9-14-2017

A Stochastic Model of Plausibility in Live-Virtual-Constructive Environments

Jeremy R. Millar

A STOCHASTIC MODEL OF PLAUSIBILITY IN LIVE-VIRTUAL-CONSTRUCTIVE ENVIRONMENTS

DISSERTATION

Jeremy R. Millar, Major, USAF

AFIT-ENG-DS-17-S-015

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-DS-17-S-015

A STOCHASTIC MODEL OF PLAUSIBILITY IN LIVE-VIRTUAL-CONSTRUCTIVE ENVIRONMENTS

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Jeremy R. Millar, B.S.C.S., M.S.C.S

Major, USAF

September 2017

AFIT-ENG-DS-17-S-015

A STOCHASTIC MODEL OF PLAUSIBILITY IN LIVE-VIRTUAL-CONSTRUCTIVE ENVIRONMENTS

DISSERTATION

Jeremy R. Millar, B.S.C.S., M.S.C.S
Major, USAF

Committee Membership:

Douglas D. Hodson, PhD
Chairman

Gilbert L. Peterson, PhD
Member

Darryl K. Ahner, PhD
Member

ADEDEJI B. BADIRU, PhD
Dean, Graduate School of Engineering and Management

AFIT-ENG-DS-17-S-015

Abstract

Distributed live-virtual-constructive simulation promises a number of benefits for the test and evaluation community, including reduced costs, access to simulations of limited availability assets, the ability to conduct large-scale multi-service test events, and recapitalization of existing simulation investments. However, geographically distributed systems are subject to fundamental state consistency limitations that make assessing the data quality of live-virtual-constructive experiments difficult. This research presents a data quality model based on the notion of plausible interaction outcomes. This model explicitly accounts for the lack of absolute state consistency in distributed real-time systems and offers system designers a means of estimating data quality and fitness for purpose. Experiments with World of Warcraft player trace data validate the plausibility model and exceedance probability estimates. Additional experiments with synthetic data illustrate the model's use in ensuring fitness for purpose of live-virtual-constructive simulations and estimating the quality of data obtained from live-virtual-constructive experiments.

Table of Contents

Page

Abstract . . . iv

List of Figures . . . vi

List of Tables . . . viii

List of Abbreviations . . . ix

I. Introduction . . . 1

II. Background . . . 6

2.1 Consistency Models . . . 6
Strong Consistency . . . 7
Eventual Consistency . . . 7
Consistent Prefix . . . 8
Bounded Staleness . . . 8
Monotonic Reads . . . 8
Read My Writes . . . 8
2.2 CAP Theorem . . . 11
2.3 State Synchronization . . . 12
Types of Error in Distributed Virtual Environments . . . 12
State Update Policies . . . 16
2.4 Tolerance Intervals . . . 20
2.5 Summary . . . 20

III. Modelling Plausibility in LVC Simulation . . . 23

3.1 LVC Architecture . . . 23
3.2 Modeling Plausibility . . . 25
3.3 Computing Plausibility Exceedance Probabilities . . . 28

IV. Analysis and Applications . . . 35

4.1 Model Validation . . . 35
4.2 Performance Evaluation . . . 43

V. Conclusion . . . 48

Bibliography . . . 51

Appendix A. Publications . . . 59

List of Figures

Figure		Page
1	A simple two-entity distributed virtual environment.	9
2	State changes for a simple two-entity distributed virtual environment.	9
3	Aging of distributed state data [20].	14
4	Entities, implementation, and dynamic shared state [23].	26
5	A simple two-state model of interaction plausibility.	28
6	Factors influencing interaction plausibility and the dependencies between them.	29
7	Excerpt from a World of Warcraft entity position trace.	36
8	Position trace of an example World of Warcraft entity.	37
9	Acceleration histogram for the World of Warcraft entity in Figure 8.	38
10	95% upper tolerance bound ($\alpha = 0.05$) across the range of expected update periods and latencies. Response surface derived from World of Warcraft replays.	39
11	Contour plot of the upper tolerance surface in Figure 10. Contours are plotted for plausibility limits from $L = 0.1$ m to $L = 1.1$ m in increments of 0.1 m.	40
12	Jump time distribution for the plausible to implausible state transition with plausibility limit = 0.1, latency = 0.035, and update period = 0.020.	41
13	Jump time distribution for the plausible to implausible state transition with plausibility limit = 0.1, latency = 0.5, and update period = 0.3.	42
14	Boxplot showing percentage of time spent in the plausible state for model predictions and validation sets.	43

Figure Page

15 Update period as a function of plausibility limit with a delay of 100 ms and maximum acclerations of 1, 5, and 10 G. Blue and red lines indicate 100 ms and 300 ms update periods respectively. 44

16 Deviation between the 95% upper tolerance bound $(1 - \alpha = 0.05)$ and plausibility limit $L = 1.0$ for an entity with normally distributed acceleration $N(0, 1)$. 45

17 Performance comparison of plausibility-based periodic and dead-reckoning update policies across a range of plausibility limits ($L = 0$ to $L = 5$) with a nominal latency of 100 ms. Performance is expressed as a percentage of total frames within the plausibility limit, i.e., $D_t(S_c, S_s) \leq L$. 46

18 Performance comparison of plausibility-based periodic and dead-reckoning update policies across a range of plausibility limits ($L = 0$ to $L = 5$) with a nominal latency of 300 ms. Performance is expressed as a percentage of total frames within the plausibility limit, i.e., $D_t(S_c, S_s) \leq L$. 47

List of Tables

Table		Page
1	Consistency Guarantees [49].	7
2	A sequence of writes for a simple distributed virtual environment.	10
3	Possible read results for each consistency model. Correct states are shown in bold. Stale states are shown in italics. Invalid states are shown in plain type. Ordered pairs represent (triangle position, square position).	10

List of Abbreviations

Abbreviation		Page
LVC	Live-Virtual-Constructive	1
DIS	Distributed Interactive Simulation	1
HLA	High Level Architecture	1
TENA	Test and Training Enabling Network Architecture	1
T&E	Test & Evaluation	1
DVE	Distributed Virtual Environment	2
CAP	Consistency-Availability-Partition Tolerance	6
TSI	Time-Space Inconsistency	13

A STOCHASTIC MODEL OF PLAUSIBILITY IN
LIVE-VIRTUAL-CONSTRUCTIVE ENVIRONMENTS

I. Introduction

Live-Virtual-Constructive (LVC) simulations are complex systems comprising a combination of live (real people operating real equipment), virtual (real people operating simulated equipment, or vice versa), and constructive (wholly simulated) entities interacting in a virtual space [23]. Entities may represent individuals, equipment, systems, or groups of these such as a platoon or battalion and represent the actors within the simulation. Physically, LVC simulations are structured as a set of nodes consisting of compute, storage, input/output, and simulation resources connected via telecommunications networks. Nodes support the simulation of one or more entities and are often geographically distributed to leverage unique assets, e.g., physical test range space or high-fidelity full motion simulators. Nodes are often connected in a peer-to-peer fashion and communicate using protocols such as Distributed Interactive Simulation (DIS) [12], the High Level Architecture (HLA) [9] or the Test and Training Enabling Network Architecture (TENA) [40].

Distributed LVC simulation promises a number of benefits for the test and evaluation (T&E) community, including reduced costs, access to simulations of limited availability assets, the ability to conduct large-scale multi-service test events, and recapitalization of existing simulation investments [14]. Consequently, the Department of Defense (DoD) is increasingly turning to LVC simulation and virtual environments to support T&E events. LVC simulations have been used to test communications for unmanned aircraft systems [39], conduct cyber security analysis [51], and quantify

radar measurement errors [22].

LVC simulations are typically designed as fully replicated, geographically distributed database applications with real-time constraints. Consequently, each simulation node maintains a complete copy of the state database for all simulated entities which must remain synchronized throughout a simulation run. The data of interest is composed of entity and world state information and derived quantities such as collisions or weapons effectiveness. Entity state data is replicated to meet availability and responsiveness requirements. The inclusion of live entities and users imposes real-time constraints on database responsiveness since long read/write latencies cannot be tolerated. Consequently, entity state updates (i.e., writes to a database record) are applied locally before propagation to other system nodes. Receipt of state updates is delayed due to network latency, queuing, and system architecture [21]. Thus, not all nodes see the same simulation state at the same time. If updates cease, the system will eventually become consistent [50]. As such, LVC simulations can be viewed in the same context as eventually consistent distributed datastores such as Amazon's Dynamo [10], Cassandra [28], or Megastore [5].

For any eventually consistent distributed database, a fundamental question is "How eventual is eventual?" Common measures of eventual consistency are time (how long does it take for readers to see the result of a write) and versions (how many versions old is a given read result) [3]. For LVC simulations and other distributed virtual environments (DVEs), entity deviation (e.g., Euclidean distance) from a "true" value is a common measure of consistency [53, 2, 55].

Quantifying the numerical error associated with eventual consistency is the first key challenge for LVC simulations. There is a growing body of literature characterizing the consistency of distributed databases such as Dynamo and Cassandra [52, 41, 4], these works focus on time or version staleness as the measure of consistency. More-

over, distributed databases are designed to support read-heavy workloads, are often distributed across a datacenter rather than geographically, and tolerate relatively long (on the order of a few seconds) responses to an update. In contrast, LVC simulations are more concerned with numerical error, have a balanced workload due to reading and writing all entity states at each time step, are distributed geographically, and require rapid response and synchronization times to minimize user frustration and state divergence. Consequently, existing research into the effects of eventual consistency on distributed databases is of limited applicability to LVC simulation.

A second challenge lies in assessing whether inconsistencies in the replicated state, reflected as measurement errors, lie within a specified precision tolerance. This assessment must be conducted during system design to ensure the simulation is capable of meeting precision requirements. Additionally, it must occur during execution to provide a quantification of the uncertainty associated with each measurement. Quantifying the uncertainty associated with the simulation state is crucial to assessing the quality of simulation outputs [38].

A third challenge for LVC simulations is assessing the quality of measurements without a known truth value, particularly for a discrete value [36]. This is especially true for derived quantities that depend on inconsistent state data such as collisions and weapons effects. In this case, each interacting node may compute a result that is correct according to its state replica yet different from other interacting nodes. Furthermore, the uncertainty can vary based on the node taking the measurement.

This research presents a model of LVC simulation data quality in terms of plausible outcomes for entity interactions. A key observation is that inconsistent data is usable if the inconsistency is small enough that all interacting parties agree on the outcome. If such a limit can be specified, then *a priori* estimates of the data quality can be obtained from the model. Thus, this work defines plausibility in terms of

the inconsistency of simulation state between replicas and an *a priori* limit on the maximum allowable inconsistency. If the inconsistency exceeds the limit, then one participant in the interaction may view the results as implausible. This can lead to user disengagement or erroneous measurements. A formal model of plausibility in LVC simulations is presented and a method of estimating the probability of exceeding the plausibility limit based on statistical tolerance intervals is derived. Estimation of exceedance probabilities provides insight into a simulation's ability to meet its interaction requirements and fitness for purpose (i.e., entertainment, training, or T&E). Moreover, computing exceedance probabilities provides a quantification of the uncertainty associated with a measurment taken from an LVC simulation – that is, one can say that a given measurement is within a certain tolerance with probability p and confidence $1 - \alpha$. Experiments with World of Warcraft player traces show that the model of plausibility and exceedence probability estimates presented here are effective in determining the probability a particular entity interaction is within its associated plausibility limit.

Plausibility as defined here is one component of a simulation's quality of entertainment or service [44]. Other typical components of quality of service are latency and responsiveness. Both latency and responsiveness relate to a user's overall satisfaction with an LVC simulation, while plausibility is related to the correctness of outcomes. Latency and responsiveness issues have been well studied, while the literature on plausibility of interaction outcomes is relativley sparse. Consequently, this work presents novel and complementary methods to ascertain the quality and correctness of LVC simulations.

The remainder of this work is organized as follows: Chapter II provides an overview of the relevant literature. Chapter III outlines a stochastic model of plausibility. Chapter IV presents the results of model validation experiments. Chapter V

concludes. The appendices include several published conference and journal articles comprising the bulk of this research.

II. Background

This chapter provides a review of the literature related to assessing plausibility in distributed virtual environments and LVC simulation. It discusses general consistency models for distributed systems, including replicated datastores, and the Consistency-Availability-Partition Tolerance (CAP) theorem. This discussion is followed by a review of the DVE state synchronization literature. Finally, some background on statistical tolerance intervals is presented.

2.1 Consistency Models

Maintaining consistency across data replicas is a fundamental issue in distributed systems [47]. Data is replicated to enhance reliability and improve performance. Replicating data enhances reliability by making multiple copies available to clients; if one copy is unavailable, the client can read or write to another replica immediately. This allows system nodes to be taken down for maintenance and reduces the impact of unforeseen network partitions. System performance can be improved by replicating data near clients; locality matters. Placing copies of the data near clients improves read/write response times. Additionally, it can improve system scalability by spreading processing load over multiple servers. Tanenbaum and Van Steen [47] provide the standard text on consistency in distributed systems. Their work discusses data and client-centric consistency models, replica management techniques, and consistency protocols.

The advent of commercially available distributed key-value stores such as Amazon's Dynamo [19], Cassandra [28], and Project Voldemort [13] has led to renewed interest in eventually consistent replication models. Terry [49] provides an overview of six consistency guarantees found in modern replicated database systems. They are

based on a simple model in which clients read and write a generic data store. The data store is replicated across a set of servers and clients may access it concurrently. Writes are ordered and eventually received and applied by all servers. Reads return previously written values, although the value returned by a read operation may not be the latest value written. Consistency guarantees dictate the allowable return values for a read and are defined by the set of previous writes visible to a particular read operation. The consistency guarantees described by Terry are listed in Table 1 and summarized in the following sections.

Table 1. Consistency Guarantees [49].

Strong Consistency	See all previous writes.
Eventual Consistency	See subset of previous writes.
Consistent Prefix	See ordered sequence of previous writes.
Bounded Staleness	See all "old" writes.
Monotonic Reads	See increasing subset of writes.
Read My Writes	See all writes performed by reader.

Strong Consistency.

Strong (or absolute) consistency guarantees that a read of a data object returns the last value written to it. This is clearly a desirable characteristic for DVEs as it ensures replicas remain perfectly aligned and the illusion of a single shared environment is maintained. However, strong consistency comes at the cost of reduced availability and throughput as the database system must replicate any write operation before processing further requests.

Eventual Consistency.

Eventual consistency is the weakest consistency model described by Terry [49], guaranteeing only that the database replicas will be consistent at some future time after all updates have ceased. Consequently, a read operation is allowed to return

any value that has ever been written. While this may seem to be an overly loose model of consistency, in practice, eventual consistency is often good enough [26] and can perform quite a bit better than strongly consistent systems [1].

Consistent Prefix.

Consistent prefix guarantees that a reader will see an ordered sequence of writes beginning with the first write to the data store. It ensures that the data returned by a read operation was actually written at some point in the past.

Bounded Staleness.

Bounded staleness ensures the result of a read operation is not too out of date [47, 49]. Staleness is typically defined in terms of a time bound T, e.g., 5 minutes. Reads are guaranteed to return values written T minutes ago (or more recently). Other definitions of staleness are possible; for example, k-staleness returns a value within the last k versions rather than from a fixed time window [4].

Monotonic Reads.

Monotonic reads is a session guarantee that applies to sequence of reads performed by a particular client [50]. Like eventual consistency, monotonic reads allow reading arbitrarily stale data; however, it guarantees the results of a sequence of reads is increasingly up-to-date.

Read My Writes.

Read my writes also applies to a sequence of reads performed by a particular client [47, 49]. This consistency model guarantees that a reader sees the effect of its own writes or writes performed more recently by another client. Read my writes provides

no guarantees beyond eventual consistency for data read by other clients or for clients who have performed no writes.

To illustrate these consistency models, consider the simple distributed virtual environment depicted in Figure 1. Here, two clients are connected via a network. Each maintains a model of a single entity represented by the square and triangle, as well as a replica of the entire state database. For simplicity, assume the entities are constrained to movement in a single dimension as illustrated in Figure 2. Furthermore, assume the state data of interest is the entity position.

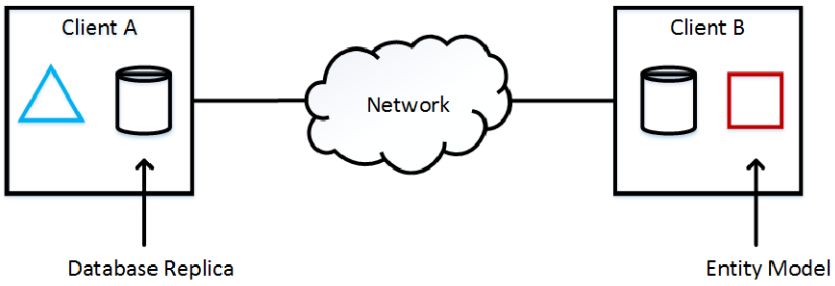


Figure 1. A simple two-entity distributed virtual environment.

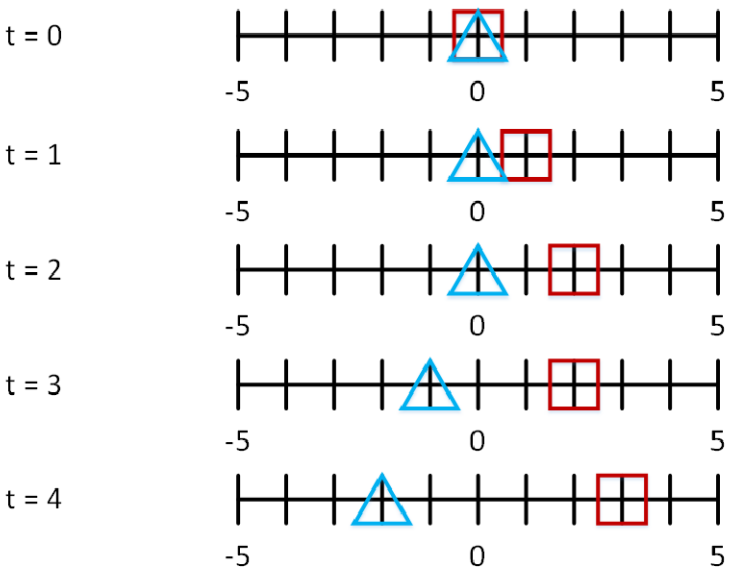


Figure 2. State changes for a simple two-entity distributed virtual environment.

Each client writes to its local database replica as it updates the position of its

modeled entity. Updates are propagated to the other client in some fashion; the details of the replication protocol are irrelevant for this example. Table 2 presents a sequence of writes to the DVE database, i.e., state updates. The effect of these writes on the DVE state is illustrated in Figure 2.

Table 2. A sequence of writes for a simple distributed virtual environment.

$t = 0$	Write("Square",1)
$t = 1$	Write("Square",2)
$t = 2$	Write("Triangle",-1)
$t = 3$	Write("Square",3)
$t = 4$	Write("Triangle",-2)

For each consistency model, Table 3 presents the possible results of issuing a read after all writes have taken place at $t = 4$. Each ordered pair in the table represents the state of the DVE. The first coordinate is the triangle entity's position, and the second is the square's position. The correct state at $t = 4$ is shown in bold. Stale states that were correct at some time in the past ($t < 4$) are shown in italics. States that never existed but may be returned by a consistency model are shown in plain type.

Table 3. Possible read results for each consistency model. Correct states are shown in bold. Stale states are shown in italics. Invalid states are shown in plain type. Ordered pairs represent (triangle position, square position).

Strong Consistency	**(-2,3)**
Eventual Consistency	*(0,0)*, *(0,1)*, *(0,2)*, (0,3), (-1,0), (-1,1), *(-1,2)*, (-1,3), (-2,0), (-2,1), (-2,2), **(-2,3)**
Consistent Prefix	*(0,0)*, *(0,1)*, *(0,2)*, *(-1,2)*, **(-2,3)**
Bounded Staleness ($T = 2$)	*(0,2)*, *(-1,2)*, (-1,3), (-2,2), **(-2,3)**
Monotonic Reads (after reading (0,2))	*(0,2)*, (0,3), *(-1,2)*, (-1,3), (-2,2), **(-2,3)**
Read My Writes	
Square (writer)	(0,3), (-1,3), **(-2,3)**
Triangle	*(0,0)*, *(0,1)*, *(0,2)*, (0,3), (-1,0), (-1,1), *(-1,2)*, (-1,3), (-2,0), (-2,1), (-2,2), **(-2,3)**

All of the consistency models may return the correct current state of the environment; however, only the strong consistency model is guaranteed to do so and in

fact, this is the only value strong consistency may return. Eventual consistency, on the other hand, may return any combination of values that were written, even if the resulting state never actually occurred.

This latter property of reading states that never existed is shared by the bounded staleness, monotonic reads, and read my writes models to varying degrees. These models differ chiefly in the size of the set of possible read values. Consistent prefix, on the other hand, guarantees all possible read values at least existed in the database at some point although the returned value may be arbitrarily old.

Note that in addition to returning non-existent states, the weak consistency models can return results that violate causality. In practice, this can be mitigated by imposing a global ordering on messages via GPS synchronization, vector clocks [29], or similar.

2.2 CAP Theorem

The Consistency-Availability-Partition Tolerance (CAP) theorem [6], sometimes called Brewer's conjecture or Brewer's theorem, establishes a trade-space for three fundamental properties of distributed systems. These properties are 1) consistency – do all parts of the distributed system have the same data value at the same time, 2) availability – is the system available for reading or writing on demand, and 3) partition tolerance – will the entire system cease functioning if one or more system nodes are isolated. Brewer set these properties in opposition to one another (i.e., as trade-offs) and conjectured that only two properties could be fulfilled at any given time, and then only by relaxing the requirement for the third property. Gilbert and Lynch [15] provided a proof of Brewer's conjecture in 2002.

