Virtual Time Synchronization over Unreliable Network Transport

Kalyan Perumalla Richard Fujimoto
kalyan@cc.gatech.edu fujimoto@cc.gatech.edu
College of Computing, Georgia Tech
Atlanta, GA 30332-0280

Abstract

In parallel and distributed simulations, it is sometimes desirable that the application's time-stamped events and/or the simulator's time-management control messages be exchanged over a combination of reliable and unreliable network channels. A challenge in developing infrastructure for such simulations is to correctly compute simulation time advances despite the loss of some simulation events and/or control messages. Presented here are algorithms for synchronization in distributed simulations performed directly over best-effort network transport. The algorithms are presented in a sequence of progressive refinement, starting with all reliable transport and finishing with combinations of reliable and unreliable transports for both time-stamped events and time management messages. Performance results from a preliminary implementation of these algorithms are also presented. To our knowledge, this is the first work to solve asynchronous time synchronization performed directly over unreliable network transport.

1. Introduction

Traditional parallel discrete event simulation research has so far focused mainly on reliable communication platforms. However, in certain application domains, such as Distributed Interactive Simulation (DIS) and High Level Architecture (HLA), it is desirable to execute the simulations directly over unreliable (best-effort) network transport such as User Datagram Protocol (UDP). This is motivated in part by potential performance gains due to the lower overhead afforded by unreliable transport compared to reliable delivery. However, current state-of-the-art parallel/distributed simulation techniques restrict the applications either to using completely reliable communication for all time-stamped ordered event processing, or alternatively to receive-ordered processing of all events irrespective of their timestamps. This is clearly restrictive and points to a need for extending parallel/distributed simulation technology to accommodate unreliable transport in time synchronization and timestamp-ordered event exchange.

Several important issues arise in the context of building simulation infrastructure over unreliable transport: Does time management make sense if time-stamped events sent over unreliable transport can be lost? How should time management be performed in such applications? Are traditional synchronization algorithms that are based on reliable transport less or more efficient than alternative algorithms (such as those presented here) implemented directly over unreliable network transport? Here, we attempt to answer some of these questions by first presenting a parallel/distributed simulation application model that accommodates a combination of reliable and unreliable time-synchronized events, followed by a description of novel algorithms that solve the associated time synchronization problem.

1.1. Motivation

In domains such as DIS and HLA, for performance reasons, unreliable message transport services such as UDP are typically employed for exchanging events. In DIS, entity state update events are sent periodically, while intermediate notification events are also sent when the state differs significantly from the dead-reckoned state. Since regular state updates are sent periodically, the applications are designed to tolerate some losses in the intermediate state notifications between the periodic state updates. However, unlike traditional parallel and distributed discrete event simulation (PDES) applications, time synchronization is not performed, partly because of lack of efficient algorithms in the context of unreliable network transport, thus giving rise to potential for anomalies in the simulation. Traditional time synchronization algorithms are not directly useful here, since most of them assume reliable delivery. The algorithms presented here are designed to solve this problem, so that time management can be enabled in such applications.

1.2. Related Work

Little literature exists on the use of unreliable network transport for simulation time management. Several global virtual time (GVT) algorithms have been formulated, but almost all of them assume reliable message delivery. In fact, most parallel simulation synchronization algorithms have been presented in the context of reliable delivery.

In [2], fault tolerance at the level of node-failures is addressed in the context of optimistic parallel simulation, whereas we address individual message losses, and are not restricted to optimistic simulators. Specialized hardware-supported techniques for fast reductions are presented in [10], whereas we address unreliability of message delivery in the common communication platforms, such as multi-hop wide-area networks. The work that is closest in relation to our work is the time synchronization algorithms presented in [8] in the context of unreliable delivery in broadcast-based networks. Also, our algorithms have some superficial resemblance to coloring-based GVT algorithms such as Mattern's algorithm[6], although they differ significantly in that unreliable communication is supported in our algorithm.

The solution to the noncommittal barrier synchronization problem presented in [7] in the context of reliable network transport appears to be closely related to the virtual time synchronization problem. We believe that variations of the algorithms presented here can be used to solve the same noncommittal barrier synchronization problem, but in the presence of message losses.

On a more theoretical note, distributed consensus problems such as leader election and termination detection have been previously studied in the context of faulty networks[1]. However, most of that work is theoretical in nature, dealing with less benign node and link failures, and not directly applicable to efficient distributed simulation execution over best-effort networks.

