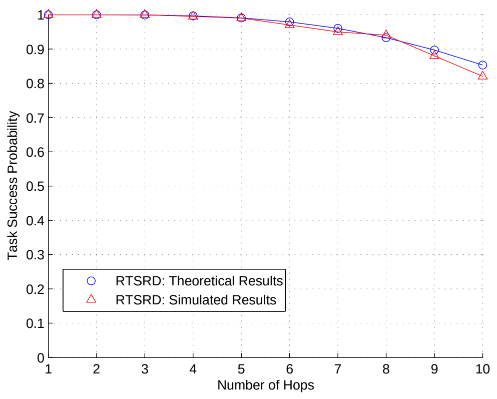
Figure 7. RTSRD *vs.* RTG-L when packet loss ratio changes

the next head node, the task success probability is anyway 1 in both algorithms. As in the illustration, our simulation accurately respects and proves the analytical results.

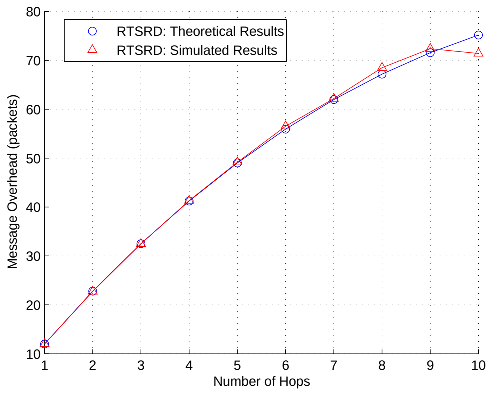
Under the same network condition, we now look at the message overhead in Figure 6(b). It is evident that the message overhead of RTG-L is drastically influenced by the number of subtasks. In contrast, RTSRD shows a small variation in terms of that. For example, when the number of subtasks increases from 4 to 10, over 2021 packets were required additionally in RTG-L, while for RTSRD, only 85 packets were added. Moreover, we should see when the number of subtasks is 1, the message overhead is zero because there is no communication required for next head node if local node can complete it. For a clear illustration, we do not demonstrate the corresponding theoretical results in this plot, which are close to the simulated results.

Another interesting comparison between RTSRD and RTG-L is in terms of the change of packet loss ratio. We repeat the simulation 200 times. For both algorithms, let the number of packets be 1 and the number of subtasks be 3. In detail, for RTSRD, we set up the number of hops as 3, the number of paths as 6. For RTG-L, we consider a network size of 1000 nodes with 6 fan outs, 3 relaying hops, and 6 gossip rounds.

As displayed in Figure 7(a), the simulation results closely follows and verifies the theoretical analyses for both RTSRD and RTG-L. In general, we can observe that when channel condition becomes worse, the task success probability will be undermined in both algorithms. Comparatively, RTSRD exhibits significantly better reliability with respect to the change in packet loss ratio. For instance, when $plr = 30\%$, the success probability of RTSRD is around 87.8% in our simulation. In contrast, RTG-L is approximately down to zero at the time. On the other hand, we may



(a) Task success probability



(b) Message overhead

Figure 8. RTSRD properties in terms of the number of hops

find, in Figure 7(b), the message overhead is decreasing when packet loss ratio increases. It is because that the higher transmission failure probability will discontinue more transmissions that should have been in consecutive nodes. As a result, less overhead will be issued. As we can see from this plot, RTG-L is orders of magnitude higher than RTSRD when pack loss ratio is lower than 40%. When packet loss ratio is high, they both converge to 6 because message will only be issued for the first 6 hops.

As shown in Figure 8, we consider how the change in the number of hops will influence the results of RTSRD. The number of paths in each path is fixed to 6; the number of subtasks is set to 3 and the number of packets is 1. The experimental results show that the task success probability will drop a little when the number of hops increases. In our simulation, the success probability is still in an acceptable status (above 0.8) when the number of hops is up to 10. Correspondingly, the message overhead will go up with the number of hops. The reason for those observations is any failure in the hops will call a failure such that more hops will lift up the failure probability. For the message overhead, more packet will be issued for the added hops. In addition, the theoretical results are verified one more by the simulation result.

6. Conclusions and future work

In this paper, we propose RTSRD, an integrated real-time scheduling and communication algorithm in unreliable distributed systems. To theoretically understand its timeliness behavior and message overhead, we build the analytical model for it. Through the theoretical and simulated comparisons with RTG-L, a gossip based real-time scheduling mechanism, RTSRD exhibits a remarkable reliability with lower message overhead under different distributed real-time system conditions. As an additional work, we also extend the model of RTG-L, and in light of that, we exploit some possible improvements for RTG-L.

Based on the results of this paper, investigations on scheduling under overloaded conditions will be studied to further improve the performance of RTSRD.

References

[1] K. Han *et al.*, “Exploiting slack for scheduling dependent, distributable real-time threads in mobile ad hoc networks,” in *RTNS*, Mar. 2007.
[2] I. Lee *et al.*, “A family of resource-Bound real-time process algebras,” in *EAPC*, Sep. 2006.
[3] OMG, “Real-time corba 2.0: Dynamic scheduling specification,” Tech. Rep., OMG, September 2001.
[4] P. Li et al., “A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints,” *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 454-469, Apr. 2006.
[5] T. Abdelzaher *et al.*, “A feasible region for meeting aperiodic end-to-end deadlines in resource pipelines,” in *ICDCS*, 2004, pp. 436–445.
[6] B. S. Manoj *et al.*, “Real-time traffic support for ad hoc wireless networks,” in *IEEE ICON*, 2002, pp. 335 – 340.
[7] N. Wang and C. Gill, “Improving real-time system configuration via a QoS-aware corba component model,” in *HICSS*, 2004, p. 10.
[8] S. Feizabadi *et al.*, “Automatic memory management in utility accrual scheduling environments ,” in *ISORC*, 2006, pp. 11–19.
[9] K. Romer, “Time synchronization in ad hoc networks,” in *MobiHoc*, 2001, pp. 173–182.
[10] R. K. Clark, “Scheduling dependent real-time activities,” Ph.D. dissertation, CMU, 1990, cMU-CS-90-155.