As an example, consider a two-node distributed datastore. Suppose the network between them is severed. Then the system can either continue to accept reads/writes

to either node, sacrificing consistency, or it can force all read/writes to occur on a single node, sacrificing availability.

2.3 State Synchronization

The problem of state synchronization in distributed virtual environments, of which LVC simulations are a subset, has been well-studied. This section presents an overview of the existing literature on error in distributed virtual environments and state update and synchronization policies.

Types of Error in Distributed Virtual Environments.

Several types of error arise in distributed virtual environments as a consequence of the loose consistency guarantees provided by the state database. Let $e_s(t)$ and $e_c(t)$ be the state of some entity as stored in the server and client database replicas at time t. For any entity state for which there is an appropriate norm (e.g., position), we can define the spatial error at time t, $\mathrm{SE}(t)$, as the distance between the server and client representations of the entity state. That is,

$$\mathrm{SE}(t) = \|e_s(t) - e_c(t)\| \tag{1}$$

Aggarwal, et al. [2] assumes a state update policy based on server-side dead reckoning and decomposes the spatial error into two components: dead reckoning error and export error. Once again, let $e_s(t)$ and $e_c(t)$ be the state of entity e as represented in the server and client state databases. Further, assume the server also maintains a dead reckoned version of e as a means of predicting the spatial error at the client. Let $\hat{e}_s(t)$ represent this approximation (or estimate). Then the dead reckoning error $\mathrm{DE}(t)$ is given by

$$\mathrm{DE}(t) = \|e_s(t) - \hat{e_s}(t)\| \tag{2}$$

If at any time t the dead reckoning error exceeds some threshold, h, an update is sent to the client and $\hat{e_s}(t)$ is reset to the true entity state $e_s(t)$. Due to propagation delay factors such as network latency and processing time at the client, the entity state at the client will differ from the dead reckoned approximation at the server, i.e., $\hat{e_s}(t)$ and $e_c(t)$ are not necessarily equal – $\hat{e_s}(t)$ is an estimate of $e_c(t)$. This difference is termed the export error, $\mathrm{EE}(t)$, and is given by

$$\mathrm{EE}(t) = \|\hat{e_s}(t) - e_c(t)\| \tag{3}$$

The total spatial error is thus the sum of the dead reckoning error at the server and the export error at the client, i.e.,

$$\mathrm{SE}(t) = \|e_s(t) - e_c(t)\| = \mathrm{DE}(t) + \mathrm{EE}(t) \tag{4}$$

Integrating spatial error with respect to time yields a consistency measure known as time-space inconsistency (TSI) [55], i.e.,

$$\mathrm{TSI}(t_0, t_1) = \int_{t_0}^{t_1} \mathrm{SE}(t)\ \mathrm{d}t \tag{5}$$

and draws an equivalence between large spatial errors of short duration and small errors of long duration. TSI has been used as the basis of a number of state update scheduling algorithms and policies [48, 30, 33, 31, 32, 34]. Most of these policies employ some form of server-side dead reckoning; consequently, the TSI can still be decomposed into dead reckoning and export errors.

There are some drawbacks to the use of TSI as an error measure. First, for certain classes of entity motion, TSI can grow without bound [42]. Second, choosing

a dead reckoning threshold can be somewhat counter-intuitive. Finally, and of most importance to DVEs in a test and analysis context, the conflation of short duration, large distance errors with long duration, short distance errors can make it difficult to determine if a measurement based on the replicated state is within tolerable limits.

A second form of error arises with respect to the age of a state variable. Hodson [21] defines the correctness of a variable as a function of a time interval – a state variable is accurate or valid for a period of time after being updated. This period of time is called the validity interval. A variable is deemed to be temporally consistent if the time of its last update, t_L, plus its validity interval, t_{VI}, is greater than or equal to the current time, i.e., if $t_L + t_{VI} \geq t$ [20].

The temporal consistency of a state variable as seen by a client can be measured by calculating the mean and variance of its age in the client's state database. Figure 3 illustrates how a state variable ages at the client. An update received at t_i may have already aged due to transmission delays; in any case, the variable ages linearly until the next update is received. The interarrival time of the updates is denoted by λ_i and there are N aging intervals.

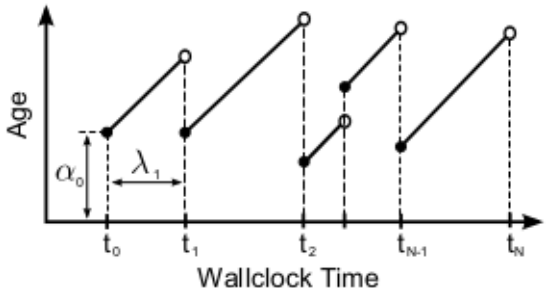


Figure 3. Aging of distributed state data [20].

The mean age of the entity state as seen by the client is given by

$$\mu_{e_c} = \frac{1}{t_N} \sum_{i=1}^{N} \left(\frac{\lambda_i^2}{2} + \alpha_{i-1}\lambda_i \right) \tag{6}$$

where e_c is the entity state as represented in the client's state database, λ_i is the

interarrival time, and t_N is the total elapsed wall-clock time over N intervals [20].

The variance of the entity state's age at the client is given by

$$\sigma^2_{e_c} = \text{mse} - \mu^2_{e_c} \tag{7}$$

where

$$\text{mse} = \frac{1}{t_N} \sum_{i=1}^{N} \left(\frac{\lambda_i^3}{3} + \alpha_{i-1}\lambda_i^2 + \alpha_{i-1}^2 \lambda_i \right) \tag{8}$$

The validity interval is determined by properties associated with how the state changes in relation to time. For any two interacting entities, there is a maximum acceptable error beyond which the interaction fails to behave correctly. For instance, spatial errors that are too large may cause a collision detection routine to trigger late, resulting in intersecting entities rendered to the display. Simulations supporting analysis can ill afford such imprecision, particularly for interactions comprising the system-under-test.

Given a precision requirement for each interaction supported by the simulation, the accuracy of the state database can be defined as follows. A replica is accurate with respect to entity e at time t if the spatial error for the entity is less than its associated precision requirement. This is a binary condition; the replica is either accurate or not. Formally, the accuracy at client c with respect to entity e is given by

$$A_e(t) = \begin{cases} 1 & \|e_s(t) - e_c(t)\| \leq p \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where $e_s(t)$ is the entity state at the server, $e_c(t)$ is the entity state at the client, and p is a (spatial) precision requirement.

The overall accuracy $\bar{A}(t)$ of a client is given by the mean accuracy over all entities, i.e.,

$$\bar{A}(t) = \frac{1}{|E|} \sum_{e} A_e(t) \tag{10}$$

where $|E|$ is the total number of entities in the simulation. Calculating the mean and variance of $A_e(t)$ and $\bar{A}(t)$ over time provides summary measures of a particular client's state database accuracy.

State Update Policies.

A variety of state update mechanisms and policies have been proposed and investigated by the research community. The most basic policies are round-robin and periodic policies[45, 46]. Round-robin state update simply places each client in a circular queue based on the time it was last sent an update. At each frame, the server updates as many clients as possible, starting with the client least recently updated. This policy tends to perform poorly in systems where the total number of clients is much larger than the number of clients that can be updated at each simulation frame.

Periodic update policies send updates to clients at defined intervals. Note that the round-robin policy is in fact periodic, however, much more complex periodic schemes are possible. For instance, some DVE systems provide state update data streams with multiple periods for the same entity. This allows clients to choose how often they receive updates – clients whose entities are not interacting with the server's choose a long period stream, while clients needing more state accuracy can choose a high frequency stream. See [37] for a review of this and other interest management techniques.

Mauve [36] proposed the use of local lag policies to improve state consistency among nodes in distributed virtual environments. Local lag delays the effect of user

inputs at the local node while sending a state update to the remote nodes. Ideally, the lag applied locally is equivalent to the propagation delay of the state update message and all nodes see the effects of the input simultaneously. Practice is rarely so accommodating since each client may have a different amount of delay, packets may be dropped, and network latency can vary. Note that local lag degrades responsiveness in favor of improved state consistency.

The DIS standard specifies the use of dead reckoning algorithms to estimate state consistency between system nodes. The vast majority of research on data consistency issues in DVEs focuses on improving and optimizing dead reckoning for a variety of conditions. An early effort by Cai, et al. [7] proposed an auto-adaptive error threshold based on areas of interest and sensitive regions around each entity. The area of interest is defined as a circular region around an entity in which the entity requires increased consistency. The sensitive region is a smaller circular region. If one entity moves into another's sensitive region, a collision or some other kind of interaction is likely. Thus the consistency requirements inside an entity's sensitive region are higher still. The authors define four threshold levels based on the relative position and overlap of entities and their regions. They conducted a series of experiments using a simulated distributed environment and measure the average spatial error and number of messages for differing numbers of entities. Their adaptive threshold algorithm shows an improvement in both the number of messages sent and the average error. Unfortunately, they provided no guidance for choosing the threshold values for their algorithm. This is important since the superiority of their approach depends on these choices. In general, however, one can expect the adaptive algorithm to strike an adequate balance between consistency and network utilization while showing an improvement over standard dead reckoning.

Liu [35] explored the use of communication subgraphs to determine the optimal

synchronization interval for DVEs with an upper bound on spatial error. He compared the use of several subgraph generation algorithms to bound the maximum delay between nodes in the virtual environment. The optimal synchronization interval was determined using a discrete time Markov chain model of the spatial error. Once the optimal interval is found, the network utilization for each of the communication subgraphs can be computed.

Delaney, et al. [11] proposed a hybrid algorithm that pairs the conventional dead reckoning model with an experimentally derived model of the user's long term strategy. The goal of the hybrid technique is to reduce network utilization at a given threshold level. Their work shows an improvement in the number of messages sent, albeit for a relatively large spatial error threshold.

Yu, et al. [54] consider the problem of allocating bandwidth to nodes in order to minimize the spatial error, subject to time-varying bandwidth constraints. They construct the problem as a constrained convex optimization problem and apply Lagrangrian relaxation to make the trade-off between consistency and network utilization. Minimization of the relaxed problem is accomplished with binary search and predictive model of bandwidth consumption based on recent history. Their work provides an empirical look at the fundamental DVE trade-off between consistency and bandwidth consumption. Although they describe in general terms a network aware bandwidth allocation algorithm, and provide results showing it outperforms uniform and proportional allocations, the algorithm is not specified in detail.

Note the preceding works attempt to optimize dead reckoning with respect to network consumption rather than consistency. Excessive bandwidth consumption is generally not an issue except when scaling a DVE to massive numbers of clients. In many practical scenarios, the number of clients is bounded and minimizing the number of messages is not required – after all, unused bandwidth is wasted bandwidth.

The remainder of this section discusses techniques that specifically optimize dead reckoning for consistency.

Building off Zhou *et al.*'s work on time-space inconsistency [55], Roberts *et al.* proposed the use of a time-space threshold instead of a spatial error threshold for dead reckoning [43]. They begin by noting that simple spatial thresholds can lead to unbounded inconsistency if the entity deviates from the dead reckoned path but remains inside the error threshold. Using a time-space threshold mitigates this problem, however, doing so can lead to large spatial errors over short time periods. The authors demonstrate a hybrid threshold metric that avoids both of these issues.

Tang and Zhou [48] developed a consistency aware update scheduling algorithm for centralized, single server DVEs with network capacity constraints. They begin by deriving optimal update schedules to minimize the impact of time-space inconsistency on user perceptions, noting that these schedules depend on the network delay and entity trajectories. The latter, of course, depend on user inputs to the simulation. The authors then present an update scheduling algorithm that estimates the time-space inconsistency at each client and sends updates to the clients with the largest error, subject to network capacity constraints. Any clients that are not updated will necessarily have a larger inconsistency during the next frame and will have a higher priority for update. Their algorithm shows substantially better performance than more naive scheduling algorithms such as round-robin or conventional dead reckoning. Li and Cai [31] extend the discussion to a multi-server environment and frame the problem as one of determining the optimal update period for each replica. They formulate the problem as convex optimization problem with inequality constraints and apply the barrier method for solution.

Both of the foregoing methods assume static network conditions with no possibility of message loss. Li *et al.* [32] consider the effects of message loss on update scheduling;

however, they ignore the effects of network delay. They develop scheduling algorithms that compensate for message loss by sending updates when the time-averaged inconsistency with loss exceeds the inconsistency without loss. Li, *et al.* [34] propose an update scheduling algorithm that modifies the dead reckoning threshold based on predicting network conditions and estimating whether the system can compensate for the effects of network delay and message loss. If the delay and loss rate can be compensated, the threshold is modified accordingly; if not, an update message is sent immediately.

2.4 Tolerance Intervals

Tolerance intervals are statistical intervals containing a specified proportion p of a sample population with confidence $1 - \alpha$ [17]. They are useful in drawing conclusions about a large number of future samples based on data from a random sample. One-sided tolerance bounds are often used to calculate exceedance probabilities, that is, the probability that a random variable exceeds a specified value [27]. They are inversely related to confidence bounds. Tolerance intervals can be calculated non-parametrically using Wilke's algorithm and order statistics. Algorithm 1 presents Matlab code for calculating non-parametric tolerance intervals using Wilke's algorithim.

2.5 Summary

Maintaining consistency across replicas is a fundamental concern for designers of LVC simulations. Several consistency models spanning the spectrum from eventual to absolute consistency are available; however, none but absolute consistency is guaranteed to provide the current state of the database in response to a read request at all replicas. State consistency is a desireable characteristic for LVC simulations to

Algorithm 1 Wilke's Algorithm for non-parametric tolerance intervals.

```
function [lower,upper]=nptol2(x,alpha,P)

n=length(x);
sorted=sort(x);
upper=repmat(max(x),1,length(P));
lower=repmat(min(x),1,length(P));

for i=1:length(P)
    r=binoinv(alpha,n,1-P(i));
    s=n-r+1;

    if(r>=1)
        lower(i)=sorted(r);
    end

    if(s<=n)
        upper(i)=sorted(s);
    end
end
```

ensure the illusion of a shared virtual environment. However, LVC simulations have strict real-time availability requirements due to the inclusion of live assets and must continue running if one or more sites or nodes are removed from the network. As a consequence of the CAP theorem, these availability and partition tolerance requirements preclude absolute consistency between all state database replicas.

Several algorithms have been developed to minimize the inconsistency between replicas. These include simple dead-reckoning schemes, local lag, consistency aware state update algorithms, methods based on time-space inconsistency, and attempts to predict network conditions and the user's long-term strategy. Importantly, none of the existing state update algorithms provide an assessment of how consistent the state database replicas are at any point in time. Moreover, depending on the specific consistency guarantees provided by the underlying distributed system, they may allow reads to a replica to return states that are erroneous and were never written

to the database. Consequently, the results of inter-entity interactions may appear implausible to one or more users of the simulation.

If limits are placed on the maximum allowable state divergence for a given entity interaction to ensure plausible outcomes, tolerance intervals can be applied to predict how often an LVC simulation is expected to stay within the limits with a specified confidence. The next chapter develops a prediction method based on these plausibility limits.

III. Modelling Plausibility in LVC Simulation

LVC simulations are geographically distributed, fully replicated database applications with real-time constraints. The inclusion of man-in-the-loop simulations and actual live (real-world) assets forces a preference for system availability and the ability to continue operating if one or more subordinate simulations are removed from the system. Consequently, LVC simulations cannot guarantee absolute state consistency across replicas due to the CAP theorem. The lack of absolute consistency raises the possibility of two interacting entities observing different outcomes for a given interaction – an undesireable characteristic that can lead to user frustration and disengagement [8] or erroneous conclusions for a T&E event.

Useful results can be obtained from an LVC simulation if the inconsistencies are small enough to allow each entity to accept the outcome of an interaction (i.e., deem the outcome plausible) regardless of their particular view of the situation. That is, if the result obtained from an interaction is plausible given the state of each entity's replica, then the state inconsistency is minor enough to be ignored. This chapter presents a formal definition of plausibility for LVC simulations and a simple stochastic model suitable for estimating how likely the result of an interaction will be plausible.

3.1 LVC Architecture

Each node in an LVC simulation hosts one or more *entities* representing objects such as tanks or aircraft. Entities represent real-world objects such as an aircraft on a training range (live entities), man-in-the-loop simulators (virtual entities), or wholly simulated assets (constructive entities). Nodes are responsible for processing user inputs, computing physics models, and maintaining state for their hosted entities. Additionally, nodes send entity state updates to other nodes so that each can maintain

a representation of the overall simulation state. In the context of a single entity, the hosting node is termed the *server* while all other nodes are *clients* or *replicas*. Note that each node in the system acts as both server and client – a node is the server for each entity it simulates and a client for all others. This is sometimes referred to as a serverless architecture.

Simulation state data is divided into two types: 1) static and 2) dynamic. Static state data includes items such as terrain geometry and other information that does not change over the course of the simulation. Static state is usually distributed to all nodes prior to beginning the simulation. Dynamic state data consists of any information about the entities or environment that can change over time such as entity position and orientation or weather. Dynamic state may be discrete or continuous. The sharing of dynamic state between nodes allows users to interact with one another as if they inhabit the same virtual space.

In order to maintain dynamic shared state, LVC simulations employ a fully replicated, eventually consistent distributed database. Each record in this database holds the state data for a single entity. Nodes host replicas of the state database. Nodes write only to the records corresponding to the entities they host; all other records are read-only. Records are updated across replicas according to some policy – common policies include sending updates periodically or when a predicted error crosses some threshold value. The policy need not be the same for all replicas. The state database is eventually consistent in the sense that once all user inputs cease, all replicas will converge to the same state. In the meantime, reads from different replicas may return different values and in fact may return any value written since the beginning of the simulation [49].

Eventually consistent databases are employed by LVC simulations because supporting live and virtual entities imposes real-time processing constraints in order to

ensure acceptable levels of fidelity. Response times to user inputs for man-in-the-loop simulations should be less than 100 ms in order to avoid user-induced oscillation effects [37]. These requirements, coupled with minimum data propagation delays, imply that absolute consistency in distributed real-time simulation is unachievable. Only after all inputs have ceased can each replica be guaranteed to have correct state data. Consequently, client-side state prediction algorithms are widely deployed to mitigate state errors between receiving updates from the server.

Figure 4 illustrates the basic structure of an LVC simulation. At the top of the figure are two entities, in this case two aircraft. These entities interact with one another in the virtual environment; for example, the fighter may try to intercept the bomber. The entities and their interactions are simulated by a pair of nodes communicating across a network as illustrated in the middle portion of Figure 4. Note that each simulation node acts as a server for its own entity and client for the other entity. Finally, the bottom portion of the figure depicts the dynamic shared state of the simulation, e.g., positional information. The state database consists of a position record for each entity. Records may be inconsistent, depending on network latency and other factors, as illustrated by the arrows labeled "Truth" and "Perceived" state.

3.2 Modeling Plausibility

The CAP theorem has an important consequence for LVC environments related to the plausibility of interaction outcomes as observed at the server and clients. Since LVC simulations prioritize availability and partition tolerance above consistency, state divergence between the server and client is virtually guaranteed. Frequently, this divergence is of little consequence. However, when the server and client must independently compute some result based on the same input variables at the same time, problems with the plausibility of outcomes can arise due to state divergence. For

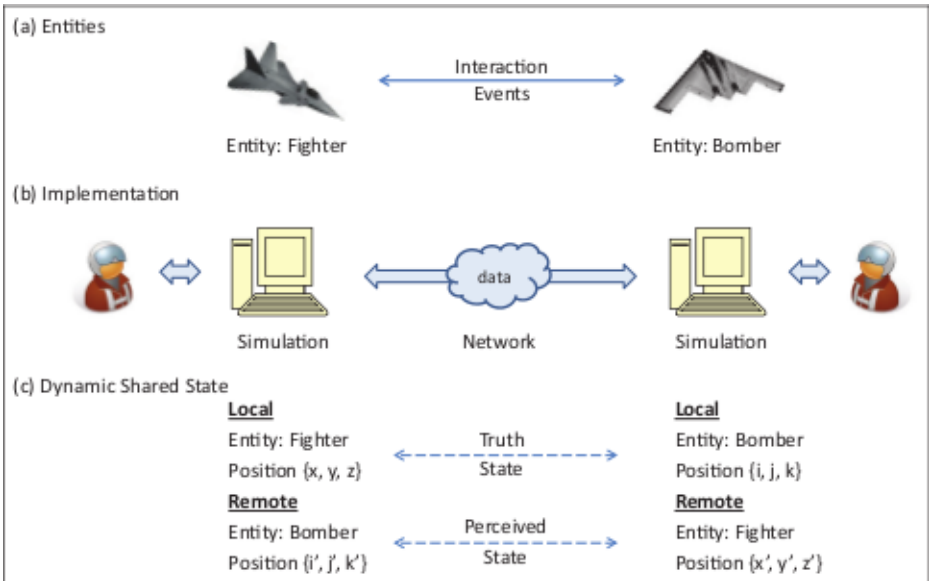


Figure 4. Entities, implementation, and dynamic shared state [23].

instance, if the interaction of interest is collision detection and the positional state of involved entities is not consistent, one node may detect a collision while another may not. This problem is exacerbated when outcome arbitration is completed – one node's version of the state may well be overwritten by a completely different result, thus yielding an implausible outcome. Steed and Oliveira refer to this as joint plausibility, i.e., the notion that two or more users accept that they are viewing the same simulation of a shared space [46].