The rest of the paper is organized as follows. A generalized model is described for simulations that exchange time-stamped events over unreliable network transport. This is followed by a description of implementation challenges for providing safe simulation time advances during the course of simulation execution, along with associated definitions. We then present the algorithms and describe their operation, followed by a report on a preliminary performance study. We conclude with a summary of results and description of related open issues.

2. Background

2.1. Simulation Model

Here we consider a generalized model of distributed simulations in which the application designates certain events as "reliable" events, and others as "unreliable" events. For our purposes, a message is defined as reliable if it is guaranteed to arrive at its destination within a certain time limit. Both reliable and unreliable events are time-stamped. The difference between the two types is in their (1) potential to be lost (2) potential to violate global simulation time order. Reliable events are never lost, and always delivered to the application in a timely manner in relation to global simulation time. Unreliable events, on the other hand, can be lost, and can arrive sufficiently late to miss their timestamp ordered processing opportunity.

For correctness, the application requires *all* reliable events to be processed in global simulation time order. However, the application is designed to tolerate the loss (non-delivery) of a certain number of unreliable events per unit execution time and still retain simulation model accuracy. Unreliable events could potentially be received with their timestamps being less than the (currently committed) simulation time of the processor.

2.2. Simulator Implementation Challenges

The use of unreliable transport in parallel/distributed simulation raises two challenges that are different from traditional PDES: (1) lost time management messages (2) lost time-stamped events.

Time Management (TM) messages: Most parallel and distributed simulators have been implemented on top of reliable network delivery. Such implementations typically fail if the assumption of reliable delivery is violated at any time during the simulation execution. Most existing time synchronization algorithms have this property of failure, and hence cannot be used unmodified over unreliable network transport. Either existing algorithms need to be modified, or new algorithms must be devised to deal with losses in TM messages.

Time-stamped Events: A fundamental problem with unreliable time-stamped events is that it is hard to distinguish between transient events and lost events. The challenge is to resolve this conflict by accounting for as many events as possible within a specified amount of time, and presume the rest of the events are lost. If some of those events indeed arrive late without getting lost, then they could still be used in the application without violating global simulation time order if their timestamps happen to be greater than current simulation time at the received processor. On the other hand, if the timestamps are less than current simulation time, then those events can be passed to the application to be dealt with accordingly. Since applications that use unreliable events typically possess functionality to deal with late events, the late delivery should not be a problem.

In summary, the main trade-off in dealing with unreliable events is to wait sufficiently long for unreliable events to arrive, but not too long to hold up the simulation time advances in case the events never arrive.

2.3. *Lower Bound on Timestamp (LBTS)*

A value called lower bound on timestamp (LBTS) is a useful quantity that can be defined in any parallel/distributed simulation system. At any given moment during simulation execution, the LBTS value at a processor is defined as the timestamp of the earliest event that can be received by that processor in the future from other processors. The LBTS value is useful in conservative parallel simulation to determine which events are safe to execute. In optimistic parallel simulation it is useful in determining when it is safe to reclaim optimistic memory and to commit other irrevocable actions. The faster the LBTS is updated as the simulation progresses, the better is the performance of the simulation. Moreover, it is desirable that the process of computing LBTS value is asynchronous in nature, so that the simulation can continue without stopping while LBTS is being computed in background.

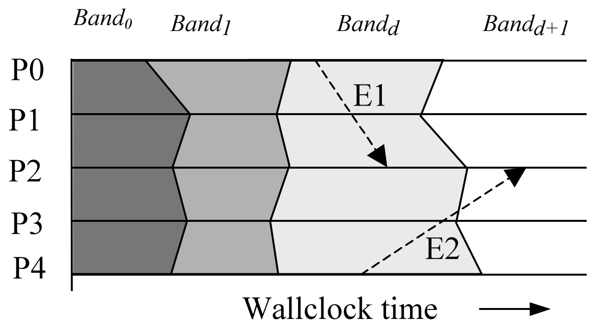

Figure 1: Illustration of wallclock time divided into bands. Event E1 is entirely contained in band d, while E2 crosses band *d* into *d+1*.

In our approach for asynchronous LBTS computation, the wallclock time at each processor is divided into contiguous bands as shown in Figure 1. The bands need not be equi-spaced, but could in fact have a staggered pattern as Figure 1 illustrates. Some events may be in transit across bands, while other events originate and terminate entirely within the same band.

In fact, the end of band $d+1$ is conveniently defined for our purposes by the latest wallclock time at which all events sent from band d are received by their destination processors. In other words, all events sent from band d are fully contained within bands d and $d+1$. All four algorithms presented here preserve this invariance.

2.4. *Definitions*

Every event E is tagged with the ID of the band d during which the event was sent. Thus each event is denoted by $E_d(t)$, or simply by E_d, where d is its sending band and t is its simulation receive time. Further, the transport type, if relevant, is shown as superscript. Thus, E^r denotes an event sent over reliable transport, and E^u denotes one sent over unreliable transport.

Let $\delta_i[d]$ denote the number of events E_d sent minus the number of events E_d received by processor i. Let $\Delta=\sum\delta_i[d]$. Let $\tau_i[d] = min(t)$ of all unprocessed events $E_{d'}(t)$ (uncommitted events, in the case of optimistic simulation) received by processor i, for all $d'<=d$.

Let $LBTS_d$ denote the smallest timestamp of all events $E_{d'}$ that originate in bands $d'<=d$ and received in future bands $d''>d$. In other words, it is the smallest timestamp of any event that is sent from any band $d'<=d$ and received in any band $d''>d$.

Note that $LBTS_d$ can be safely used as $LBTS$ in all bands $d'>d$. Also $LBTS_d<=LBTS_{d+1}$ for all d. In the algorithms presented later, the computation for $LBTS_d$ is performed during the band $d+1$.

Clearly, $LBTS_d = min(\ \tau_i[d]\ )$ over all i, if $\Delta=\sum\delta_i[d]$ equals zero. In other words, if every event originating or contained in band $d' <= d$ has been received at its destination processor, then no event received in future band $d'' > d$ can have timestamp less than $min\ \tau_i[d]$. But how do the processors know when Δ becomes equal to zero? In other words, how can the processors detect that all $E_{d'<=d}$ have reached their destinations?

2.5. *LBTS Computation*

One approach to detect exhaustion of all transient events belonging to band d is to iteratively perform a distributed reduction of all the $\delta_i[d]$ values. Once the sum (Δ) of all the values reduces to zero, the minimum of their corresponding $\tau_i[d]$ directly gives $LBTS_d$! This observation is key to the algorithms presented here. The algorithms are based on the fact that each LBTS computation can be performed as an iterated sequence of distributed reductions. The last reduction in each sequence is one that observes $\Delta=0$. The reductions in this sequence are numbered starting with zero, and every control message (not simulation event) used for reduction belonging to band d and reduction r is identified by its band number and iteration number, and denoted as V_{dr}. Each reduction itself is uniquely identified by its band and iteration numbers.

With every V_{dr}, the value of $LBTS_{d-1}$ is piggybacked, and hence reduction messages are written as $V_{dr}(L_{d-1})$ where L_{d-1} denotes the value of $LBTS_{d-1}$. Note that $LBTS_{d-1}$ is always available when $LBTS_d$ is being computed, for any d.

3. Algorithms

We now present four algorithms corresponding to four different combinations of reliability of time-stamped events and time management (TM) messages.

The first algorithm is designed for the classical PDES model: reliable events coupled with reliable TM messages (E_RV_R). The remaining three algorithms are based on the first algorithm and are progressively refined to accommodate unreliability. As a surprisingly simple variation of the first algorithm, we present the second algorithm to deal with lost TM messages, i.e., reliable events coupled with unreliable TM messages (E_RV_U). We further refine the second algorithm to give the third algorithm, which is designed for the more general case of applications using both reliable and unreliable events coupled with unreliable TM messages ($E_RE_UV_U$). Finally, as a special case of the third algorithm, we describe the fourth algorithm for an important class of applications that use both reliable and unreliable events coupled with reliable TM messages ($E_RE_UV_R$).

The algorithms are presented from the point of view of processor *i*'s execution. All processors execute the same algorithm. In all four algorithms, reduction messages are identified by their band and sequence identifiers *(d,r)*. When a reduction *(d,r)* is in progress at a processor, any arriving reduction messages belonging to an older reduction *(d',r')* are discarded (i.e., if $d'<d$ or if $d=d'$ and $r'<r$). If any reduction messages belonging to a future reduction *(d'',r'')* are received, they are buffered until the algorithm moves to that reduction (i.e., if $d''>d$ or if $d''=d$ and $r''>r$). Also, whenever a processor *i* receives a time-stamped event, it immediately adds that event to its local event queue.

As noted previously, all the algorithms are defined in such a way that computation of $LBTS_d$ is started as well as completed entirely within band $d+1$. A corollary is that all events originating in band *d* are received in bands *d* or $d+1$ and no later.