Note that plausibility does not imply that the server and client compute identical interaction results. Rather, joint plausibility requires that the computed results be close enough that users are willing to accept the arbitrated results and not reject them out of hand as nonsensical. That is, dead men must not keep shooting [36] due to a divergence in states between client and server. The maximum tolerable state divergence resulting in plausible outcomes is interaction dependent [24]. This section presents formal definitions and a simple stochastic model of plausibility.

Definition 1 *Let e_i and e_j represent entities within an LVC simulation. Then the tu-*

ple $I = (e_i, e_j, L)$ is called an interaction and specifies a state dependence relationship from e_i to e_j.

In this relationship, the node hosting e_j is the server and the node hosting e_i is the client. The quantity L is called a *plausibility limit*. Note the relationship is one-way in order to allow modeling of asymmetric entity interactions, e.g., long-range vs short-range sensors.

Definition 2 *Let S_c and S_s represent the state database replicas on the client and server, respectively. The state divergence, D, is given by*

$$D(S_c, S_s) = \|S_c - S_s\|$$

for some appropriately defined distance measure; i.e. Euclidean distance for entity position state.

The outcome of an interaction I at time t is *plausible* if $D_t(S_c, S_s) \leq L$.

Note that plausibility is a component of quality of entertainment [44] or quality of service; however, it is qualitatively different than the latency issues typically studied by distributed virtual environments researchers. It is possible for an LVC simulation or other DVE to deliver jointly plausible results while exhibiting unacceptable latency or responsiveness. For example, an LVC simulation often can be designed to deliver absolute consistency and therefore plausible outcomes; however, the responsiveness of such a simulation may be unacceptable to its users, particularly if it is distributed over large distances. Similarly, the algorithms described in Chapter II are designed to maximize state consistency and mask latency in order to improve quality of service. However, not all latency can be masked [2] and so plausibility issues may still arise.

These definitions lead to a simple two-state stochastic model of interaction plausibility as illustrated in Figure 5. With respect to an on-going interaction I, the

state divergence $D(S_c, S_s)$ is either within the plausibility limit L (depicted by the left-hand state) or exceeds the limit (the right-hand state). If the divergence is within the plausibility limit, then the outcome of the interaction on the client and server are in agreement by convention.

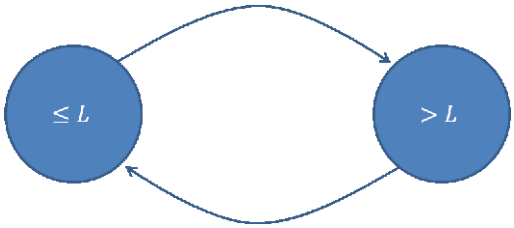


Figure 5. A simple two-state model of interaction plausibility.

Computation of the state transition probabilities associated with this model is difficult in general due to a strong dependence on the server state dynamics and the coupling between server and client states. Factors influencing the plausibility of an interaction and the dependencies between them are illustrated in Figure 6. As previously noted, the plausibility of an interaction outcome is a direct function of the server state, the client state, and an *a priori* limit on maximum tolerable error. Server state depends on the model being simulated (e.g., an F-16) and the user inputs to that model. Client state depends on the server state, end-to-end latency, and message loss rates. Note that latency and message loss are random variables and may not be directly observable by the LVC simulation. For entity position state, user input can be modeled as a random acceleration variable.

3.3 Computing Plausibility Exceedance Probabilities

Exceeding the plausibility limit for an interaction between two or more entities yields a loss of joint plausibility and incorrect results for at least one view of the world state. For entities with sufficiently smooth and well-behaved dynamics, the probability of exceeding the plausibility limit can be computed directly. Assuming

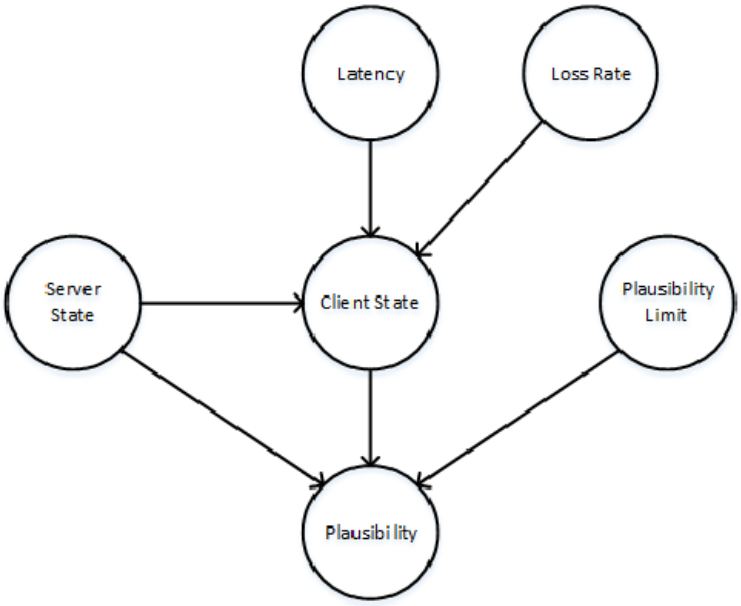


Figure 6. Factors influencing interaction plausibility and the dependencies between them.

message loss is negligible and state updates are sent from the server to the client periodically, the probability of exceeding the plausibility limit L for an interaction $I = (e_i, e_j, L)$ depends primarily on the rate of change of the server state and the latency between the server and client.

Let the *real path*, $\mathcal{R}(t)$, represent the true position of e_j at any time t. $\mathcal{R}(t)$ is a function of the user input and physics models employed by the simulation node n_j hosting e_j. At intervals, n_j sends a message updating the replica of e_j at the source node n_i. These state updates coupled with any dead reckoning algorithms in use at n_i yield the *placed path*, $\mathcal{P}(t)$. This path represents e_i's perception of e_j's position at any time t. The difference between the real and placed paths

$$E(t) = \mathcal{R}(t) - \mathcal{P}(t) \tag{11}$$

is the spatial error at time t.

Moreover, the position of the entity on the placed path after Δt seconds can be

approximated using Taylor series expansion [18],

$$\mathcal{R}(t+\Delta t) \approx \mathcal{R}(t) + \frac{\mathrm{d}\mathcal{R}}{\mathrm{d}t}\Delta t + \frac{1}{2}\frac{\mathrm{d}^2\mathcal{R}}{\mathrm{d}t^2}\Delta t^2$$

Letting $s(t) = \mathcal{R}(t)$, $v(t) = \frac{\mathrm{d}\mathcal{R}}{\mathrm{d}t}$, and $a(t) = \frac{\mathrm{d}^2\mathcal{R}}{\mathrm{d}t^2}$ yields the familiar kinematic equation

$$\mathcal{R}(t+\Delta t) \approx s(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$$

where $s(t)$, $v(t)$, and $a(t)$ represent the entity's position, velocity, and acceleration at time t.

The placed path at time $t + \Delta t$ depends on the dead reckoning algorithm in use at the replica node. If no dead reckoning is employed, then

$$\mathcal{P}(t+\Delta t) = \mathcal{P}(t) = s(t)$$

If first-order dead reckoning is employed, then

$$\mathcal{P}(t+\Delta t) = s(t) + v(t)\Delta t$$

Combining the expressions for the real and placed paths yields the following expressions for the spatial error at time $t + \Delta t$:

$$E(t+\Delta t) = v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{12}$$

in the case of no dead reckoning and

$$E(t+\Delta t) = \frac{1}{2}a(t)\Delta t^2 \tag{13}$$

in the case of first-order dead reckoning.

If the entity's maximum velocity and acceleration are known (a safe assumption) and t_{update} is the time the last update was sent, then the spatial error after receiving the update can be bounded above as

$$E(t_{update} + \Delta t) \leq v_{max}\Delta t + \frac{1}{2}a_{max}\Delta t^2 \tag{14}$$

and

$$E(t_{update} + \Delta t) \leq \frac{1}{2}a_{max}\Delta t^2 \tag{15}$$

for the no dead reckoning and first-order dead reckoning cases respectively.

For a simulation employing dead reckoning, the temporal accuracy interval [25] for a state update in the context of interaction I is computed as

$$\Delta t = \sqrt{\frac{2 * l}{||a_{max}||}} \tag{16}$$

Assuming state updates are sent from e_j to e_i periodically, the quantity Δt specifies the maximum time for which a state update is accurate. After Δt seconds, the update will have been superseded by another and the spatial error will have exceeded the tolerance of interaction I. Note that periodic state updates have been shown to result in optimal state consistency [48]. Thus, our concern is to find the largest inter-update period p that meets the error tolerance for interaction I.

To do so, we note that the temporal accuracy interval is the difference between the time the last update was sent, t_{update}, and the time of its use, t_{use}. That is,

$$\Delta t = t_{use} - t_{update}$$

Noting that the current value of the state becomes invalid when the next update is sent and letting d represent the total update propagation delay (accounting for net-

work latency, queuing, and internal delays associated with simulation architecture), the largest accuracy interval is obtained when

$$\Delta t = t_{update} + p + d - t_{update} \tag{17}$$

To summarize, let e_i be an entity interacting with entity e_j. Suppose the interaction has some maximum spatial error tolerance l. Let $||a_{max}||$ be the maximum acceleration of e_j and d be the state update propagation delay. Then the maximum period between state updates from e_j to e_i is given by

$$p = \sqrt{\frac{2 * l}{||a_{max}||}} - d \tag{18}$$

Guarantees on system latency require establishing a messaging policy such that

$$\frac{1}{2}||a_{max}||(p + d)^2 \leq l, \tag{19}$$

where a is the acceleration, p is the inter-update period, d is the delay, and l is the plausibility limit. Assuming ideal system design involves maximizing the inter-update period to minimize the number of messages sent, this research focuses on the plausibility limit in Equation 19 being exactly met, not exceeded, or

$$\frac{1}{2}||a_{max}||(p + d)^2 = l. \tag{20}$$

Solving for p gives results in Equation 18, which is effectively the largest value of p that meets the plausibility limit for specific values of a_{max} and d.

Within these equations, a_{max} and d are independent random variables and l is a constant set by the end users. As independent random variables, the joint distribution

of a_{max} and d is the product of their individual distributions, or

$$f_{A,D}(a,d) = f_A(a) f_D(d). \tag{21}$$

In this example problem, $A_{max} \sim$ Double Exponential(0,σ_1^2) and $D \sim$ Shifted Exponential(μ, σ_2^2), so

$$f_{A,D}(a,d) = \frac{1}{2\sigma_1} e^{-\frac{|a|}{\sigma_1}} \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}}. \tag{22}$$

Since p is a function of a_{max}, d, and l from Equation 18, a transformation can be performed on $f_{A,D}(a,d)$ into $f_{P,D}(p,d)$. Let

$$g(p) = p = \sqrt{\frac{2l}{a}} - d, \tag{23}$$

from Equation 18, then

$$g^{-1}(p) = a = \frac{2l}{(p+d)^2}. \tag{24}$$

Using a univariate transformation,

$$f_{P,D}(p,d) = f_{A,D}(g^{-1}(p),d) \left| \frac{\partial a}{\partial p} \right|.$$

Due to the change in monotonicity at $a = 0$, this transformation is actually the sum of two transformations

$$f_{P,D}(p,d) = f_{A,D}(|g^{-1}(p)|,d) \left| \frac{\partial a}{\partial p} \right|, a < 0$$

$$+ f_{A,D}(g^{-1}(p),d) \left| \frac{\partial a}{\partial p} \right|, a \geq 0$$

$$= \frac{1}{2\sigma_1} e^{-\frac{\left|\frac{2l}{(p+d)^2}\right|}{\sigma_1}} \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \left| \frac{\partial}{\partial p} \frac{2l}{(p+d)^2} \right| +$$

$$\frac{1}{2\sigma_1} e^{-\frac{\frac{2l}{(p+d)^2}}{\sigma_1}} \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \left| \frac{\partial}{\partial p} \frac{2l}{(p+d)^2} \right|$$

$$= \frac{1}{\sigma_1} e^{-\frac{\frac{2l}{(p+d)^2}}{\sigma_1}} \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \frac{4l}{(p+d)^3}. \tag{25}$$

Integrating over all values of d provides the marginal distribution of p

$$\begin{aligned} f_P(p) &= \int_0^\infty \frac{1}{\sigma_1} e^{-\frac{\frac{2l}{(p+d)^2}}{\sigma_1}} \\ &* \quad \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \frac{4l}{(p+d)^3} \partial d \end{aligned} \tag{26}$$

Recall, p was crafted to be the maximum inter-update period that still meets the plausibility limit. Integrating from 0 to a particular value of p provides the probability that p exceeds the required plausibility limit, or

$$\begin{aligned} \alpha &= \int_0^p \int_0^\infty \frac{1}{\sigma_1} e^{-\frac{\frac{2l}{(p+d)^2}}{\sigma_1}} \\ &* \quad \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \frac{4l}{(p+d)^3} \partial d \partial x \end{aligned} \tag{27}$$

IV. Analysis and Applications

The models of interactions, error, and plausibility presented in Chapter III can be applied in a variety of ways to assist conducting LVC T&E events. This chapter presents the results of model validation experiments and explores some relevant applications.

4.1 Model Validation

Validation of the plausibility model presented in Chapter III was conducted using World of Warcraft player entity trace data obtained from the Game Trace Archive at Delft University of Technology [16]. The traces were sampled at 1 second intervals and consist of position information from 1302 players collected over a 24-hour period. Figure 7 shows an excerpt from an entity trace. Entity velocity and acceleration data were derived from the position traces. Figure 8 shows an example entity's trajectory through the virtual environment. Figure 9 plots the entity's acceleration as a frequency histogram normalized to a probability. Note that the majority of accelerations are very close to zero, indicating either a stationary entity or straight-line motion. This heavily skewed acceleration distribution is typical for player-controlled entities moving through a virtual environment.

A random sample of entity traces were replayed through a simulator across a range of expected parameter settings in order to ascertain feasible plausibility limits. The simulator models an interaction between two entities and employs periodic state updates and client-side dead-reckoning to minimize state inconsistency. Update latencies ranged from 0.035 s (nominal cross-country latency) to 0.5 s. Update periods ranged from 0.02 s (every frame for a simulation running at 50 Hz) to 0.3 s (the maximum specified by DIS for loosely-coupled simulations). Spatial error between

```
# time, id, x, y, z, facing
63445145235, 0, -4643.46, -1077.76, 500.58, 3.13
63445145236, 0, -4643.46, -1077.76, 500.58, 3.13
63445145237, 0, -4643.46, -1077.76, 500.58, 3.13
63445145238, 0, -4643.46, -1077.76, 500.58, 3.13
63445145239, 0, -4643.46, -1077.76, 500.58, 3.13
63445145240, 0, -4643.46, -1077.76, 500.58, 3.13
63445145241, 0, -4643.46, -1077.76, 500.58, 3.13
63445145242, 0, -4643.46, -1077.76, 500.58, 3.13
63445145243, 0, -4643.46, -1077.76, 500.58, 3.13
63445145244, 0, -4639.1, -1075.05, 500.9, 0.64
63445145245, 0, -4633.43, -1070.19, 501.42, 0.73
63445145246, 0, -4633.43, -1070.19, 501.42, 0.73
63445145247, 0, -4633.43, -1070.19, 501.42, 0.73
...
```

Figure 7. Excerpt from a World of Warcraft entity position trace.

the server and client replicas was computed at each frame, i.e., every 0.2 s. 95% non-parametric upper tolerance bounds for the spatial error were computed using Wilke's algorithm [17]. The resulting response surface is plotted in Figure 10.

For a combination of period and latency, the corresponding point on the response surface represents the $95th$ percentile of the spatial error with 95% confidence. That is, 95% of the error is expected to be less than or equal to the value of the surface at any point with 95% confidence. Note that this is equivalent to a 95% lower confidence bound on the $95th$ percentile.

Figure 11 plots the upper tolerance bounds as a contour plot. Contours are plotted from 0.1 m to 1.1 m in 0.1 m increments. This is a useful way of visualizing plausibility limits for a particular LVC simulation and can assist engineers with choosing synchronization parameters and assessing expected data quality as part of the experimental design process. For instance, an engineer designing an LVC experiment on a network with 100 ms of latency and an update period of 100 ms should expect to meet a plausibility limit of 0.07 m 95% of the time with 95% confidence. Note, how-

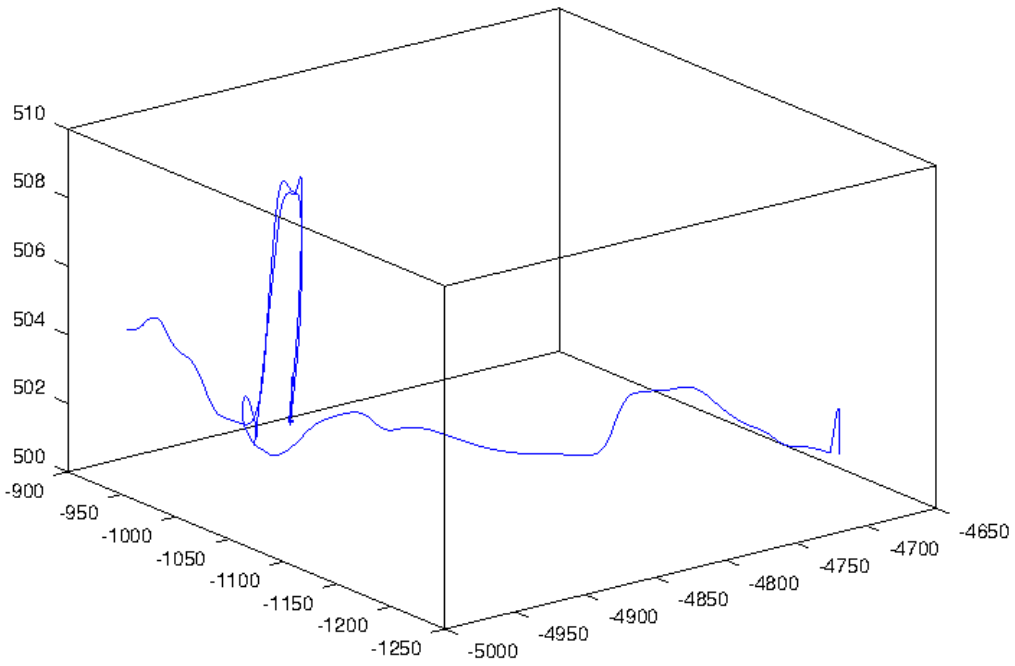


Figure 8. Position trace of an example World of Warcraft entity.

ever, that the foregoing analysis applies only if the entity motion for the experiment is similar to the World of Warcraft traces used to generate the contour plot.

Validation of the two-state stochastic plausibility model was conducted by Monte Carlo simulation of the model as a discrete-time semi-Markov process. Jump time distributions for the state transitions were generated empirically from a random sample of World of Warcraft entity traces with a plausibility limit of $L = 0.1$. Figures 12 and 13 plot the jump time probability distribution functions for two different latency/update period settings. Figure 12 assumes a latency of 0.035 and update period of 0.020 while Figure 13 assumes a latency of 0.5 and an update period of 0.3. These parameter settings are analogous to the extremes of the response surface plotted in

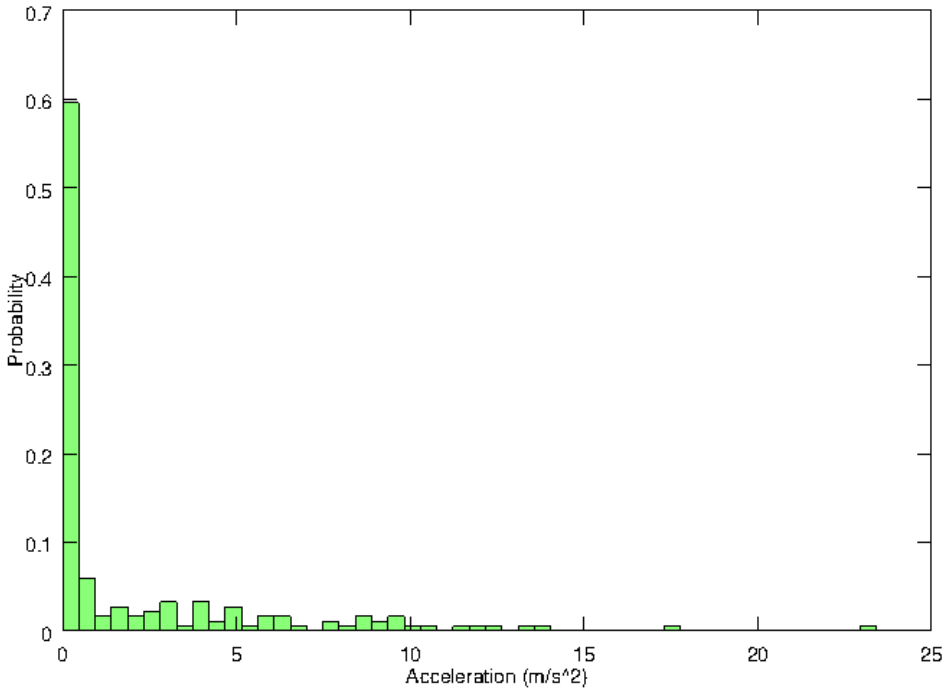


Figure 9. Acceleration histogram for the World of Warcraft entity in Figure 8.