Note that in both reliable and unreliable transports, the algorithms do not require message order to be preserved by the network.

3.1. *Reliable Events and Reliable TM Messages*

The algorithm for this model is shown in the following box. This algorithm possesses some resemblance with other coloring-based global virtual time (GVT) algorithms, such as Mattern's algorithm [6]. The band numbers roughly correspond to the colors in those algorithms; however, the use of multiple reductions per band is unique to our algorithm.

Although other well-known algorithms exist in PDES literature for time synchronization over reliable transport, our Algorithm 1 is unique in that the algorithms for unreliable transport follow as natural extensions to this algorithm, as will be seen in ensuing sections.

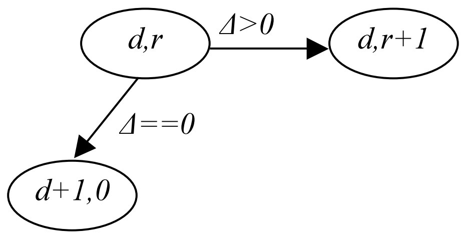
$$\begin{tikzcd} d,r \arrow[r, "\Delta>0"] \arrow[d, "\Delta==0"'] & d,r+1 \\ d+1,0 & \end{tikzcd}$$

Figure 2: Transitions from reduction *(d,r)* in Algorithm 1.

```
Algorithm 1: E_R V_R At each processor i:
1.  For all d, $\delta_i[d]=0$; $\tau_i[d]=\infty$.
2.  $d=0$
3.  $r=0$
4.  $\tau_i[d]=min(\tau_i[d], MinQ_i)$
5.  Start-reduction($d, r, \delta_i[d], \tau_i[d]$)
6.  While not end of reduction($d,r$)
    6.1  If $E_d(t)$ is received
         { $\tau_i[d]=min(\tau_i[d], t)$; $--\delta_i[d]$ }
    6.2  If any $E$ is sent
         {tag $E$ as $E_{d+1}$; $\delta_i[d+1]++$}
7.  $(\Delta,\tau)$=reduced-value($d,r$)
8.  If $\Delta>0$ then   { $r++$; goto 5 }
9.  Else ($\Delta==0$) { Output $LBTS_d=\tau$; $d++$; goto 3 }
```

In line 1, all the $\delta_i[d]$ values are initialized to zero since no events are sent or received during any band at the beginning of simulation. Similarly, all the $\tau_i[d]$ values are initialized to infinity. The algorithm starts with band 0, by initializing *d* to zero (line 2). For each band, it starts with the sequence of reductions, starting with reduction zero (line 3). The LBTS computation for a band *d* starts by initializing $\tau_i[d]$ to the smallest timestamp of events in a snapshot of its local event queue ($MinQ_i$) on line 4. This snapshot covers all events that may have arrived before the processor entered the $LBTS_d$ computation. The loop between lines 5 and 9 inclusive is used to iterate through the reduction sequence of band *d* until $LBTS_d$ is computed.

During each iteration of the loop, a distributed reduction R_{dr} is started (line 5), with $\delta_i[d]$ and $\tau_i[d]$ as processor *i*'s contribution to the reduction values. Note that $\delta_i[d]$ are reduced using the *sum* operator, while $\tau_i[d]$ are reduced using the *minimum* operator. The reduced value is stored in (Δ,τ), where, Δ represents the sum of all $\delta_i[d]$ (the total number of outstanding events in the network that are yet to reach their destinations), and τ is the minimum of all $\tau_i[d]$. If Δ does not equal zero, it implies that not all events generated in band *d* have been accounted for in τ (some events are in transit). In that case, another reduction is attempted by continuing the loop to move to the next reduction $r+1$ (line 8). If Δ equals zero, then it clear that there are no more outstanding events in transit in the network. Hence τ represents the minimum of all events generated within band d that are not yet processed by the processors, which is nothing but $LBTS_d$. Hence, $LBTS_d$ is generated as output and then the algorithm moves to the next band $d+1$ (line 9). Figure 2 illustrates the transitions from reduction (d,r) to the next reductions.

Even while a reduction is in progress, the simulation could send and/or receive other events, since the LBTS computation is asynchronous. These events are handled in lines 6.1 and 6.2. If an event originating in band d is received, then $\tau_i[d]$ is updated to take the timestamp of the received event into account, and $\delta_i[d]$ is decremented to note the fact that one more event of band d has been accounted for (line 6.2). If an event is being sent, that event is tagged as originating in the next band $d+1$, and the corresponding $\delta_i[d+1]$ is incremented to note the fact that one more event originated in band $d+1$.

It is easy to prove by induction on d that Algorithm 1 correctly computes $LBTS_d$.

3.2. Reliable Events and Unreliable TM Messages

Algorithm 1 requires surprisingly few modifications to deal with lost reduction messages. As such, the second algorithm is a natural extension to algorithm 1 to function in the presence of unreliable reduction messages. The extension is to essentially perform timeouts on incoming reduction messages, and act on timeouts.

Let us examine the effect of a lost reduction message in algorithm 1. First it should be noted that some processors might still be able to complete their current reduction and move on to the next reduction or next band. This is possible, for example, if the message is lost in the last level in a butterfly communication pattern for hierarchical reduction [3]. Other processors fail to complete their current reduction, waiting directly for the lost message, or indirectly for messages that are supposed to be generated based on the lost message.

Thus, three cases arise in Algorithm 1 if a reduction message is lost:

Case 1: All processors fail to complete their current reduction (d,r) waiting for the message that will never arrive.

Case 2: Some processors successfully complete their reduction while others fail. Those processors that do succeed observe that the Δ value has still not reached zero, and hence they move on to the next reduction within the same band, i.e., to $(d,r+1)$. The other failed processors are still waiting for their current reduction (d,r) to complete.

Case 3: Those processors that succeed observe that Δ equals zero, and hence successfully complete the computation of $LBTS_d$ and move on to the next band $d+1$, starting with reduction $(d+1,0)$.

The first two cases can be easily addressed by adding a timeout mechanism to reductions. Upon waiting for a predefined time interval, reductions complete abnormally with a Δ value of ∞. The processors then will continue with the algorithm as though the failed reduction in fact completed with a non-zero value for Δ, making it appear as though some more messages of band d are in transit.

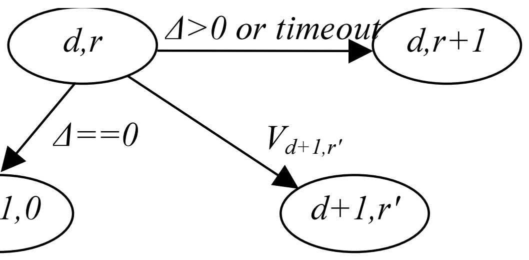

Figure 3: Transitions from reduction *(d,r)* in Algorithm 2.

Now consider case 3. In this case, some processors are still waiting for their current reduction to complete, but might receive reduction messages corresponding to the next band $(d+1,r')$ from the successful processors. Recall that the value of $LBTS_d$ is always piggybacked as L_d in reduction messages, $V_{d+1,r'}$, of band $d+1$. The waiting processor can exploit this fact when it receives a reduction message of $d+1$, by using that L_d value as $LBTS_d$ to immediately terminate its current reduction. Moreover, for the next band $d+1$, it can advance to reduction r' instead of starting with reduction 0. These transitions are illustrated in Figure 3, and the Algorithm 2 is given in the following box, expressed as a modification to Algorithm 1. The modification is to add timeout mechanism to reductions, and to terminate the currently active reduction (d,r) if a future reduction message $V_{d+1,r'}$ is received, and catch up to that future reduction.

```
Algorithm 2: $E_R V_U$ At each processor $i$:
Same as Algorithm 1, but with the following added:
    6.3  If $V_{(d+1)r'}(L_d)$ is received
         { Output $LBTS_d=L_d$; $d$++; $r=r'$; goto 4 }
```

It is very interesting that tolerance to lost reduction messages can be easily achieved by adding just a couple of lines to the reliable delivery-based algorithm. Thus it can be noted that resilience to network transport unreliability is conceptually very easy to achieve in simulation time management.

3.3. Reliable and Unreliable Events and Unreliable TM Messages

We now turn to the more general case in which applications can send time-stamped events on both reliable and unreliable transports, and also want to perform time management over unreliable transport. All events sent over reliable transport must always be factored into time management; however, there is flexibility with regard to the number of unreliable events that can be missed in time management, which in turn translates into a trade-off for performance optimization.

We exploit this flexibility by introducing two parameters, α and β, using which this algorithm can be tuned to suit the application's performance needs. The parameter α is defined as a limit on the number of reductions performed per band. The parameter β is defined as a limit on the number of unreliable events that the application can tolerate per band, if all those β events (eventually) violate global timestamp order or never arrive. A special case is when $\beta=\infty$, in which case $LBTS_d$ can be advanced without ever waiting for unreliable events.