Figure 10.

The model was run for 1000 independent trials and the percentage of time (in terms of frames) spent in the plausible state was recorded. Similarly, two validation sets of 1000 entity traces were replayed through the simulator and the percentage of time spent in the plausible state was recorded. Figure 14 plots the model predictions and validation sets as a boxplot. Note that there is some difference between the population means between the model prediction and validation sets. 2-sided t-tests return p values of 0.022 and 0.031 for the difference in means between the model and validation sets 1 and 2 respectively. However, the difference is relatively minor and variance tests show no statistical difference between the populations. Consequently,

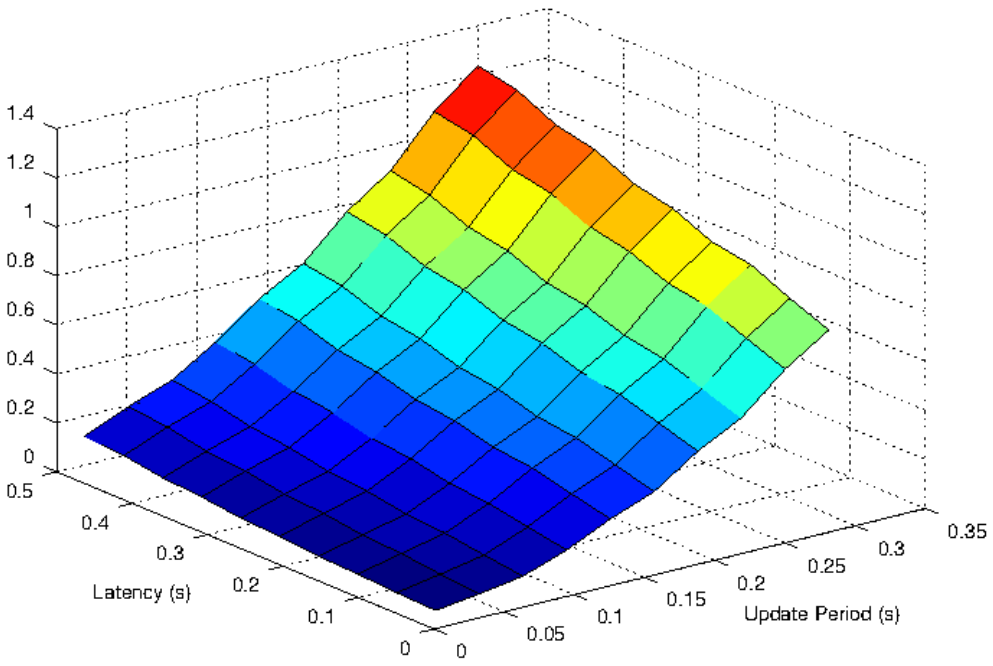


Figure 10. 95% **upper tolerance bound** ($\alpha = 0.05$) **across the range of expected update periods and latencies. Response surface derived from World of Warcraft replays.**

the model provides a reasonable prediction for the World of Warcraft data.

Validation of the optimal update period model given by Equation 18 was conducted using a simulated LVC. The server node hosts a single entity employing a random acceleration motion model. The client node maintains a replica of the server's entity and employs local dead reckoning as a consistency control algorithm. Updates consist of the current position and velocity of the entity and are sent from the server to the client at periodic intervals.

Figure 15 plots the predicted optimal update periods as a function of the plausibility limit for entities with a maximum acceleration of 1, 5, and 10 times the force of gravity. The predicted response curves assume an entity undergoing constant maxi-

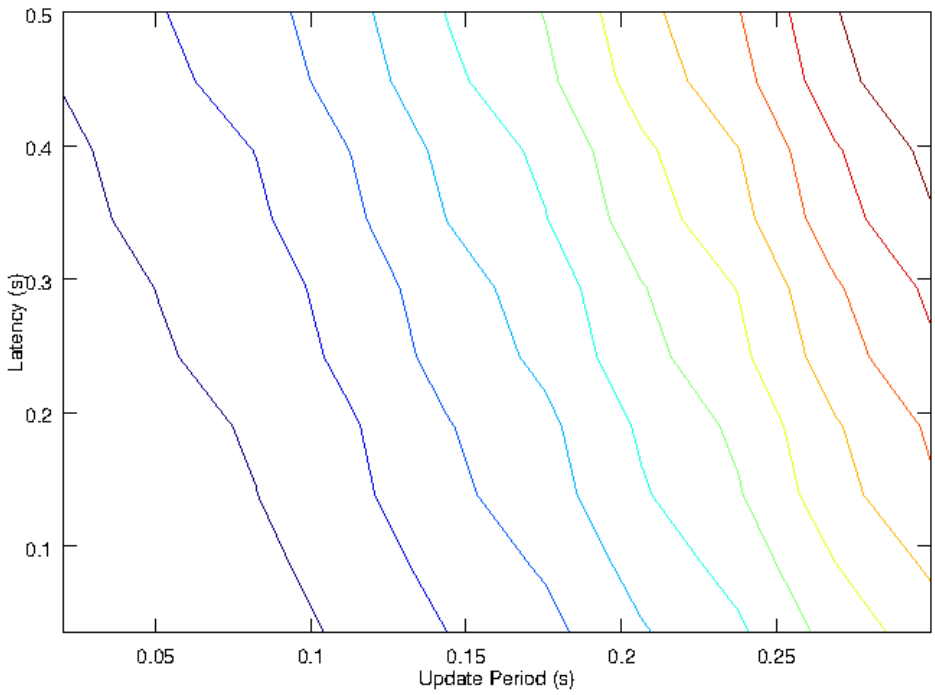


Figure 11. Contour plot of the upper tolerance surface in Figure 10. Contours are plotted for plausibility limits from $L = 0.1$ m to $L = 1.1$ m in increments of 0.1 m.

mum acceleration and therefore indicate a worst-case scenario. The blue and red lines indicate update periods of 100 ms and 300 ms respectively. These are the maximum limits recommended by the DIS standard for tightly-coupled and loosely-coupled simulations. Cross-referencing these lines with the response curves indicate the maximum obtainable plausibility limit for an entity undergoing constant maximum acceleration.

In the case of an entity whose maximum acceleration is 10 G (typical of military aircraft), the maximum plausibility limit for any interaction I is 2 m. Any interaction that requires a maximum spatial deviation less than 2 m cannot be guaranteed to provide a correct outcome. This is often manifested as a divergence in interaction outcomes for each participating entity. For example, a collision detection routine

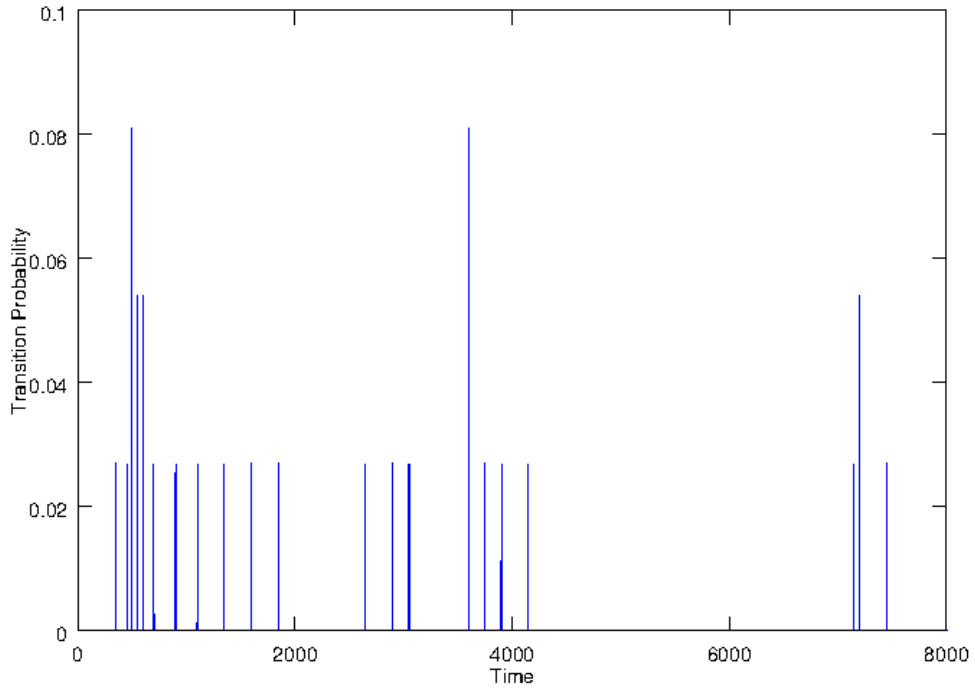


Figure 12. Jump time distribution for the plausible to implausible state transition with plausibility limit = 0.1, latency = 0.035, and update period = 0.020.

requiring accuracy of 1 m may result in one entity observing a collision while the other does not.

As noted, the model provides a worst-case prediction of the update period required to obtain a plausibility limit of L. In real DVEs, entities rarely undergo sustained maximum accelerations. In fact, entity acceleration is practically zero the vast majority of the time. If the requirement for meeting the plausibility limit is relaxed in a statistical fashion, then the required update period can be quite a bit higher than that predicted by the optimal model. That is, if the spatial error is allowed to exceed the limit some fraction of the time, then tolerance limits can be computed at a specified level of confidence. The guarantee provided by Equation 18 then becomes something

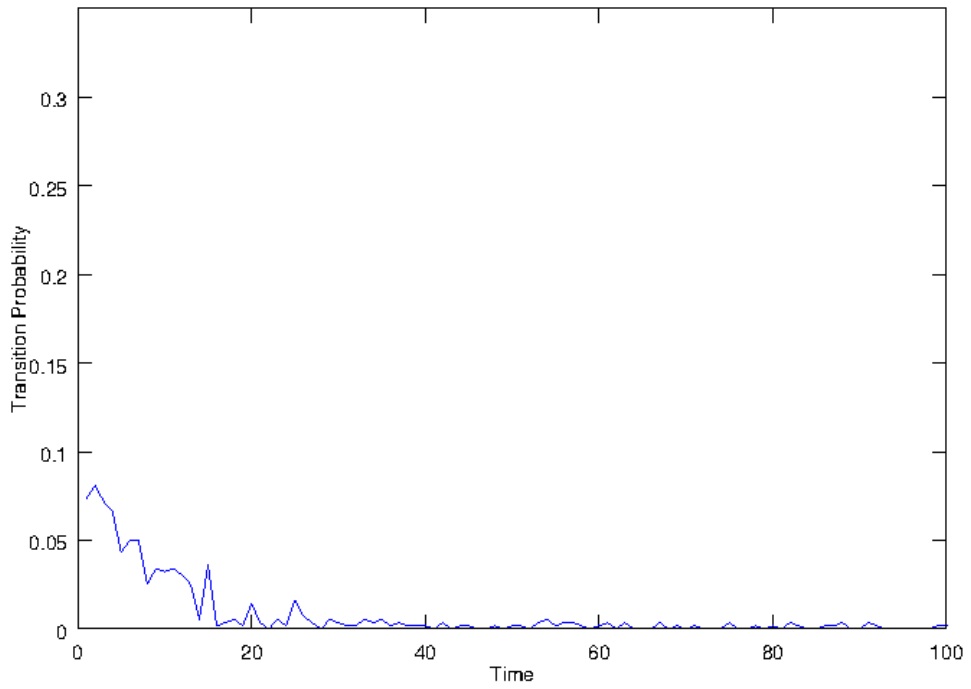


Figure 13. Jump time distribution for the plausible to implausible state transition with plausibility limit = 0.1, latency = 0.5, and update period = 0.3.

similar to "with confidence $1-\alpha$, the spatial error will be within the plausibility limit L 95% of the time." More formally, we have the following optimization problem:

$$\min \sqrt{\frac{2 * L}{a_{max}}} - d - L \tag{28}$$

Figure 16 plots the deviation between the 95% upper tolerance bound ($1 - \alpha = 0.05$) for an entity moving according to a normally distributed acceleration profile (a $N(0, 1)$, $a_{\max} = 9.8$) and a plausibility limit $L = 1.0$ with an update propagation delay of 100 ms. The predicted update period according to Equation 18 is 0.3518 s. On the other hand, relaxing the consistency constraints and solving the minimization

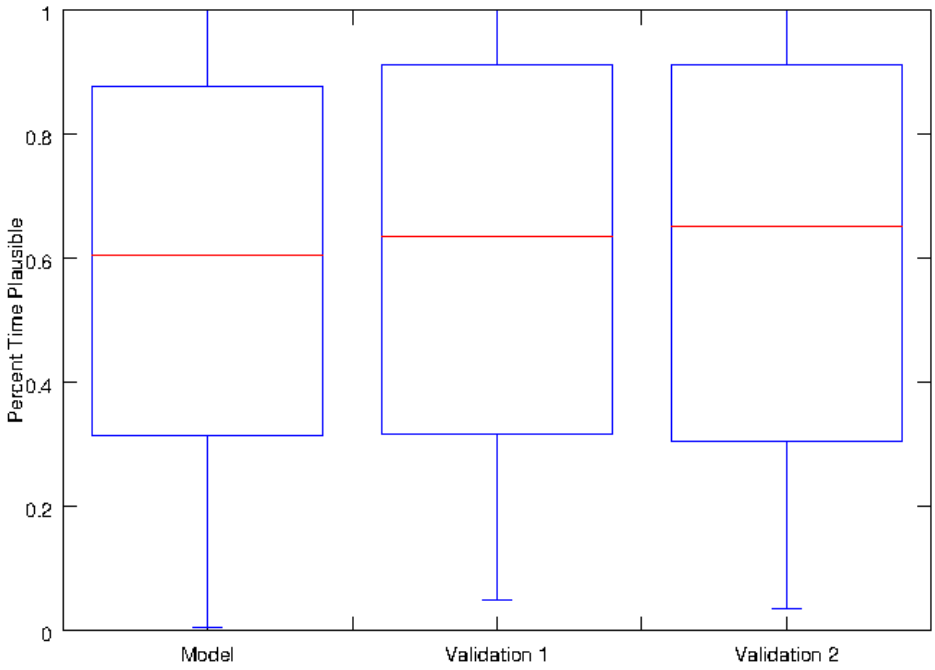


Figure 14. Boxplot showing percentage of time spent in the plausible state for model predictions and validation sets.

problem given in Equation 28 yields an update period of 1.75 s. This larger period ensures that 95% of the spatial error over the course of the simulation will be less than or equal to $L = 1.0$ with 95% confidence.

4.2 Performance Evaluation

Finally, a performance comparison between plausibility-based periodic state updates and dead-reckoning updates using the World of Warcraft entity trace data was conducted. For this experiment, 100 randomly sampled entity traces were replayed through a simulator for a range of plausibility limits ($L = 0$ to $L = 5$ spatial units). Update periods were computed according Equation 18 for the plausibility-based up-

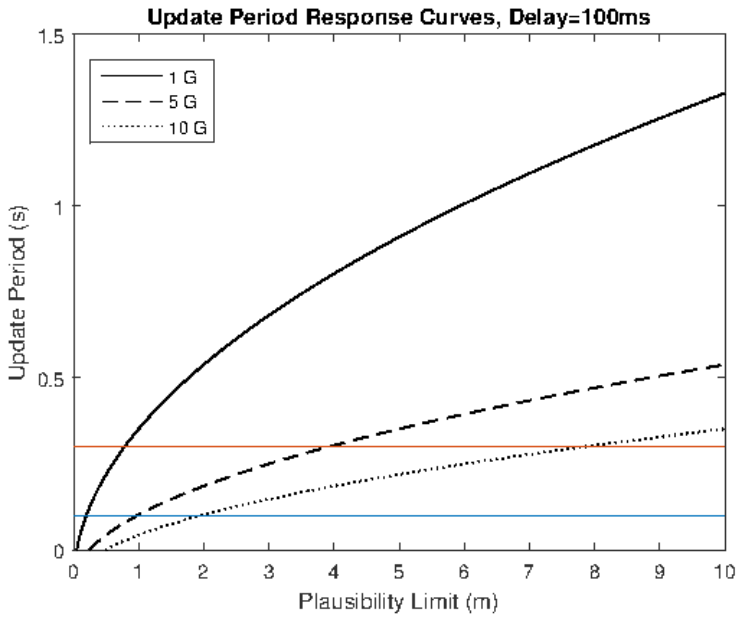


Figure 15. Update period as a function of plausibility limit with a delay of 100 ms and maximum acclerations of 1, 5, and 10 G. Blue and red lines indicate 100 ms and 300 ms update periods respectively.

date policy. The dead-reckoning error threshold was set equal to the plausibility limit. Performance was measured in terms of the percentage of frames within the plausibility limit. Figure 17 plots the results of the comparison for a nominal latency of 100 ms. Figure 18 plots the results for a latency of 300 ms. These latencies were chosen to correspond to worst-case latencies for tightly-coupled and loosely-coupled simulations (respectively) as specified in the DIS standard. On average, plausibility-based updates result in deviations less than the plausibility limit approximately 95% of the time (97.93% for 100 ms latency, 94.46% for 300 ms latency). This is expected since the update period was selected to provide a 95% upper tolerance bound. Conversely, dead-reckoning results in slightly fewer plausible frames, with 96.48% (100 ms latency) and 89.64% (300 ms latency) falling within the plausibility limit.

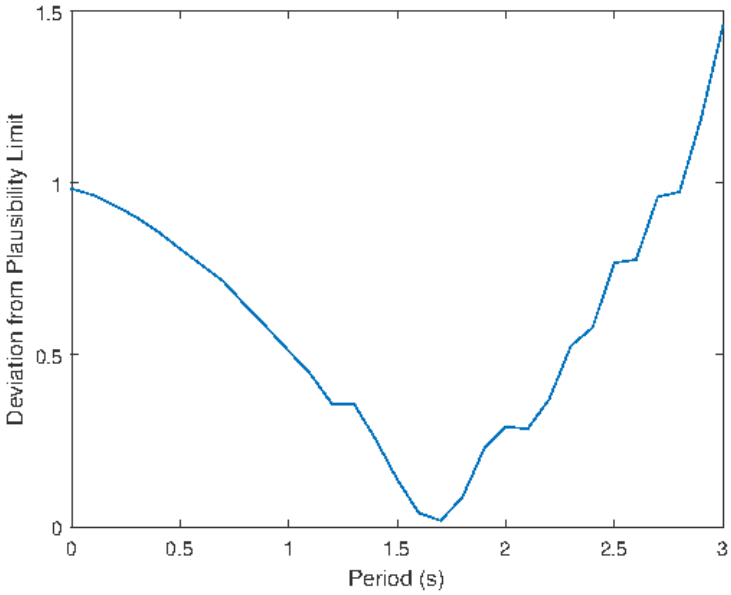


Figure 16. Deviation between the 95% upper tolerance bound $(1 - \alpha = 0.05)$ and plausibility limit $L = 1.0$ for an entity with normally distributed acceleration $N(0, 1)$.

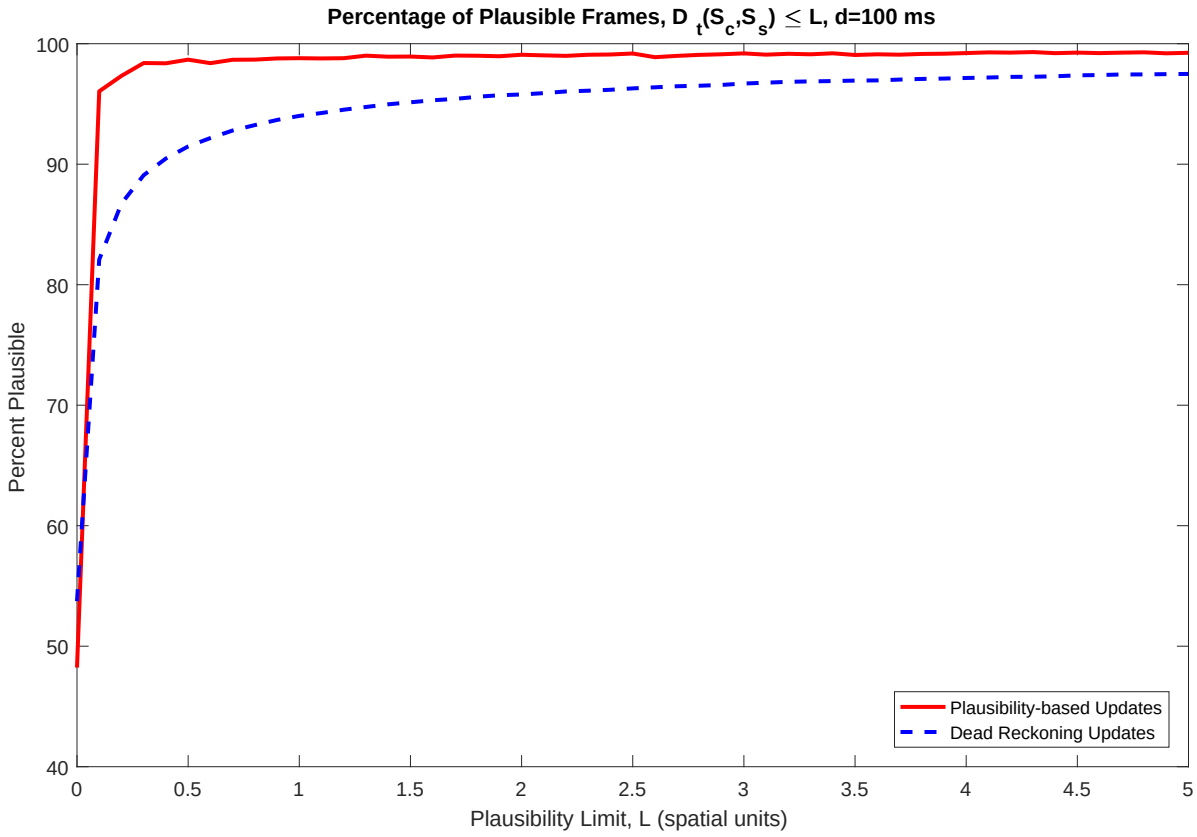


Figure 17. Performance comparison of plausibility-based periodic and dead-reckoning update policies across a range of plausibility limits ($L = 0$ to $L = 5$) with a nominal latency of 100 ms. Performance is expressed as a percentage of total frames within the plausibility limit, i.e., $D_t(S_c, S_s) \leq L$.