The parameter α can be viewed as controlling the maximum amount of wallclock time spent waiting for unreliable events, while β can be viewed as controlling the maximum number of unreliable events that can be ignored in the LBTS computation.

The algorithm is shown in the following box. This algorithm follows along the lines of Algorithm 2, except that the conditions for transitions from one reduction to the next are slightly more complex.

```
Algorithm 3: $E_R E_U V_U$ At each processor $i$:
1.  For all $d$, $\delta^r_i[d]= \delta^u_i[d]=0; \tau_i[d]=\infty$.
2.  $d=0$
3.  $r=0$
4.  $\tau_i[d]=min(\tau_i[d], MinQ_i)$
5.  Start-reduction($d, r, \delta^r_i[d], \delta^u_i[d], \tau_i[d]$)
6.  While not end of reduction($d,r$)
    6.1  If $E_d(t)$ is received
         6.1.1  $\tau_i[d]=min(\tau_i[d], t);$
         6.1.2  If $E_d$ is reliable { --$\delta^r_i[d]$ }
         6.1.3  Else (unreliable) { --$\delta^u_i[d]$ }
    6.2  If any $E$ is sent
         6.2.1  Tag $E$ as $E_{d+1}$
         6.2.2  If $E$ is reliable { $\delta^r_i[d+1]$++ }
         6.2.3  Else (unreliable) { $\delta^u_i[d+1]$++ }
    6.3  If $V_{(d+1)r'}(L_d)$ is received
         { Output $LBTS_d=L_d$; $d$++; $r=r'$; goto 4 }
7.  $(\Delta^r, \Delta^u, \tau)$=reduced-value($d,r$)
8.  If $\Delta^r>0$ or ($\Delta^u>\beta$ and $r<\alpha$) then { $r$++; goto 5 }
9.  Else { Output $LBTS_d=\tau$; $d$++; goto 3 }
```

First, each $\delta_i[d]$ is split into two terms: $\delta^r_i[d]$ and $\delta^u_i[d]$, where $\delta^r_i[d]$ corresponds to reliable events and $\delta^u_i[d]$ corresponds to unreliable events. Similarly, Δ is split into Δ^r and Δ^u. For correctness of simulation, all processors must necessarily keep iterating for $LBTS_d$ until the total number of transient reliable messages in the system, given by Δ^r, becomes zero for $LBTS_d$ to be correct. Otherwise, $LBTS_d$ could potentially advance further than the timestamp of a transient reliable event that can arrive later. In contrast, by definition, the application can tolerate up to β unreliable events that violate global timestamp order.

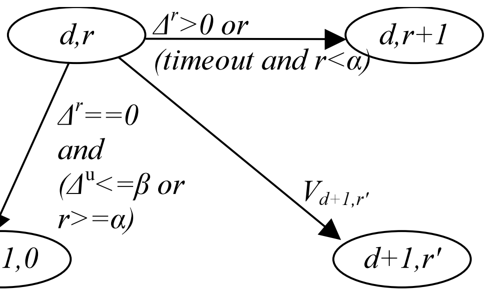

Figure 4: Transitions from reduction *(d,r)* in Algorithm 3.

Except for the way transport types are used for events, this algorithm is similar to Algorithm 2, and differs with it in the following ways: (1) Reductions are performed on triples $(\delta^r_i[d], \delta^u_i[d], \tau_i[d])$, instead of pairs $(\delta_i[d], \tau_i[d])$. (2) Whenever an event is sent or received, the appropriate event counter is updated corresponding to the event's transport type. (3) The termination condition for reduction sequence and LBTS computation for a band are modified appropriately to accommodate unreliable events.

3.4. Reliable and Unreliable Events and Reliable TM Messages

In an important class of applications, such as Distributed Interactive Simulation (DIS), applications utilize a mixture of reliable and unreliable events. Periodic state updates, which contain critical information, are sent over reliable transport, while less critical events are sent over unreliable transport. Time management in such applications can be performed over reliable transport. The last algorithm addresses time management in such applications.

This algorithm, in fact, can be expressed as a special case of Algorithm 3. First, the timeout value for reductions can be set to infinity, since reduction messages sent over reliable transport are never lost. Secondly, α can be set to a value that ensures that the algorithm waits for a fixed amount of time before giving up waiting for any transient unreliable events. Note that α should be not be set to too low a value, since otherwise it could potentially ignore all transient unreliable events. Finally, β can be set to infinity, which essentially translates to the fact that no unreliable events will ever hold up the progress of LBTS.

4. Performance Study

We have completed a preliminary implementation of the algorithms and incorporated them into the time management module of the *Federated Simulations*

Development Kit (FDK) from Georgia Tech[5]. The FDK is a modular set of libraries designed for the development of Run Time Infrastructures (RTIs) for parallel and distributed simulation systems, and includes an RTI that implements a subset of the High Level Architecture (HLA) services.

To study the effects of unreliable transport on the performance of time management, we tested the implementation using two applications. The first is a time-stepped application that exercises simulation time advances in the absence of inter-processor event exchange (i.e., invokes HLA-like *TimeAdvanceRequest* service). The application stresses the speed of asynchronous reduction, with the metric of interest being the number of LBTS computations that successfully complete per second of wallclock time. The second is a TCPI/IP traffic simulation using the Parallel and Distributed NS (PDNS), which uses the FDK for event exchange and synchronization. The PDNS simulation included both time-stamped event exchange and simulation time synchronization. The experiments were run on a network of workstations. In one scenario, the workstations were connected by local area network (Ethernet) at Georgia Tech, and in the other, the workstations spanned Georgia Tech, Dartmouth College and Carnegie Mellon University.

Unfortunately, we observed few message losses in either scenario. The performance of time management (number of LBTS computations per second) remained the same between UDP-based and TCP-based communication. This can be attributable to the fact that TCP can perform as efficiently as UDP in the absence of losses, and the simulation incurs the overhead of TCP connection setup among processors only at initialization time. The absence of losses prevented us from making conclusions about the performance of reliable and unreliable transports, except for the observation that our algorithms performed no worse than reliable transport-based algorithms in the absence of message losses.

As an alternative scenario, we used reliable transport (TCP) and artificially dropped messages (using a uniform random number generator) in the RTI communication module before they are submitted to the time management module. This provided us control on the actual loss probability realized in the network during execution as seen by the time management module.

The experiments were executed on a cluster of 16 Intel Pentium III 550 MHz processors connected by fast Ethernet. The machines were normally loaded when the experiments were executed (i.e., there were other user processes running on the system). The results are shown in Figure 5, comparing the rate of completed LBTS computations for different values of message loss probability *q* (10%, 1% and 0.1%), against that of reliable delivery (no loss).

As expected, increasing the loss probability decreases the LBTS computation rate. The performance for low loss probability (q=0.1%) is similar to that of no losses, showing that the algorithm is capable of dynamically extracting the superior performance of reliable delivery if the actual observed losses are low. For higher loss probability (q=10%), the reduction algorithm is observed to be not sufficiently robust for larger number of processors. For more than 4 processors, reductions timed out more frequently due to the higher loss probability; degrading the overall LBTS rate. The average number of reduction iterations per band was between 1 and 2.

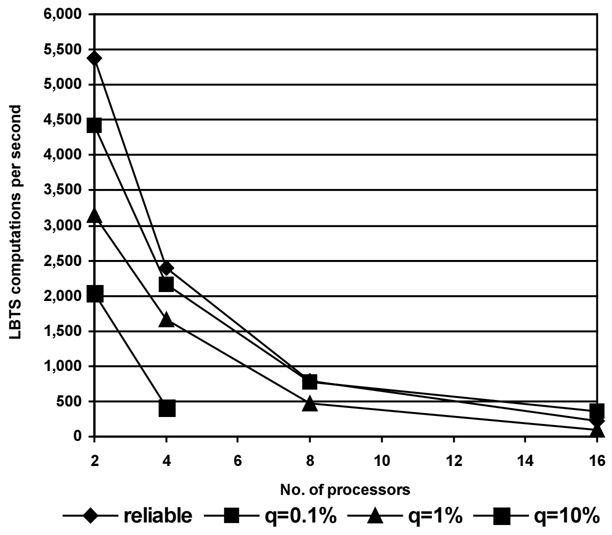

Figure 5: Performance of Algorithm 2 for different values of message loss probability *q*.

Evidently, more extensive performance analysis is needed before conclusive notes on the performance differential between reliable and unreliable transport-based synchronization can be made.