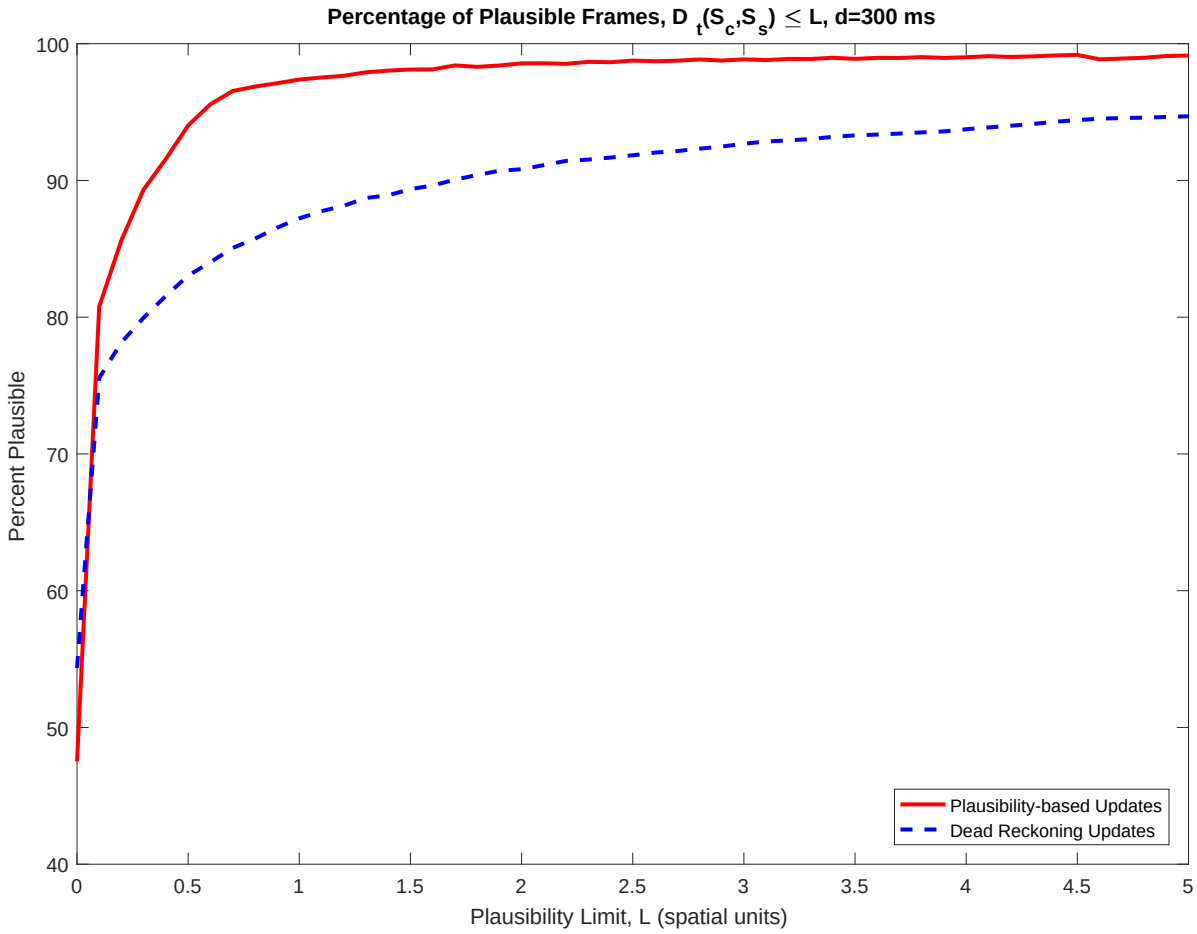


Figure 18. Performance comparison of plausibility-based periodic and dead-reckoning update policies across a range of plausibility limits ($L = 0$ to $L = 5$) with a nominal latency of 300 ms. Performance is expressed as a percentage of total frames within the plausibility limit, i.e., $D_t(S_c, S_s) \leq L$.

V. Conclusion

Assessing data quality is a key challenge for successful application of LVC simulation in T&E environments. A core property of geographically distributed systems such as LVC simulations is the inability to guarantee consistent state data at all times and all nodes. The CAP theorem implies that the best consistency guarantee available to systems that must remain available and tolerant to network partition is eventual consistency with "read-my-writes" session guarantee. Because of this fundamental limitation, it cannot be assumed that all entities in a LVC simulation see and experience the same sequence of events. Crucially, this can lead to nonsensical and implausible results – a sure problem for simulations designed to support engineering test and evaluation. The problem is akin to trying to make accurate measurements with a device without graduation and of suspect precision.

This work has presented a model of plausibility for LVC simulations. Here, plausibility is defined as an interaction result that all participating entities are willing to accept and that makes sense given the state of the virtual environment as seen by each participant. The key observation is that absolute state consistency is not required for plausible outcomes. Rather, the state need only be close enough for all parties to agree on the results. This observation gives rise to the notion of an *interaction*; that is, a state dependence between two entities in the LVC environment and an associated *plausibility limit*. So long as the inconsistency associated with the entities' state database replicas is bounded by the plausibility limit, the outcome of the interaction is assumed to be plausible.

Since dead-reckoning is commonly used as a consistency maintenance mechanism with the intent of reducing divergence between entities and replicas, an error model was derived combining the residual error from dead-reckoning with the notion of plausibility limits. This model can be used by LVC simulation designers to choose

state update periods based on maximum rates of change for entity state variables and end-to-end propagation delays with the intent of staying within the plausibility limit for a particular interaction.

Additionally, a simple stochastic model of plausibility was developed based on Monte Carlo simulation of a semi-Markov process that can provide statistical estimates of how likely the LVC simulation is to remain within an interaction's plausibility limit. This allows the simulation engineer to assess fitness for purpose early in the design process. If the LVC simulation is unlikely to meet precision requirements (e.g., due to high latency or rapid rates of state change), then some other means of conducting the experiment can be found before large amounts of time or money are spent.

Finally, it seems fitting to offer some broad observations on LVC simulation for T&E based on this work. Firstly, LVC simulation offers on opportunities that may be unobtainable via traditional T&E venues. For instance, it will be difficult (if not impossible) to present threat environments to current generation fighter aircraft (e.g., the F-35) that are saturated enough to stress the aircraft on a live test range. LVC simulation can provide any threat density desired, and can provide threats that are not physically available. This leads to a second benefit of LVC simulation – the ability to employ assets, entities, and systems that might otherwise be unavailable for live test – B-2 bombers for instance. And, of course, there are real cost savings to be had through the use of LVC simulation.

However, some caution is warranted. The challenges associated with assessing the quality of data collected from an LVC simulation are significant. A momentary increase in network latency, due perhaps to a flash crowd or viral video, can result in state errors large enough to skew test results. Post facto correlation of such events is difficult at best. The potential for confounding factors is high unless dedicated

networks with well-known quality of service are available. Even then, a poor choice of synchronization algorithm or parameter can have similarly damaging effects on data quality. LVC simulations have much potential to fundamentally change test and evaluation, but their limitations and confounding factors must be well understood and explicitly accounted for.

Bibliography

1. Abadi, Daniel J. "Consistency tradeoffs in modern distributed database system design". *Computer-IEEE Computer Magazine*, 45(2):37, 2012.

2. Aggarwal, Sudhir, Hemant Banavar, Amit Khandelwal, Sarit Mukherjee, and Sampath Rangarajan. "Accuracy in dead-reckoning based distributed multi-player games". *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, 161–165. ACM, 2004.

3. Bailis, Peter and Ali Ghodsi. "Eventual consistency today: limitations, extensions, and beyond". *Commun. ACM*, 56(5):55–63, May 2013. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/2447976.2447992`.

4. Bailis, Peter, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. "Probabilistically bounded staleness for practical partial quorums". *Proc. VLDB Endow.*, 5(8):776–787, April 2012. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=2212351.2212359`.

5. Baker, Jason, Chris Bond, James C Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. "Megastore: Providing Scalable, Highly Available Storage for Interactive Services." *CIDR*, volume 11, 223–234. 2011.

6. Brewer, Eric A. "Towards robust distributed systems". *PODC*, volume 7. 2000.

7. Cai, Wentong, Francis Lee, and Lian Chen. "An auto-adaptive dead reckoning algorithm for distributed interactive simulation". *Proceedings of the thirteenth workshop on Parallel and distributed simulation*, 82–89. IEEE Computer Society, 1999.

8. Chen, Kuan-Ta, Polly Huang, and Chin-Laung Lei. "Effect of network quality on player departure behavior in online games". *Parallel and Distributed Systems, IEEE Transactions on*, 20(5):593–606, 2009.

9. Dahmann, Judith S, Richard M Fujimoto, and Richard M Weatherly. "The department of defense high level architecture". *Proceedings of the 29th conference on Winter simulation*, 142–149. IEEE Computer Society, 1997.

10. DeCandia, Giuseppe, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. "Dynamo: amazon's highly available key-value store". *ACM SIGOPS Operating Systems Review*, volume 41, 205–220. ACM, 2007.

11. Delaney, Declan, Tomás Ward, and Séamus McLoone. "On reducing entity state update packets in distributed interactive simulations using a hybrid model". IASTED, 2003.

12. DIS Steering Committee. "IEEE Standard for Distributed Interactive Simulation-Application Protocols". *IEEE Standard*, 1278, 1998.

13. Feinberg, A. "Project Voldemort: Reliable distributed storage". *Proceedings of the 10th IEEE International Conference on Data Engineering*. 2011.

14. Ferguson, Bernard and Neyer Torrico. "Distributed – The Next Step in T&E". *The ITEA Journal*, 35:132–140, 2014.

15. Gilbert, Seth and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". *ACM SIGACT News*, 33(2):51–59, 2002.

16. Guo, Yong and Alexandru Iosup. "The Game Trace Archive". *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, NetGames '12, 4:1–4:6. IEEE Press, Piscataway, NJ, USA, 2012. ISBN 978-1-4673-4578-1. URL `http://dl.acm.org/citation.cfm?id=2501560.2501566`.

17. Hahn, Gerald J and William Q Meeker. *Statistical intervals: a guide for practitioners*. John Wiley & Sons, 1991.

18. Hanawa, Dai and Tatsuhiro Yonekura. "On the error modeling of dead reckoned data in a distributed virtual environment". *Cyberworlds, 2005. International Conference on*, 8–pp. IEEE, 2005.

19. Hastorun, Deniz, Madan Jampani, Gunavardhan Kakulapati, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. "Dynamo: Amazons highly available key-value store". *In Proc. SOSP*. Citeseer, 2007.

20. Hodson, Douglas D and Rusty O Baldwin. "Characterizing, Measuring, and Validating the Temporal Consistency of Live–Virtual–Constructive Environments". *Simulation*, 85(10):671–682, 2009.

21. Hodson, Douglas D and Rusty O Baldwin. "Performance analysis of live-virtual-constructive and distributed virtual simulations: defining requirements in terms of temporal consistency". 2009.

22. Hodson, Douglas D, Alex J Gutman, Bruce Esken, and Raymond R Hill. "Quantifying Radar Measurement Errors in a Live-Virtual-Constructive Environment to Determine System Viability: A Case Study". *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 1548512913503740, 2013.

23. Hodson, Douglas D and Raymond R Hill. "The Art and Science of Live, Virtual and Constructive Simulation for Test and Analysis". *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 1548512913506620, 2013.

24. Itzel, Laura, Verena Tuttlies, Gregor Schiele, and Christian Becker. "Consistency management for interactive peer-to-peer-based systems". *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 1. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

25. Kopetz, Hermann. *Real-time systems: design principles for distributed embedded applications*. Springer, 2011.

26. Kraska, Tim, Martin Hentschel, Gustavo Alonso, and Donald Kossmann. "Consistency Rationing in the Cloud: Pay only when it matters". *Proceedings of the VLDB Endowment*, 2(1):253–264, 2009.

27. Krishnamoorthy, Kalimuthu and Thomas Mathew. *Statistical tolerance regions: theory, applications, and computation*, volume 744. John Wiley & Sons, 2009.

28. Lakshman, Avinash and Prashant Malik. "Cassandra: structured storage system on a p2p network". *Proceedings of the 28th ACM symposium on Principles of distributed computing*, 5–5. ACM, 2009.

29. Lamport, Leslie. "Time, Clocks, and the Ordering of Events in a Distributed System". *Commun. ACM*, 21(7):558–565, July 1978. ISSN 0001-0782. URL `http://doi.acm.org/10.1145/359545.359563`.

30. Li, Yusen and Wentong Cai. "Consistency aware dead reckoning threshold tuning with server assistance in client-server-based dves". *Computer and Informa-*

tion Technology (CIT), 2010 IEEE 10th International Conference on, 2925–2932. IEEE, 2010.

31. Li, Yusen and Wentong Cai. "Determining Optimal Update Period for Minimizing Inconsistency in Multi-server Distributed Virtual Environments". *Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '11, 126–133. IEEE Computer Society, Washington, DC, USA, 2011. ISBN 978-0-7695-4553-0. URL `http://dx.doi.org/10.1109/DS-RT.2011.10`.

32. Li, Z, W Cai, X Tang, and S Zhou. "Loss-aware DR-based update scheduling for improving consistency in DVEs". *Journal of Simulation*, 6(3):164–178, 2012.

33. Li, Zengxiang, Wentong Cai, Xueyan Tang, and Suiping Zhou. "Dead reckoning-based update scheduling against message loss for improving consistency in DVEs". *Principles of Advanced and Distributed Simulation (PADS), 2011 IEEE Workshop on*, 1–9. IEEE, 2011.

34. Li, Zengxiang, Xueyan Tang, Wentong Cai, and Xiaorong Li. "Compensatory dead-reckoning-based update scheduling for distributed virtual environments". *SIMULATION*, 2013.

35. Lui, John C. S. "Constructing communication subgraphs and deriving an optimal synchronization interval for distributed virtual environment systems". *Knowledge and Data Engineering, IEEE Transactions on*, 13(5):778–792, 2001.

36. Mauve, Martin. "How to keep a dead man from shooting". *Interactive Distributed Multimedia Systems and Telecommunication Services*, 199–204. Springer, 2000.

37. Morse, Katherine L et al. *Interest management in large-scale distributed simulations*. Information and Computer Science, University of California, Irvine, 1996.

38. Pace, Dale K. "Comprehensive consideration of uncertainty in simulation use". *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, 2012. URL `http://dms.sagepub.com/content/early/2012/09/03/1548512912455471.abstract`.

39. Parker, Eric Paul, Nadine Elizabeth Miner, Brian Peter Van Leeuwen, and James Brian Rigdon. "Testing unmanned autonomous system communications in a Live/Virtual/Constructive environment". *International Test and Evaluation Association Journal (ITEA)*, 30:513–522, 2009.

40. Powell, Edward T and J Russell Noseworthy. "The test and training enabling architecture (TENA)". *Engineering Principles of Combat Modeling and Distributed Simulation*, 449, 2012.

41. Rahman, Muntasir Raihan, Wojciech Golab, Alvin AuYoung, Kimberly Keeton, and Jay J Wylie. "Toward a principled framework for benchmarking consistency". *Proceedings of the Eighth USENIX conference on Hot Topics in System Dependability*, 8–8. USENIX Association, 2012.

42. Roberts, Dave, Rob Aspin, Damien Marshall, Seamus Mcloone, Declan Delaney, and Tomas Ward. "Bounding inconsistency using a novel threshold metric for dead reckoning update packet generation". *Simulation*, 84(5):239–256, 2008.

43. Roberts, Dave, Damien Marshall, S MacLoone, Declan Delaney, Tomas Ward, and R Aspit. "Exploring the use of local consistency measures as thresholds for dead reckoning update packet generation". *Distributed Simulation and Real-*

Time Applications, 2005. DS-RT 2005 Proceedings. Ninth IEEE International Symposium on, 195–202. IEEE, 2005.

44. Saldana, Jose and Mirko Suznjevic. "QoE and Latency Issues in Networked Games". *Handbook of Digital Games and Entertainment Technologies*, 509, 2017.

45. Singhal, Sandeep Kishan. *Effective remote modeling in large-scale distributed simulation and visualization environments*. Ph.D. thesis, Stanford University, 1996.

46. Steed, Anthony and Manuel Fradinho Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments*. Elsevier, 2009.

47. Tanenbaum, Andrew and Maarten Van Steen. *Distributed systems*. Pearson Prentice Hall, 2007.

48. Tang, Xueyan and Suiping Zhou. "Update scheduling for improving consistency in distributed virtual environments". *Parallel and Distributed Systems, IEEE Transactions on*, 21(6):765–777, 2010.

49. Terry, Doug. "Replicated data consistency explained through baseball". *Communications of the ACM*, 56(12):82–89, 2013.

50. Terry, Douglas B, Alan J Demers, Karin Petersen, Mike J Spreitzer, Marvin M Theimer, and Brent B Welch. "Session guarantees for weakly consistent replicated data". *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*, 140–149. IEEE, 1994.

51. Van Leeuwen, Brian, Vincent Urias, John Eldridge, Charles Villamarin, and Ron Olsberg. "Performing cyber security analysis using a live, virtual, and constructive (LVC) testbed". *Military Communications Conference, 2010-MILCOM 2010*, 1806–1811. IEEE, 2010.

52. Wada, Hiroshi, Alan Fekete, Liang Zhao, Kevin Lee, and Anna Liu. "Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective." *CIDR*, volume 11, 134–143. 2011.

53. Yu, Haifeng and Amin Vahdat. "Design and evaluation of a conit-based continuous consistency model for replicated services". *ACM Transactions on Computer Systems (TOCS)*, 20(3):239–282, 2002.

54. Yu, Yang, Zhu Li, Larry Shi, Yi-Chiun Chen, and Hua Xu. "Network-aware state update for large scale mobile games". *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, 563–568. IEEE, 2007.

55. Zhou, Suiping, Wentong Cai, Bu-Sung Lee, and Stephen J Turner. "Time-space consistency in large-scale distributed virtual environments". *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(1):31–47, 2004.

Appendix A. Publications

Several portions of this dissertation have been published in peer-reviewed journals and conferences. Citations and text for these articles are included below for reference and convenience.

- Multi-Objective Optimization of Dead-Reckoning Error Thresholds for Virtual Environments. Jeremy R. Millar, Douglas D. Hodson, Gary B. Lamont, and Gilbert L. Peterson. 2014 International Conference on Collaboration Technologies and Systems (CTS). May, 2014.

- Data Quality Challenges in Distributed Live-Virtual-Constructive Test Environments. Jeremy R. Millar, Douglas D. Hodson, Gilbert L. Peterson, and Darryl K. Ahner. Journal of Data and Information Quality. April, 2016.

- Consistency and Fairness in Real-Time Distributed Virtual Environments: Paradigms and Relationships. Journal of Simulation. 2016.

- Deriving LVC State Synchronization Parameters from Interaction Requirements. Jeremy R. Millar, Doulgas D. Hodson, and Richard Seymour. 20th ACM/IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2016). September, 2016.

- Sources of Unresolvable Uncertainty in Weakly Predictive Distributed Virtual Environments. Jeremy R. Millar, Jason A. Blake, Douglas D. Hodson, J.O. Miller, and Raymond R. Hill. Proceedings of the 2016 Winter Simulation Conference. December, 2016.

- Optimizing Update Scheduling Parameters for Distributed Virtual Environments Supporting Operational Test. Jeremy R. Millar, Douglas D. Hodson,

Gilbert L. Peterson, and Darryl K. Ahner. Concurrency and Computation: Practice and Experience. Accepted, to appear.

Multi-Objective Optimization of Dead-Reckoning Error Thresholds for Virtual Environments

Jeremy R. Millar, Douglas D. Hodson, Gary B. Lamont, Gilbert L. Peterson
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

***Abstract*—Design trade-offs between state consistency and system response time are commonplace in virtual environments. Systems typically rely on predictive consistency algorithms such as dead-reckoning to control consistency and response time. Dead-reckoning error threshold selection determines the consistency/response time trade-off. We extend this trade-off space to explicitly account for the concept of system fairness. We derive a multi-objective optimization problem and apply multi-objective evolutionary algorithms to solve for Pareto optimal error thresholds.**

***Keywords*—Virtual simulation environments, dead reckoning, multi-objective optimization**

I. Introduction

In order to maintain the illusion of a shared environment users must view the same information at the same time [1]–[4]. However, waiting for all nodes to acknowledge receipt of a state update has a negative effect on system responsiveness, thus breaking the sense of immersion and reducing the quality of user experience. Indeed, research has shown that network latencies of 60 ms induce enough response lag to detract from the play experience in some networked games, while latencies above 100 ms result in game abandonment [5], [6]. Similarly, high levels of state inconsistency result in user dissatisfaction due to nonsensical results, e.g., dead men shooting [7].