5. Conclusions and Future Work

Time-synchronized distributed simulation over best-effort/unreliable networks is an important problem that has largely gone unresolved so far. Little literature exists that deals with solutions for simulation time management in the presence of event and control message losses. Here, we have presented algorithms to address these issues. To our knowledge, ours is among the first works to address this problem in the context of general best-effort networks. We have first defined a simulation model in the context of unreliable delivery of time-stamped events, which has not traditionally been considered in PDES. We have shown that unreliable delivery of time management messages can be dealt with in a relatively straightforward fashion, as a simple extension of our algorithm for reliable transport. In addition, none of our algorithms assumes that the network preserves message order. Using the algorithms presented here, more complex time-managed applications can be developed using a mixture of reliable and unreliable time-stamped events, and using reliable or unreliable time management messages. It is now clear that unreliable events can in fact be explored for use in real-life applications. Additional work, however, remains in the area of performance analysis, optimization and tuning, as discussed next.

5.1. Reduction Timeout Value Estimation

It is clear that the timeout value used for detecting failed reductions affects the rate of LBTS computations. Longer timeouts imply longer time for processors to discover failed reductions, thus wasting time. On the other hand, lower timeout values make the processors timeout too early, thus artificially missing messages that might complete the reduction. The best timeout value is one that is based on dynamically tracking network delays, and varying it accordingly. It is very hard to predict message delays in multi-hop networks; however, a reasonable alternative would be to start the timeout value at a large conservative value, and gradually adjust it based on a history of actual delays observed for the received messages. Dynamic adjustment techniques for an optimal timeout value remain to be investigated. Some of the techniques from networking research, such as TCP timeout mechanisms, could be potentially applied here.

5.2. Threshold for Unreliable Events

In applications using both unreliable and reliable time-stamped events, the threshold β determines the time spent waiting for unreliable events in transit to arrive at their destinations. Thus, the β value affects the rate of LBTS computations. Larger β values waste time waiting for the events that will never arrive. Smaller β values advance the LBTS more rapidly than the unreliable events can arrive, potentially advancing LBTS beyond the timestamps of some or all of the outstanding unreliable events. Thus, there is a tradeoff between waiting sufficiently long to receive as many unreliable events as possible, and waiting sufficiently little to not hold up the LBTS computation for receiving the unreliable events that may have actually been lost. Additional research is needed to dynamically estimate and adjust the threshold β to its optimal value.

5.3. Robust and Scalable Reduction

Another interesting research item is the design of robust and scalable distributed reduction algorithms that perform well even in the presence of significant number of lost messages. The challenge is to devise a distributed reduction algorithm that scales with the number of processors as well as with message loss probability. When used within the LBTS algorithms presented here, it can improve the efficiency of time management by providing a high degree of probability that a reduction will complete without timing out, despite message losses.

5.4. Additional Performance Evaluation

The biggest challenge to a performance evaluation of the algorithms was that we could not control the amount of message losses observed on the network connections. To address this, we are exploring network emulation-based experimentation approaches. We also intend to study the performance of the algorithms on a more extensive set of applications (e.g. *ModSAF*[11]) over wide-area networks using UDP.

6. Acknowledgements

The authors would like to thank David Nicol and Maria Hybinette for providing access to their workstations, at Dartmouth and CMU respectively, for wide area experiments. Thanks also to Mostafa Ammar and George Riley for helpful comments on TCP vs. UDP performance.

7. References

[1] Afek, Y. and M. Saks, "Detecting Global Termination Conditions in the Face of Uncertainty," Principles of Distributed Computing, August 1987.

[2] Damani, O.P., V.K. Garg, "Fault Tolerant Distributed Simulation," the 12th Workshop on Parallel and Distributed Simulation, May 1998.

[3] Fujimoto, R.M., "Parallel and Distributed Simulation Systems," Wiley Inter-science, 2000.

[4] Fujimoto, R.M., "Time Management in the High Level Architecture," Simulation, Vol. 71, No. 6, December 1998.

[5] Fujimoto, R.M., T. McLean, K. Perumalla and I. Tacic, "Design of High-performance RTI Software", Proceedings of Distributed Simulations and Real-time Applications, August 2000.

[6] Mattern, F, "Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation", Journal of Parallel and Distributed Computing, 1993.

[7] Nicol, D., "Noncommittal Barrier Synchronization," Parallel Computing, vol. 21, 1995.

[8] Riley, G.F., et al, "Network Aware Time Management and Event Distribution," the 14th Workshop on Parallel and Distributed Simulation, May 2000.

[9] Riley, G.F., et al, "A Generic Framework for Parallelization of Network Simulations",MASCOTS, 1999.

[10] Srinivasan, S., et al, "Implementation of Reductions in Support of PDES on a Network of Workstations," the 12th Workshop on Parallel and Distributed Simulation, May 1998.

[11] Modular Semi-Automated Forces, http://www.modsaf.org.