Consistency can be made arbitrarily good at the expense of system responsiveness by executing the system in lockstep across all participating nodes. Responsiveness, on the other hand, is limited by system architecture and network topology [8]. While perfectly responsive systems could in principle achieve perfect consistency, physical constraints such as the speed of light ensure even the most responsive virtual environments will exhibit some inconsistency [9]. Moreover, as responsiveness improves, the scalability of the system is limited as the system infrastructure becomes overloaded with state updates. A primary goal of virtual environment systems design is to maximize system responsiveness while simultaneously maximizing consistency.

An additional design objective, particularly for network games and interactive simulations, is the notion of system fairness. Fairness measures the disparity between nodes with respect to consistency and responsiveness [6]. The more similar the nodes are with respect to these measures, the more fair the virtual environment. Fair environments ensure that all participants experience similar levels of consistency and response and that no user gains an advantage due to system architecture.

Many virtual environments use predictive consistency mechanisms to improve response time to an input while still achieving acceptable consistency. Dead-reckoning [3], [4], [10] is widely used due to its simplicity and performance. Dead reckoning allows for some amount of inconsistency at remote nodes to ensure local response times remain within acceptable perceptual thresholds. This is achieved by allowing remote nodes to predict the state of the local node between state updates. The key parameter controlling response time and consistency is the state update frequency, which is itself determined by an error threshold. This threshold is the system designer's primary means of tuning consistency, responsiveness, and fairness.

In this paper, we characterize the choice of error threshold as a multi-objective optimization problem. Our approach differs from other work in this area by explicitly including fairness as an objective. We solve the resulting tri-objective problem (i.e., consistency, responsiveness, and fairness) using evolutionary algorithms and a simulation of the virtual environment. This approach allows us to choose dead-reckoning error thresholds that are Pareto optimal for a given system architecture. Finally, we use a simulation to explore the consistency, responsiveness, and fairness trade-offs for a simple virtual environment.

The remainder of this paper is structured as follows: Sections II and III briefly discuss the concepts of dead reckoning and fairness, respectively. Section IV presents a multi-objective model of the error threshold problem. Section V outlines a system architecture for solving the multi-objective error threshold problem. Section VI describes experiments undertaken to validate the architecture and compare performance of various optimization algorithms. Section VII analyzes the results of those experiments. Section VIII outlines related work and Section IX concludes.

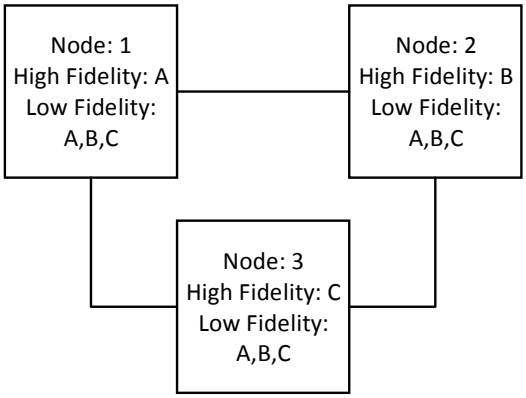


Figure 1. A virtual environment consisting of three nodes and three entities. Each node maintains an authoritative high fidelity model for its local entity and low fidelity models for all entities in the system.

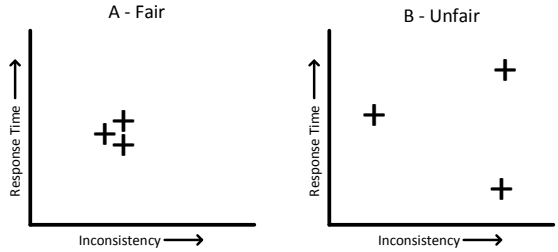


Figure 2. Fairness plots for two three node virtual environments. Each + symbol represents a node's location in two-dimensional fairness space. The nodes in system A are tightly clustered with similar consistency and responsiveness values. Therefore, system A is fair. Conversely, system B is unfair since the nodes are widely dispersed in fairness space.

II. Dead Reckoning

A virtual environment is a distributed software system supporting multiple users interacting in real-time that provides shared senses of space, presence, and time [2]. The IEEE published Standard 1278, Distributed Interactive Simulation (DIS) [10], to provide a common protocol and messaging standard for communicating between nodes in a virtual environment. While there are other standards available (e.g., HLA [11] or TENA [12]), DIS provides a de facto standard for defense oriented virtual environments (i.e., networked simulators) and its consistency maintenance mechanisms are widely used in networked games [4]. Consequently, we restrict our attention to dead-reckoning algorithms as defined by the DIS standard.

The DIS standard defines a predictive consistency maintenance protocol called *dead-reckoning*. Under dead-reckoning, each node maintains a set of low-fidelity models for each remote entity in the system in addition to the high-fidelity models for its hosted entities. Figure 1 depicts a virtual environment consisting of three nodes and three entities. Each node provides an authoritative, high-fidelity model for one entity. Additionally, each node maintains a low-fidelity model of all other entities in the system. Crucially, each node also maintains a low-fidelity model of its own local entity.

The low-fidelity models allow a node to update entity positions between state updates using dead-reckoning algorithms. Low-fidelity models typically operate using simplified dynamics such as first order kinematics. Note that all nodes execute the same dead-reckoning model. State updates are sent by a node whenever the divergence (i.e., difference) between the position of the high-fidelity and low-fidelity models of its hosted entities exceeds a pre-determined threshold. This threshold is the key parameter controlling the dead-reckoning algorithm.

Choosing an appropriate error threshold is a system dependent design decision. Generally speaking, lower thresholds yield better consistency. However, improved consistency comes at the cost of increased network traffic. Depending on network characteristics such as available bandwidth, it is possible to overwhelm the network and increase system response time (that is, the time it takes for all nodes to see an update). Additionally, as network load increases, consistency can actually decrease as well [13].

III. Fairness

Consistency and response time are local properties, that is, they are measured pair-wise. Thus the consistency measured between nodes 1 and 2 with respect to entity A in Figure 1 might well be different than that measured between nodes 2 and 3. Global properties are also of interest, particularly the notion of fairness [6], [14]. A system is fair if no user has an advantage over others due to consistency or response time.

Fairness is measured by projecting each node participating in a virtual environment into a two-dimensional fairness space with consistency as one dimension and system response time as the other. A cluster cohesion measure such as within-class scatter is computed for all nodes. Low scatter indicates that the nodes are tightly clustered in fairness space. Thus, each node has similar consistency levels and response times and no participant experiences a significant advantage or handicap. On the other hand, a high scatter value indicates that nodes have dissimilar consistency values and response times. This affords some participants advantages in terms of state consistency or response while handicapping others.

Figure 2 illustrates these ideas for two three-node virtual environments labeled system A and system B. Note that the horizontal axis measures *inconsistency* so that consistency degrades as one moves away from the origin. The vertical axis measures system response time, i.e., the time required for a state update from one node to reach all other nodes. For both dimensions lower values (closer to the origin) are more desirable. Each '+' symbol indicates an individual node's position in fairness space based on average consistency level

and response time.

For system A, the nodes are clustered fairly tightly, indicating a fair system. Each participating node has a similar consistency level and response time. Thus no participant has a distinct advantage in terms of better information about the environment or more rapid environmental response. Conversely, the nodes in system B are widely dispersed in fairness space. This indicates an unfair system. One node has a distinct advantage in terms of data consistency, one has an advantage in response time, and one is severely handicapped in both dimensions.

A system can be fair while exhibiting poor performance with respect to data consistency or response time. Similarly, a system with generally good performance can be unfair so long as at least one node has sufficiently different performance characteristics. Consequently, it is incumbent upon system designers to consider fairness in addition to the more traditional trade-offs between consistency and responsiveness.

IV. Multi-Objective Model

In order to optimize the dead-reckoning error threshold, we need to define and compute the following quantities:

1) average inconsistency,
2) average response time,
3) and fairness.

A. Computing Inconsistency

The virtual environment community has settled on two major inconsistency measures:

1) spatial inconsistency (variously termed spatial error, export error, etc),
2) and time-space inconsistency [15],

Spatial inconsistency is simply the difference between the local, dead-reckoning estimate of an entity's position and its true, high-fidelity position. Time-space inconsistency is the spatial inconsistency integrated over a time period to account for the fact that even small errors can be meaningful if they last long enough. Of the two, spatial inconsistency is the more common, largely because it is simple to compute. Additionally, choosing thresholds for time-space inconsistency can be non-intuitive since the value no longer corresponds to a simple error. For these reasons, we consider spatial inconsistency as our measure of interest for this research. However, the optimization techniques employed here are applicable regardless of the specific inconsistency measure. Indeed, they may well make time-space inconsistency more attractive by eliminating manual input of the threshold value.

We compute the average spatial inconsistency as follows: let $P_i(t)$ be the true position of entity i at time t. Let $P_i^j(t)$ be the position of entity i as represented by node j at time t according to its dead-reckoning model. Then the average (pairwise) inconsistency with respect to entity i at node j is given by

$$\frac{1}{T}\sum_{t=1}^{T}|P_i(t) - P_i^j(t)| \tag{1}$$

Note that this assumes a one-to-one mapping between enitities and hosts; i.e., node i hosts the high-fidelity model for entity i (and no others). Averaging Equation 1 over all entities for a particular node j gives the average spatial inconsistency experienced by node j, i.e.,

$$\frac{1}{N}\sum_{i=1,i\neq j}^{N}\frac{1}{T}\sum_{t=1}^{T}|P_i(t) - P_i^j(t)| \tag{2}$$

Computing Equation 2 for all nodes and averaging provides the average global system inconsistency associated with remote entity positions, i.e.,

$$\frac{1}{N^2T}\sum_{i=1}^{N}\sum_{i=1,i\neq j}^{N}\sum_{t=1}^{T}|P_i(t) - P_i^j(t)| \tag{3}$$

We desire to minimize this inconsistency measure.

B. Computing Response Time

Local response time is associated with the time it takes to process user inputs. We consider the response time associated with propagating state updates from a given node to all other nodes in the system. In order to account for queuing effects in the implemented software system itself, as well as all network-induced latencies, this value should be measured in an end-to-end fashion. That is, the clock begins when the sending application executes the send operation and not when the operating system and network hardware actually place the bits on the wire. Similarly, it ends when the receiving application (not host or operating system) has received the data.

Response time can be calculated as follows: let t_{ij} be the amount of time required to send an update from node i to node j. The the response time is given by

$$\max_j t_{ij}, j = 1 \ldots N, j \neq i \tag{4}$$

where N is the total number of nodes in the system. Note that this value may vary with time since it depends on environmental factors such as network load. For simplicity, we assume this value is constant.

Averaging Equation 4 over all nodes provides a measure of system response time, i.e.,

$$\frac{1}{N}\sum_{i=1}^{N}\max_{j} t_{ij}, j = 1 \ldots N, i \neq j \quad (5)$$

We seek to minimize this response time.

C. Computing Fairness

Equations 2 and 4 provide a means of locating each node in a two-dimensional fairness space. System fairness is computed as the cohesion of the resulting data cluster. Let the vector f_i be the location in fairness space of node i. Then the system fairness is given by

$$\sum_{i=1}^{N}(f_i - c)^2 \quad (6)$$

where c is the centroid of the N fairness locations f_i. Minimizing this value corresponds to a tighter grouping in fairness space.

D. Multi-Objective Error Threshold Problem

We are now in a position to define selection of the dead-reckoning error threshold as a multi-objective optimization problem. Let x be the spatial error threshold. Let $\vec{f} = (f_1 f_2 f_3)$, where f_1 is given by Equation 3, f_2 is given by Equation 5, and f_3 is given by Equation 6. Then we wish to find

$$\min_{x} f(\vec{x}) \text{ s.t. } BW - BW_{max} \leq 0 \quad (7)$$

where BW is the system bandwidth requirement based on the number of state update messages sent and BW_{max} is the system's maximum available bandwidth.

V. System Architecture

In general, there is not a single solution to the multi-objective optimization problem defined by Equation 7. Instead, a set of solutions characterizing the trade-offs between individual objectives is obtained. This notion is formalized through the concepts of Pareto dominance and Pareto optimality.

Definition 1 (Pareto Dominance): Without loss of generality, assume a multi-objective minimization problem. A solution x dominates solution y if $f_i(x) \leq f_i(y)\ \forall i$ and $\exists j$ such that $f_j(x) < f_j(y)$. Pareto dominance is denoted $x \preceq y$.

Definition 2 (Pareto Optimality): A solution x is Pareto optimal if $\neg\exists\ x' \preceq x$; that is, if no other solution dominates x.

A set of Pareto optimal solutions is called a Pareto optimal set and its image in objective space is called the Pareto front. In solving multi-objective optimization problems, we seek to find or approximate the Pareto optimal set and its associated trade-offs represented by the Pareto front.

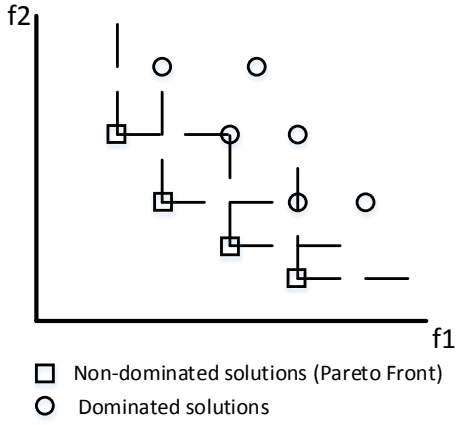


Figure 3. Pareto front, dominated solutions, and non-dominated solutions for a bi-objective minimization problem.

Figure 3 presents these concepts graphically for a bi-objective minimization problem. Square dots represent non-dominated solutions on the Pareto front. Round dots represent dominated solutions. The dotted lines represent dominance areas – a solution denoted by a square dot dominates any solution above and to its right. Although not drawn, this relationship holds for solutions not on the Pareto front as well.

Solutions to multi-objective optimization problems should lie on or as close as possible to the Pareto Front. Additionally, solutions should cover a broad section of the Pareto front. Multi-objective evolutionary algorithms are a preferred means of solving multi-objective optimization problems because they can find multiple Pareto optimal solutions in a single run. Additionally, multi-objective evolutionary algorithms are able to handle concavity and discontinuity on the Pareto front [16] making them ideal for exploring the trade-off space.

Implementation of a solver for the error threshold problem requires two fundamental subsystems: a multi-objective optimization routine, and a simulation of the virtual environment. We built optimization portion of our solver on the JMetal [17] multi-objective optimization framework. JMetal is a Java-based framework providing abstractions for problems, algorithms, and experiments. It includes a large number of MOEAs as well as standard benchmark problems. Additionally, JMetal provides an experimental framework capable of multiple independent runs and basic statistics gathering. We have extended JMetal with an implementation of the error threshold problem. This extension evaluates candidate solutions by invoking a virtual environment simulator, reading its output, and computing values for each objective function.

A simulation of the distributed virtual environment was developed using the OMNet++ [18] discrete event simulation

framework. OMNet++ is a discrete event simulation framework for building C++-based network simulation. Simulations are defined in terms of interacting modules that communicate via timed messages defining the events in the system. Runtime libraries are provided to manage the simulation infrastructure (e.g., the future events list, event scheduling, etc). Extensions provide a variety of network nodes and protocols to assist developers.

For each evaluation, our solver generates a network description file describing the node types, network topology, and parameters to simulate. With the exception of the dead-reckoning error threshold to evaluate, the contents of this file are fixed. The simulator is invoked with the error threshold under consideration and run for a configurable number of time steps. It outputs trajectory and message log files for each entity in the virtual environment.

The trajectory log file for each entity includes its actual position at each time step. It also includes, for each time step, the perceived location of all other entities in the system. Taken as a whole, these data allow us to reconstruct the actual and perceived locations for all pairs of entities at all times.

The message log file for each entity records the start time for each message sent as well as the time each incoming message was received. Taking these data as a whole allows us to compute maximum response times for each state update.

VI. Experiments

Validation of the multi-objective approach to setting error thresholds requires a particular virtual environment for investigation. This environment should be deterministic with well understood decision models and dynamics for each entity. Additionally, all entities should be synthetic to allow for statistically significant numbers of trials and long simulations. Finally, the dynamics of each entity should depend on one or more other entities and should be complex enough to provide interesting data.

We leverage Reynolds' boids model of flocking behavior [19] to provide a simple system with complex enough dynamics to generate an interesting Pareto front. The model defines flocking behavior as an emergent system property based on individual behaviors. Entities called boids move through a virtual space in three dimensions. The behavior of each boid is highly coupled to all other boids as each seeks to align its motion with its neighbors, steer towards the center of its neighbors, and avoid collisions. These simple behaviors allow complex flocking to emerge without explicitly designing it into the system.

Boids are attractive as a system model for a number of reasons. Firstly, individual boid behaviors are easy to reason about even though the aggregate flock behavior can be complex. Secondly, boids represent a worst-case scenario in that each entity's behavior depends on all other entities at every time step. Real systems with less coupling should outperform the boids model. Finally, we can define flock cohesion, or the average distance between a boid's flight path and the flock's mean flight path, as a simple measure of system performance.

TABLE I
Algorithm parameters.

Parameter	NSGA-II	SPEA2	MCTS
Population size	100	100	100
Archive size	100	100	100
Max evaluations	500	500	500
Crossover probability	0.9	0.9	-
Crossover distribution index	20.0	20.0	-
Mutation probability	1.0	1.0	-
Mutation distribution index	20.0	20.0	-
Exploration coefficient	-	-	$\frac{1}{\sqrt{2}}$

To investigate the shape of the error threshold Pareto front for the boids, a series of single factor experiments were undertaken. The goals of these experiments are to: 1) demonstrate the validity of the multi-objective optimization approach to determining dead-reckoning error thresholds, 2) ascertain the shape and location of the Pareto front for a representative virtual environment, and 3) compare the performance of the NSGA-II [20], SPEA2 [21], and MCTS [22] multi-objective optimization algorithms on the error threshold problem.

Two experiments were run using slightly different configurations. In the first, a 3 node fully-connected boids network was established. 30 runs were made for each of the NSGA-II, SPEA2, and MCTS algorithms. The simulation ran for 60,000 steps for each candidate solution. All network parameters (e.g., propagation delay, jitter, etc) were fixed and homogeneous. This leads to a constant response time based solely on the network's propagation delay. Therefore, a simple count of messages sent was substituted for Equation 5 with a goal of minimizing total traffic. This is a reasonable thing to do since it models aggregate traffic levels and is associated with system scalability.

The second experiment used a 5 node fully-connected boids network with time-varying network characteristics. Each link was given a constant propagation delay of 500 ms and a fixed bandwidth of 1.5 Mbps. For each transmission, jitter was sampled from a truncated normal distribution with mean of 100 ms and variance of 60 ms. Link saturation was modeled with a simple queuing mechanism – messages are held until the link becomes available. Retransmits and dropped packets were not modeled. Response time was measured as the second objective. 10 runs were made for each algorithm with 60,000 steps per simulation invocation.

Table I lists the parameters used for each algorithm. Note that crossover, mutation, and selection operators refer to built-in operators provided by JMetal. The exploration coefficient for MCTS sets a trade-off between exploration of new tree branches and exploitation of known good branches. For multi-objective problems, there should be a coefficient for each

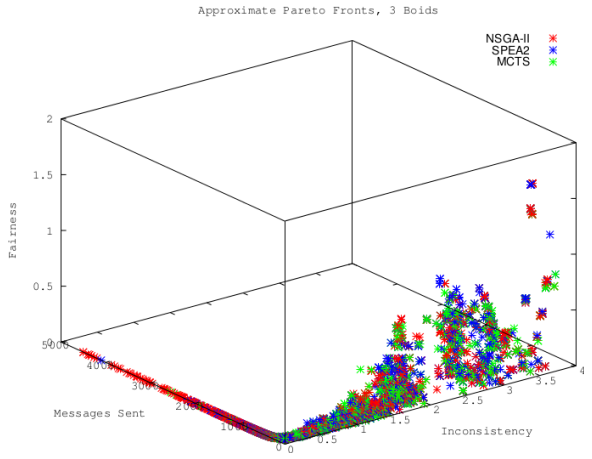


Figure 4. Scatter plot showing the approximate Pareto fronts returned by NSGA-II, SPEA2, and MCTS for Experiment 1.

TABLE II
WILCOXON RANK-SUM TEST RESULTS FOR EXPERIMENT 1.

	SPEA 2	MCTS
NSGAII	▲	▲
SPEA2		▲

objective. Since little was known *a priori* about the structure of the search space, all objectives use the same coefficient value.

VII. RESULTS AND ANALYSIS

Figure 4 plots the approximate Pareto fronts returned by NSGA-II, SPEA2, and MCTS in objective space for Experiment 1. All three algorithms achieve good convergence and diversity and show a distinct Pareto front. As expected, the best values cluster near the origin. There are well-defined trade-offs between inconsistency and message traffic and inconsistency and fairness. Low volumes of messaging result in high inconsistency and poor fairness.

Algorithm performance was compared using the hypervolume quality indicator [16]. The hypervolume indicator measures how much of the objective space is dominated by the solutions in a given set. Consequently, it provides a good indicator of both convergence to the Pareto front and diversity. The hypervolume was calculated for each algorithm run ($N = 30$). Algorithms were compared using the Wilcoxon rank-sum test against the null hypotheses that the median samples were drawn from the same distribution. The Wilcoxon results indicate NSGA-II outperforms both SPEA2 and MCTS. Additionally, one-way ANOVA indicates statistically significant differences in the sample medians ($p = 0.0,\ \alpha = 0.05$).

Figure 5a plots the approximate Pareto fronts returned by NSGA-II, SPEA2, and MCTS in objective space for Experiment 2. Figures 5b to 5d plot the planar projections of the data in Figure 5a. All three algorithms achieve good convergence and diversity and show a distinct Pareto front.

TABLE III
MANN-WHITNEY RANK-SUM TEST RESULTS FOR EXPERIMENT 2.

	SPEA2	MCTS
NSGA-II	-	▲
SPEA2		-

Algorithm performance was again compared with respect to the hypervolume indicator. Due to the small sample size ($N = 10$), the Mann-Whitney rank-sum test was used instead of the Wilcoxon rank-sum test. Results are tabulated in Table III. No statistical difference was found between NSGA-II and SPEA2 or between SPEA2 and MCTS. However, NSGA-II was found to outperform MCTS ($p = 0.0493,\ \alpha = 0.05$).

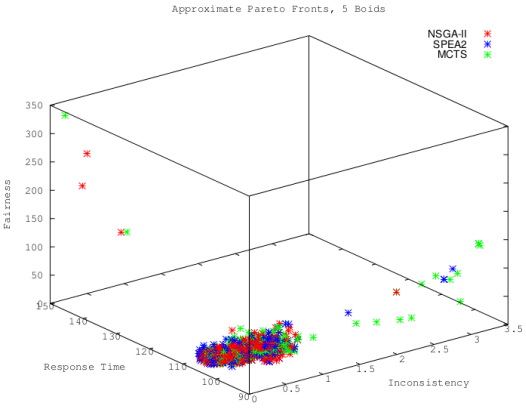
Both experiments show well-defined Pareto fronts indicating trade-offs between inconsistency, response time, and fairness. Additionally, there are definite lower limits to performance in any of these dimensions.

For example, Experiment 2 clearly shows that there is a minimum achievable inconsistency and that as inconsistency approaches this value, the response time increases dramatically. This drives an attendant degradation in fairness. This result is important since it implies that there are diminishing returns as one approaches the theoretical minimum for inconsistency (see Figures 5b and 5c).

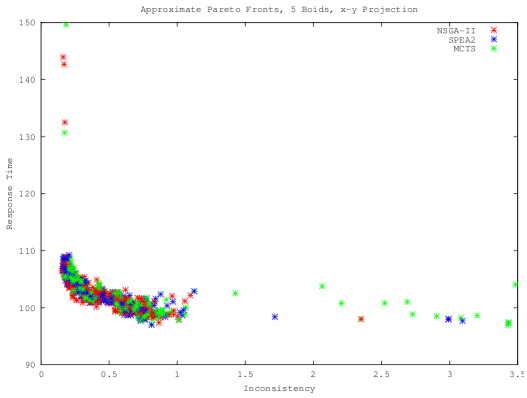
Fortunately, there is a wide area of acceptable performance with low inconsistency, low response time, and reasonable fairness. However, there is wide variability in fairness in this region driven primarily by changing message latency due to jitter and congestion. It should be noted; however, that while response times remain low in this region, the amount of state updates sent becomes rather large as borne out by Experiment 1. The systems under test in this work are small; real-world systems include many more entities and nodes. Thus, while response time may not become a design constraint, overall message volume may well limit scalability. Additionally, while not modeled here, one should also expect response time to increase as message volume increases due to network routing.

A second observation is that the achieved spatial inconsistencies are quite small. The lower bound is approximately 0.25 spatial units. The boids simulation under study uses a virtual world 20 units wide in each dimension, giving an achievable spatial error of about 1%. Whether this is an acceptable level of error depends on the purpose of the simulation. For the boids, velocities are small and 1% error is quite good as the boids maintained their flocking behavior. However, for an aircraft simulation with large velocities (e.g., supersonic), 1% error can translate to a large distance. This too is a valuable result for virtual environment designers – the best achievable objective values may well be too large for the intended application.

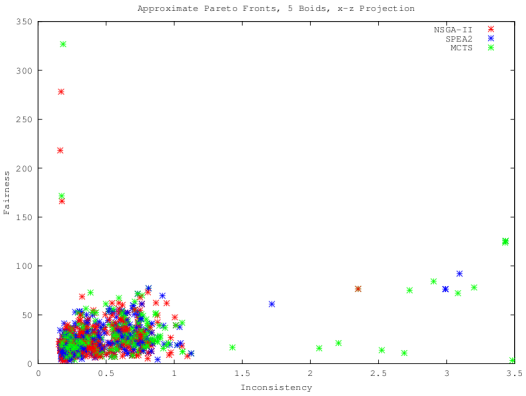
Additionally, use of a non-optimal threshold has distinct negative effects on system performance, i.e., flock cohesion. A three boid system with no network delay and perfect



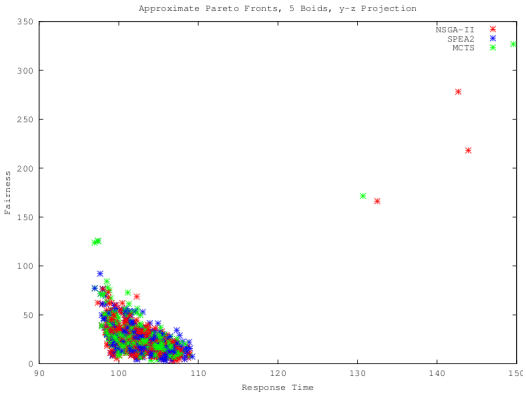
(a) Approximate Pareto fronts achieved by NSGA-II, SPEA2, and MCTS for Experiment 2.



(b) Projection on the inconsistency-response time plane.



(c) Projection front on the inconsistency-fairness plane.



(d) Projection front on the response time-fairness plane.

Figure 5. Approximate Pareto fronts achieved by NSGA-II, SPEA2, and MCTS for Experiment 2.

information (each entity has the exact position of all others at all times and dead reckoning is not necessary) achieves a flock cohesion measure of 7.667. Adding a modest network delay of 100 ms and an optimal dead reckoning threshold achieves a cohesion measure of 8.573. However, even a slightly non-optimal threshold of 2% spatial error results in increased inconsistency and a flock cohesion measure of 17.45.

Furthermore, conducting a study such as this during the system design phase can bring some insight into fitness of purpose. For instance, in the military domain one might wish to perform an operational test of some weapons system capability using existing simulation resources. If those resources are unable to achieve requisite consistency or response times based on an analysis such as this, the use of simulators should be abandoned. Similar results hold for other domains. In general, this approach can provide the designer some assurance that meeting performance requirements for consistency, response time, and fairness are achievable goals.

Finally, NSGA-II outperformed both SPEA2 and MCTS on the error threshold problem. This was unexpected since SPEA2 and most modern methods tend to outperform NSGA-II by a wide margin on standard test suites. However, the result is not unwelcome since NSGA-II executes faster than both SPEA2 and MCTS.

Some caution in applying these results is in order. It should be noted that the simulation used was a relatively low fidelity network simulation. Bit and packet errors were not modeled. Neither were routing effects, retransmits, or dropped packets.

VIII. Related Work

A common objective in virtual environment research and design is the maintenance of adequate consistency levels in the face of limited system resources such as throughput or network latency [23]. Several authors ([2]–[4], [23], [24]) have highlighted the trade-off between system consistency and system responsiveness as a defining characteristic of virtual environments.

Several papers have been published optimizing various aspects of consistency management and state update scheduling in particular [25]–[29]. Li and Cai [30] formulate the problem of minimizing inconsistency subject to network capacity constraints as a convex optimization problem. Tang and Zhou [31] derive optimal update schedules based on minimizing time-space inconsistency [15]. However, these analyses do not account for fairness as an explicit objective.

Chen and Zarki [14] define a relationship between system consistency, network delay, and quality of experience, including fairness, but do not provide methods for finding the optimal trade-off point.

IX. Conclusion

Choosing dead-reckoning error thresholds for distributed virtual environments is a non-trivial undertaking. The threshold choice impacts system performance in a number of dimensions including consistency, response time, and fairness. We have presented a multi-objective optimization model that accounts for the relationships between these three quantities. Additionally, we have built an architecture for the associated optimization problem based on multi-objective evolutionary algorithms and simulation of a simple virtual environment.

Experiments with simple virtual environment models indicate use of multi-objective optimization techniques is a viable means of choosing dead-reckoning thresholds. Analysis of the resulting Pareto fronts show diminishing returns as one approaches the theoretical minimum for state inconsistency. Additionally, our results highlight the trade-offs between consistency, response time, and fairness. Application of our model and solver with appropriate entity dynamics can assist virtual environment designers in tuning their applications for best possible performance.

References

[1] S. K. Singhal, "Effective remote modeling in large-scale distributed simulation and visualization environments," Ph.D. dissertation, Stanford University, 1996.

[2] S. Singhal and M. Zyda, *Networked virtual environments: design and implementation*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999.

[3] J. Smed and H. Hakonen, *Algorithms and Networking for Computer Games*. Wiley. com, 2006.

[4] A. Steed and M. F. Oliveira, *Networked Graphics: Building Networked Games and Virtual Environments*. Elsevier, 2009.

[5] P. Quax, P. Monsieurs, W. Lamotte, D. De Vleeschauwer, and N. Degrande, "Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 2004, pp. 152–156.

[6] J. Brun, F. Safaei, and P. Boustead, "Fairness and playability in online multiplayer games," *Faculty of Informatics-Papers*, p. 232, 2006.

[7] M. Mauve, "How to keep a dead man from shooting," in *Interactive Distributed Multimedia Systems and Telecommunication Services*. Springer, 2000, pp. 199–204.

[8] D. D. Hodson and R. O. Baldwin, "Performance analysis of live-virtual-constructive and distributed virtual simulations: defining requirements in terms of temporal consistency," 2009.

[9] Y. Zhang, L. Chen, and G. Chen, "Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2006, p. 7.

[10] DIS Steering Committee, "IEEE standard for distributed interactive simulation-application protocols," *IEEE Standard*, vol. 1278, 1998.

[11] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The department of defense high level architecture," in *Proceedings of the 29th conference on Winter simulation*. IEEE Computer Society, 1997, pp. 142–149.

[12] J. R. Noseworthy, "The test and training enabling architecture (TENA) supporting the decentralized development of distributed applications and lvc simulations," in *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*. IEEE, 2008, pp. 259–268.

[13] D. Marshall, S. McLoone, T. Ward, and D. Delaney, "Does reducing packet transmission rates help to improve consistency within distributed interactive applications?" 2006.

[14] P. Chen and M. El Zarki, "Perceptual view inconsistency: an objective evaluation framework for online game quality of experience (qoe)," in *Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2011, p. 2.

[15] S. Zhou, W. Cai, B.-S. Lee, and S. J. Turner, "Time-space consistency in large-scale distributed virtual environments," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, no. 1, pp. 31–47, 2004.

[16] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuisen, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.

[17] J. J. Durillo and A. J. Nebro, "jMetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0965997811001219

[18] A. Varga, "Omnet++," in *Modeling and Tools for Network Simulation*. Springer, 2010, pp. 35–59.

[19] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4. ACM, 1987, pp. 25–34.

[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.

[21] E. Zitzler, M. Laumanns, L. Thiele, E. Zitzler, E. Zitzler, L. Thiele, and L. Thiele, "Spea2: Improving the strength pareto evolutionary algorithm," 2001.

[22] W. Wang, M. Sebag *et al.*, "Multi-objective monte-carlo tree search," in *Asian conference on machine learning*, 2012.

[23] D. Delaney, T. Ward, and S. McLoone, "On consistency and network latency in distributed interactive applications: A surveypart i," *Presence: Teleoperators and Virtual Environments*, vol. 15, no. 2, pp. 218–234, 2006.

[24] ——, "On consistency and network latency in distributed interactive applications: A survey-part ii," *Presence: Teleoperators and Virtual Environments*, vol. 15, no. 4, pp. 465–482, 2006.

[25] K.-H. Shim and J.-S. Kim, "A dead reckoning algorithm with variable threshold scheme in networked virtual environment," in *Systems, Man, and Cybernetics, 2001 IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 1113–1118.

[26] Y. Li and W. Cai, "Consistency aware dead reckoning threshold tuning with server assistance in client-server-based dves," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 2925–2932.

[27] Z. Li, W. Cai, X. Tang, and S. Zhou, "Dead reckoning-based update scheduling against message loss for improving consistency in dves," in *Principles of Advanced and Distributed Simulation (PADS), 2011 IEEE Workshop on*. IEEE, 2011, pp. 1–9.

[28] Z. Li, X. Tang, W. Cai, and X. Li, "Compensatory dead-reckoning-based update scheduling for distributed virtual environments," *SIMULATION*, 2013.

[29] W. Cai, F. Lee, and L. Chen, "An auto-adaptive dead reckoning algorithm for distributed interactive simulation," in *Proceedings of the thirteenth workshop on Parallel and distributed simulation*. IEEE Computer Society, 1999, pp. 82–89.

[30] Y. Li and W. Cai, "Determining optimal update period for minimizing inconsistency in multi-server distributed virtual environments," in *Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*, ser. DS-RT '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 126–133. [Online]. Available: http://dx.doi.org/10.1109/DS-RT.2011.10

[31] X. Tang and S. Zhou, "Update scheduling for improving consistency in distributed virtual environments," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 6, pp. 765–777, 2010.

Data Quality Challenges in Distributed Live-Virtual-Constructive Test Environments

JEREMY R. MILLAR, DOUGLAS D. HODSON, GILBERT L. PETERSON and DARRYL K. AHNER, Air Force Institute of Technology

General Terms: Design, Measurement, Performance

Additional Key Words and Phrases: Distributed simulation, performance estimation

ACM Reference Format:
Jeremy R. Millar, Douglas D. Hodson Gilbert L. Peterson, and Darryl K. Ahner. 2015. Data Quality Challenges in Distributed LVC Test Environments *ACM J. Data Inform. Quality* V, N, Article XXXX (XXXX 2015), 3 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Live-virtual-constructive (LVC) simulations are complex systems comprising a combination of live (real people operating real equipment), virtual (real people operating simulated equipment, or vice versa), and constructive (wholly simulated) entities. Nodes in the system support the simulation of one or more entities and are often geographically distributed to leverage unique assets, e.g., physical test range space or high-fidelity full motion simulators. Nodes are connected in a peer-to-peer fashion and communicate using protocols such as Distributed Interactive Simulation (DIS) [DIS Steering Committee 1998], the High Level Architecture (HLA) [Dahmann et al. 1997], or the Test and Training Enabling Network Architecture (TENA) [Powell and Noseworthy 2012].

Distributed LVC simulation promises a number of benefits for the test and evaluation (T&E) community, including reduced costs, access to simulations of limited availability assets, the ability to conduct large-scale multi-service test events, and recapitalization of existing simulation investments. Consequently, the Department of Defense (DoD) is increasingly turning to LVC simulation and virtual environments to support T&E events. LVC simulations have been used to test communications for unmanned aircraft systems [Parker et al. 2009], conduct cyber security analysis [Van Leeuwen et al. 2010], and quantify radar measurement errors [Hodson et al. 2013].

Ensuring rigorous results for T&E events supported by LVC simulation requires addressing three fundamental data quality challenges: quantifying numerical errors due to weakly consistent nodes; assessing measurement accuracy with respect to tolerance requirements; and assessing measurment quality in the absence of absolute truth values.

Author's addresses: J. R. Millar, D. D. Hodson, and G. L. Peterson, Department of Electrical and Computer Engineering, Air Force Institute of Technology; D. K. Ahner, Department of Operations Research, Air Force Institute of Technology.

DOI: http://dx.doi.org/10.1145/0000000.0000000

2. DATA QUALITY CHALLENGES

LVC simulations are typically designed as fully replicated, geographically distributed database applications with real-time constraints. The data of interest is composed of entity and world state information and derived quantities such as collisions or weapons effectiveness. This data must be replicated at each node to meet availability and responsiveness requirements. The inclusion of live or virtual entities imposes real-time constraints on database responsiveness since long read/write latencies cannot be tolerated. Consequently, entity state updates (i.e., writes to a database record) are propagated to other system nodes after taking effect locally and are delayed due to network latency. Thus, not all nodes see the same simulation state at the same time. If updates cease, the system will eventually become consistent [Terry et al. 1994]. As such, LVC simulations can be viewed in the same context as eventually consistent distributed datastores such as Amazon's Dynamo [DeCandia et al. 2007], Cassandra [Lakshman and Malik 2009], or Megastore [Baker et al. 2011].

For any eventually consistent distributed database, a fundamental question is "How eventual is eventual?" Common measures of eventual consistency are time (how long does it take for readers to see the result of a write) and versions (how many versions old is a given read result) [Bailis and Ghodsi 2013]. For LVC simulations and other distributed virtual environments, deviation (e.g., Euclidean distance) from a "true" value is a common measure of consistency [Yu and Vahdat 2002; Aggarwal et al. 2004; Zhou et al. 2004].

Quantifying the numerical error associated with eventual consistency is a key challenge for LVC simulations. While there is a growing body of literature characterizing the consistency of distributed databases such as Dynamo and Cassandra [Wada et al. 2011; Rahman et al. 2012; Bailis et al. 2012], these works focus on time or version staleness as the measure of consistency, feature read-heavy workloads, and are not necessarily geographically distributed. In contrast, LVC simulations are more concerned with numerical error, have a balanced read/write workload, and are distributed geographically.

During a test event, measurements may be taken at any simulation node. Inconsistencies in the replicated state are reflected as measurement errors. A second challenge lies in assessing whether each measurement error lies within a precision tolerance. This assessment must be conducted during system design to ensure the simulation is capable of meeting test requirements. Additionally, it must occur during the test execution to provide a quantification of the uncertainty associated with each measurement.

A third challenge for LVC simulations is assessing the quality of measurements without a known truth value, particularly in the discrete case [Mauve 2000]. This is especially true for derived quantities that depend on inconsistent state data such as collisions and weapons effects. In this case, each interacting node may compute a result that is correct according to its state replica and different from other interacting nodes. Furthermore, the uncertainty can vary based on the node taking the measurement.

3. CONCLUSION

LVC simulations enable large-scale operationally relevant T&E events at reduced cost, provide access to limited availability assets, and recapitalize existing simulations. System architectures based on weakly consistent replicated databases and unreliable update protocols yield three fundamental challenges for data quality: quantifying error due to eventual consistency; assessing measurment accuracy with respect to desired tolerances; and assessing measurement quality in the absence of a truth value. Addressing these challenges is fundamental to ensuring the veracity and rigor of T&E events supported by LVC simulation.

REFERENCES

Sudhir Aggarwal, Hemant Banavar, Amit Khandelwal, Sarit Mukherjee, and Sampath Rangarajan. 2004. Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 161–165.

Peter Bailis and Ali Ghodsi. 2013. Eventual consistency today: limitations, extensions, and beyond. *Commun. ACM* 56, 5 (May 2013), 55–63. DOI:http://dx.doi.org/10.1145/2447976.2447992

Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. 2012. Probabilistically bounded staleness for practical partial quorums. *Proc. VLDB Endow.* 5, 8 (April 2012), 776–787. http://dl.acm.org/citation.cfm?id=2212351.2212359

Jason Baker, Chris Bond, James C Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. 2011. Megastore: Providing Scalable, Highly Available Storage for Interactive Services.. In *CIDR*, Vol. 11. 223–234.

Judith S Dahmann, Richard M Fujimoto, and Richard M Weatherly. 1997. The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*. IEEE Computer Society, 142–149.

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, Vol. 41. ACM, 205–220.

DIS Steering Committee. 1998. IEEE Standard for Distributed Interactive Simulation-Application Protocols. *IEEE Standard* 1278 (1998).

Douglas D Hodson, Alex J Gutman, Bruce Esken, and Raymond R Hill. 2013. Quantifying Radar Measurement Errors in a Live-Virtual-Constructive Environment to Determine System Viability: A Case Study. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* (2013), 1548512913503740.

Avinash Lakshman and Prashant Malik. 2009. Cassandra: structured storage system on a p2p network. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*. ACM, 5–5.

Martin Mauve. 2000. How to keep a dead man from shooting. In *Interactive Distributed Multimedia Systems and Telecommunication Services*. Springer, 199–204.

Eric Paul Parker, Nadine Elizabeth Miner, Brian Peter Van Leeuwen, and James Brian Rigdon. 2009. Testing unmanned autonomous system communications in a Live/Virtual/Constructive environment. *International Test and Evaluation Association Journal (ITEA)* 30 (2009), 513–522.

Edward T Powell and J Russell Noseworthy. 2012. The test and training enabling architecture (TENA). *Engineering Principles of Combat Modeling and Distributed Simulation* (2012), 449.

Muntasir Raihan Rahman, Wojciech Golab, Alvin AuYoung, Kimberly Keeton, and Jay J Wylie. 2012. Toward a principled framework for benchmarking consistency. In *Proceedings of the Eighth USENIX conference on Hot Topics in System Dependability*. USENIX Association, 8–8.

Douglas B Terry, Alan J Demers, Karin Petersen, Mike J Spreitzer, Marvin M Theimer, and Brent B Welch. 1994. Session guarantees for weakly consistent replicated data. In *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*. IEEE, 140–149.

Brian Van Leeuwen, Vincent Urias, John Eldridge, Charles Villamarin, and Ron Olsberg. 2010. Performing cyber security analysis using a live, virtual, and constructive (LVC) testbed. In *Military Communications Conference, 2010-MILCOM 2010*. IEEE, 1806–1811.

Hiroshi Wada, Alan Fekete, Liang Zhao, Kevin Lee, and Anna Liu. 2011. Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective.. In *CIDR*, Vol. 11. 134–143.

Haifeng Yu and Amin Vahdat. 2002. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems (TOCS)* 20, 3 (2002), 239–282.

Suiping Zhou, Wentong Cai, Bu-Sung Lee, and Stephen J Turner. 2004. Time-space consistency in large-scale distributed virtual environments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 14, 1 (2004), 31–47.

Received ; revised ; accepted

Consistency and Fairness in Real-Time Distributed Virtual Environments: Paradigms and Relationships

Jeremy R. Millar*, Douglas D. Hodson†, Gilbert L. Peterson‡, and Darryl K. Ahner§
Air Force Institute of Technology, Wright-Patterson AFB, OH 45433
Email: {jeremy.millar, douglas.hodson, gilbert.peterson, darryl.ahner}@afit.edu
Telephone: *937-255-3636 x4228 †x4719, ‡x4281, §x4708

***Abstract*—Distributed real-time virtual environments entail a well-known set of trade-offs resulting in a lack of state consistency across simulation nodes. This lack of consistency gives rise to a number of challenges that must be addressed by system designers to ensure users remain engaged and satisfied. This paper surveys several approaches to defining, measuring, and controlling the consistency of simulation state data and relates them to the concepts of fairness and fitness for purpose. The interplay between consistency and fairness, in relation to fitness for purpose must be carefully considered by system engineers to ensure appropriate design trade-offs.**

***Index Terms*—distributed virtual environments; simulation; consistency; fairness**

I. Introduction

Distributed virtual environments (DVEs) are real-time, man-in-the-loop, geographically distributed simulations sharing data in the same manner as a distributed database (Millar et al., 2015). They are found in a variety of application domains such as entertainment and gaming; collaboration systems; tele-robotics; training; and engineering test and analysis. DVEs require high levels of data consistency to support a shared sense of time, space, and interaction among users. Additionally, there are real-time constraints placed on the system's responsiveness to user inputs. Real-time responsiveness and data consistency are often contradictory requirements that define a trade space, particularly on public networks where users may be globally distributed and latencies can be quite high. Adequately understanding and addressing this tradeoff is necessary to keep users engaged and requires a thorough understanding of the interplay between real-time distributed database design, the effects of latency on user behavior, and the need for DVEs to present temporally and spatially consistent views to all users.

This article reviews the distributed virtual environment literature with respect to three key areas: 1) defining and measuring data consistency errors between database replicas, 2) update polices for maintaining state data consistency, and 3) methods for measuring and achieving fair playing fields where no user has a systemic advantage due to geographic or network location. Additionally, we describe the relationship between approaches used to characterize state space consistency and responsiveness issues. Finally, we provide some thoughts on the utility of DVEs with respect to engineering test and analysis applications.

The remainder of this paper is organized as follows: Section II outlines the system model and assumptions used throughout the paper. Section III defines several error models in common use by the DVE community. Section IV surveys common state update polices. Section V presents models of fairness in virtual environments. Finally, Section VI provides some commentary with regard to system design considerations and future research directions.

II. System Model

DVEs are often designed as a collection of loosely coupled, geographically distributed nodes communicating in a peer-to-peer networked architecture with the goal of presenting users with a shared sense of time and space (Singhal and Zyda, 1999). Nodes are loosely coupled in the sense that each makes no assumptions about the capabilities of other nodes beyond the ability to communicate using a common protocol. Nodes may be geographically distributed in order to take advantage of unique, one-of-a-kind assets or to bring together assets that are themselves geographically distributed. The use of a peer-to-peer communication model allows nodes to operate independently. DVEs of particular interest comply with one of several architectural standards such as Distributed Interactive Simulation (DIS) (DIS Steering Committee, 1998), the High Level Architecture (HLA) (Kuhl et al., 1999), or the Test and Training Enabling Architecture (TENA) (Noseworthy, 2008; Powell and Noseworthy, 2012).

Each node in a DVE may host one or more *entities* representing objects such as tanks or aircraft. Entities may represent real-world objects such as an aircraft on a training range (live entities), man-in-the-loop simulators (virtual entities), or wholly simulated assets (constructive entities). Nodes are responsible for processing user inputs, physics models, and state maintenance for their hosted entities. Additionally, nodes send entity state updates to other nodes so that each can maintain a representation of the overall simulation state. In the context of a single entity, the hosting node is termed the *server* while all other nodes are *clients* or *replicas*. Note that each node in the system acts as both server and client – a node is the server for each entity it simulates and a client for all others. This is sometimes referred to as a "serverless" architecture.

Simulation state can be divided into two types: 1) static, and 2) dynamic. Static state data includes items used by the

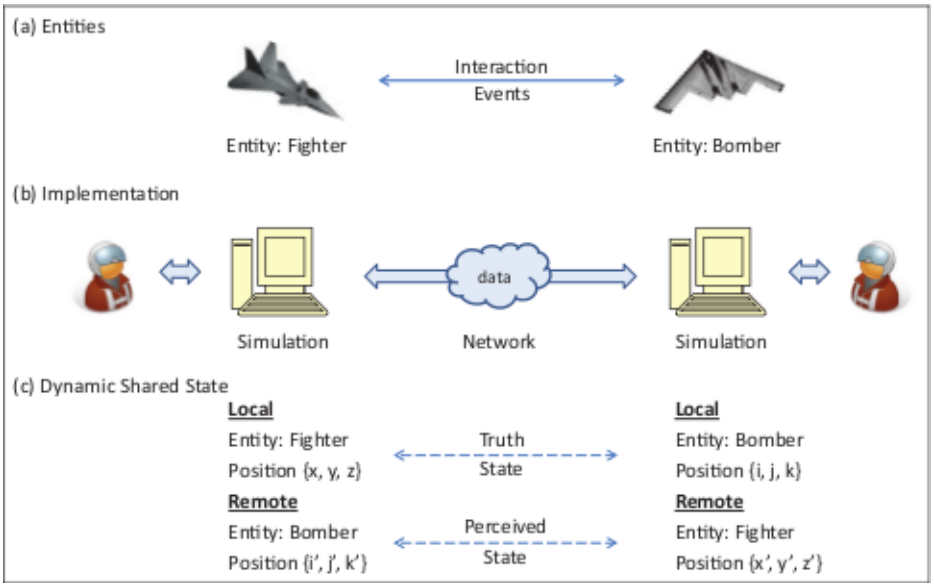


Fig. 1. Entities, implementation, and dynamic shared state (Hodson and Hill, 2013).

simulation that do not change, such as terrain. Dynamic state data consists of information about the entities or environment that can change over time such as entity position and orientation or weather. Dynamic state may be discrete or continuous. The sharing of dynamic state between nodes allows users to interact with one another as if they inhabit the same virtual environment.

This sharing and storage of dynamic shared state can be viewed as nodes maintaining a fully replicated, eventually consistent distributed database. Each record in this database holds the state data for a single entity. Nodes host replicas of the state database. Nodes write only to the records corresponding to the entities they host; all other records are read-only. For a given entity, the server node updates the replicas according to some policy – common policies include sending updates periodically or when a predicted error crosses some threshold value. The policy need not be the same for all replicas or data types. The state database is eventually consistent in the sense that once all user inputs cease, all replicas will converge to the same state (i.e., same values). In the meantime, reads from different replicas may return different values and in fact may return any value written since the beginning of the simulation (Terry, 2013).

Eventually consistent databases are employed by DVEs because supporting live and virtual entities imposes real-time processing constraints. Response times to user inputs for man-in-the-loop simulations should be less than 100 ms in order to avoid user-induced oscillation effects (Morse et al., 1996). These requirements, coupled with minimum data propagation delays, imply that absolute consistency in distributed real-time simulation is unachievable. Only after all inputs have ceased and some time has passed can each replica be guaranteed to have correct state data. Consequently, client-side state prediction algorithms are widely deployed to mitigate state errors between receiving updates from the server.

Figure 1 illustrates the basic structure of a distributed virtual environment. At the top of the figure are two entities, in this case two aircraft. These entities interact with one another in the virtual environment; for example, the fighter may try to intercept the bomber. The entities and their interactions are simulated by a pair of nodes communicating across a network as illustrated in the middle portion of Figure 1. Note that each simulation node acts as a server for its own entity and client for the other entity. Finally, the bottom portion of the figure depicts the dynamic shared state of the simulation, e.g., positional information. The state database consists of a position record for each entity. Records may be inconsistent, depending on network latency and other factors, as illustrated by the arrows labeled “Truth” and “Perceived” state.

III. Types of Error in Distributed Virtual Environments

Several types of error arise in DVEs as a consequence of the loose consistency guarantees provided by the state database. Let $e_s(t)$ and $e_c(t)$ be the state of some entity as stored in the server and client database replicas at time t. For any entity state for which there is an appropriate norm (e.g., position), we can define the spatial error at time t, $\mathrm{SE}(t)$, as the distance between the server and client representations of the entity state. That is,

$$\mathrm{SE}(t) = \|e_s(t) - e_c(t)\| \tag{1}$$

Aggarwal et. al. (Aggarwal et al., 2004) assumes a state update policy based on server-side dead reckoning and decomposes the spatial error into two components: dead reckoning error and export error. Once again, let $e_s(t)$ and $e_c(t)$ be the state of entity e as represented in the server and client state databases. Further, assume the server also maintains a dead

reckoned version of e as a means of predicting the spatial error at the client. Let $\hat{e_s}(t)$ represent this approximation (or estimate). Then the dead reckoning error $\mathrm{DE}(t)$ is given by

$$\mathrm{DE}(t) = \|e_s(t) - \hat{e_s}(t)\| \tag{2}$$

If at any time t the dead reckoning error exceeds some threshold, h, an update is sent to the client and $\hat{e_s}(t)$ is reset to the true entity state $e_s(t)$. Due to propagation delay factors such as network latency and processing time at the client, the entity state at the client will differ from the dead reckoned approximation at the server, i.e., $\hat{e_s}(t)$ and $e_c(t)$ are not necessarily equal – $\hat{e_s}(t)$ is an estimate of $e_c(t)$. This difference is termed the export error, $\mathrm{EE}(t)$, and is given by

$$\mathrm{EE}(t) = \|\hat{e_s}(t) - e_c(t)\| \tag{3}$$

The total spatial error is thus the sum of the dead reckoning error at the server and the export error at the client, i.e.,

$$\mathrm{SE}(t) = \|e_s(t) - e_c(t)\| = \mathrm{DE}(t) + \mathrm{EE}(t) \tag{4}$$

Integrating spatial error with respect to time yields a consistency measure known as time-space inconsistency (TSI) (Zhou et al., 2004), i.e.,

$$\mathrm{TSI}(t_0, t_1) = \int_{t_0}^{t_1} \mathrm{SE}(t)\ \mathrm{d}t \tag{5}$$

and draws an equivalence between large spatial errors of short duration and small errors of long duration.

TSI has been used as the basis of a number of state update scheduling algorithms and policies (Tang and Zhou, 2010; Li and Cai, 2010; Li et al., 2011; Li and Cai, 2011; Li et al., 2012, 2013). Most of these policies employ some form of server-side dead reckoning; consequently, the TSI can still be decomposed into dead reckoning and export errors.

There are some drawbacks to the use of TSI as an error measure. First, for certain classes of entity motion, TSI can grow without bound (Roberts et al., 2008). Second, choosing a dead reckoning threshold can be somewhat counter-intuitive. Finally, and of most importance to DVEs in a test and analysis context, the conflation of short duration, large distance errors with long duration, short distance errors can make it difficult to determine if a measurement based on the replicated state is within tolerable limits.

A second form of error arises with respect to the age of a state variable. Hodson defines the correctness of a variable as a function of a time interval – a state variable is accurate or valid for a period of time after being updated. This period of time is called the validity interval. A variable is deemed to be temporally consistent if the time of its last update, t_L, plus its validity interval, t_{VI}, is greater than or equal to the current time, i.e., if $t_L + t_{VI} \geq t$ (Hodson and Baldwin, 2009).

The temporal consistency of a state variable as seen by a client can be measured by calculating the mean and variance of its age in the client's state database. Figure 2 illustrates how a state variable ages at the client. An update received at t_i may

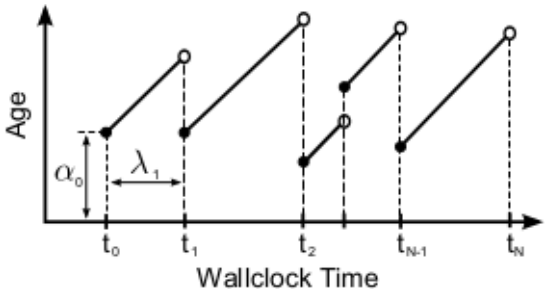


Fig. 2. Aging of distributed state data (Hodson and Baldwin, 2009).

have already aged due to transmission delays; in any case, the variable ages linearly until the next update is received. The interarrival time of the updates is denoted by λ_i and there are N aging intervals.

The mean age of the entity state as seen by the client is given by

$$\mu_{e_c} = \frac{1}{t_N} \sum_{i=1}^{N} \left(\frac{\lambda_i^2}{2} + \alpha_{i-1}\lambda_i \right) \tag{6}$$

where e_c is the entity state as represented in the client's state database, λ_i is the interarrival time, and t_N is the total elapsed wall-clock time over N intervals (Hodson and Baldwin, 2009).

The variance of the entity state's age at the client is given by

$$\sigma_{e_c}^2 = \mathrm{mse} - \mu_{e_c}^2 \tag{7}$$

where

$$\mathrm{mse} = \frac{1}{t_N} \sum_{i=1}^{N} \left(\frac{\lambda_i^3}{3} + \alpha_{i-1}\lambda_i^2 + \alpha_{i-1}^2\lambda_i \right) \tag{8}$$

The validity interval is determined by properties associated with how the state changes in relation to time. For any two interacting entities, there is a maximum acceptable error beyond which the interaction fails to behave correctly. For instance, spatial errors that are too large may cause a collision detection routine to trigger late, resulting in intersecting entities rendered to the display. Simulations supporting analysis can ill afford such imprecision, particularly for interactions comprising the system-under-test.

Given a precision requirement for each interaction supported by the simulation, the accuracy of the state database can be defined in a number of ways. Firstly, a replica is accurate with respect to entity e at time t if the spatial error for the entity is less than its associated precision requirement. This is a binary condition; the replica is either accurate or not. Formally, the accuracy at client c with respect to entity e is given by

$$A_e(t) = \begin{cases} 1 & \|e_s(t) - e_c(t)\| \leq p \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where $e_s(t)$ is the entity state at the server, $e_c(t)$ is the entity state at the client, and p is a (spatial) precision requirement.

The overall accuracy $\bar{A}(t)$ of a client is given by the mean accuracy over all entities, i.e.,

$$\bar{A}(t) = \frac{1}{|E|} \sum_{e} A_e(t) \tag{10}$$

where $|E|$ is the total number of entities in the simulation.

Calculating the mean and variance of $A_e(t)$ and $\bar{A}(t)$ over time provides summary measures of a particular client's state database accuracy.

IV. State Update Policies

A variety of state update mechanisms and policies have been proposed and investigated by the research community. The most basic policies are round-robin and periodic policies. Round-robin state update simply places each client in a circular queue based on the time it was last sent an update. At each frame, the server updates as many clients as possible, starting with the client least recently updated. This policy tends to perform poorly in systems where the total number of clients is much larger than the number of clients that can be updated at each simulation frame.

Periodic update policies send updates to clients at defined intervals. Note that the round-robin policy is in fact periodic, however, much more complex periodic schemes are possible. For instance, some DVE systems provide state update data streams with multiple periods for the same entity. This allows clients to choose how often they receive updates – clients whose entities are not interacting with the server's choose a long period stream, while clients needing more state accuracy can choose a high frequency stream. See (Morse et al., 1996) for a review of this and other interest management techniques.

Mauve proposed the use of local lag policies to improve state consistency among nodes in distributed virtual environments (Mauve, 2000). Local lag retards the effect of user inputs at the local node while sending a state update to the remote nodes. Ideally, the lag applied locally is equivalent to the propagation delay of the state update message and all nodes see the effects of the input simultaneously. Practice is rarely so accommodating since each client may have a different amount of delay, packets may be dropped, and network latency can vary. Note that local lag degrades responsiveness in favor of improved state consistency.

The DIS standard specifies the use of dead reckoning algorithms similar to those outlined in Section II. The vast majority of research on data consistency issues in DVEs focuses on improving and optimizing dead reckoning for a variety of conditions. An early effort by Cai, Lee, and Chen proposed an auto-adaptive error threshold based on areas of interest and sensitive regions around each entity (Cai et al., 1999). The area of interest is defined as a circular region around an entity in which the entity requires increased consistency. The sensitive region is a smaller circular region. If one entity moves into another's sensitive region, a collision or some other kind of interaction is likely. Thus the consistency requirements inside and entity's sensitive region are higher still. The authors define four threshold levels based on the relative position and overlap of entities and their regions. They conducted a series of experiments using a simulated distributed environment and measure the average spatial error and number of messages for differing numbers of entities. Their adaptive threshold algorithm shows an improvement in both the number of messages sent and the average error. Unfortunately, they provided no guidance for choosing the threshold values for their algorithm. This is important since the superiority of their approach depends on these choices. In general, however, one can expect the adaptive algorithm to strike an adequate balance between consistency and network utilization while showing an improvement over standard dead reckoning.

Liu (Lui, 2001) explored the use of communication subgraphs to determine the optimal synchronization interval for DVEs with an upper bound on spatial error. He compared the use of several subgraph generation algorithms to bound the maximum delay between nodes in the virtual environment. The optimal synchronization interval was determined using a discrete time Markov chain model of the spatial error. Once the optimal interval is found, the network utilization for each of the communication subgraphs can be computed.

Delaney, Ward, and McLoone (Delaney et al., 2003) proposed a hybrid algorithm that pairs the conventional dead reckoning model with an experimentally derived model of the user's long term strategy. The goal of the hybrid technique is to reduce network utilization at a given threshold level. Their work shows an improvement in the number of messages sent, albeit for a relatively large spatial error threshold.

Yu *et al* (Yu et al., 2007) consider the problem of allocating bandwidth to nodes in order to minimizes the spatial error, subject to time-varying bandwidth constraints. They construct the problem as a constrained convex optimization problem and apply Lagrangrian relaxation to make the trade-off between consistency and network utilization. Minimization of the relaxed problem is accomplished with binary search and predictive model of bandwidth consumption based on recent history. Their work provides an empirical look at the fundamental DVE trade-off between consistency and bandwidth consumption. Although they describe in general terms a network aware bandwidth allocation algorithm, and show provide results showing it outperforms uniform and proportional allocations, the algorithm is not specified in detail.

Note the preceding works attempt to optimize dead reckoning with respect to network consumption rather than consistency. Excessive bandwidth consumption is generally not an issue except when scaling a DVE to massive numbers of clients. In many practical scenarios the number of clients is bounded and minimizing the number of messages is not required – after all, unused bandwidth is wasted bandwidth. The remainder of this section discusses techniques that specifically optimize dead reckoning for consistency.

Building off Zhou *et al*'s work on time-space inconsistency (Zhou et al., 2004), Roberts *et al* proposed the use of a time-space threshold instead of a spatial error threshold for dead reckoning (Roberts et al., 2005). They begin by noting that simple spatial thresholds can lead to unbounded inconsistency

if the entity deviates from the dead reckoned path but remains inside the error threshold. Using a time-space threshold mitigates this problem, however, doing so can lead to large spatial errors over short time periods. The authors demonstrate a hybrid threshold metric that avoids both of these issues.

Tang and Zhou (Tang and Zhou, 2010) developed a consistency aware update scheduling algorithm for centralized, single server DVEs with network capacity constraints. They begin by deriving optimal update schedules to minimize the impact of time-space inconsistency on user perceptions, noting that these schedules depend on the network delay and entity trajectories. The latter, of course, depend on user inputs to the simulation. The authors then present an update scheduling algorithm that estimates the time-space inconsistency at each client and sends updates to the clients with the largest error, subject to network capacity constraints. Any clients that are not updated will necessarily have a larger inconsistency during the next frame and will have a higher priority for update. Their algorithm shows substantially better performance than more naive scheduling algorithms such as round-robin or conventional dead reckoning.

Li and Cai (Li and Cai, 2011) extend the discussion to a multi-server environment and frame the problem as one of determining the optimal update period for each replica. They formulate the problem as convex optimization problem with inequality constraints and apply the barrier method for solution.

Both of the foregoing methods assume static network conditions with no possibility of message loss. Li *et al* (Li et al., 2012) consider the effects of message loss on update scheduling; however, they ignore the effects of network delay. They develop scheduling algorithms that compensate for message loss by sending updates when the time-averaged inconsistency with loss exceeds the inconsistency without loss.

Li, Tang, Cai, and Li (Li et al., 2013) propose an update scheduling algorithm that modifies the dead reckoning threshold based on predicting network conditions and estimating whether the system can compensate for the effects of network delay and message loss. If the delay and loss rate can be compensated, the threshold is modified accordingly; if not, an update message is sent immediately.

V. Fairness Issues

In designing distributed virtual environments, it's important to ensure the system is fair. Here, fairness is taken to mean that each node in the system interoperates in such a way that no node has a systemic advantage or disadvantage (Siegfried et al., 2011). This is particularly important for comparative analyses where two or more systems are being evaluated against each other with a common objective.

Several authors have proposed various measures of fairness. The simplest of these compute the spatial error variance for each entity across all system nodes. Various update policies have been proposed to ensure fairness according to this simple model (Aggarwal et al., 2005; Li et al., 2011, 2012).

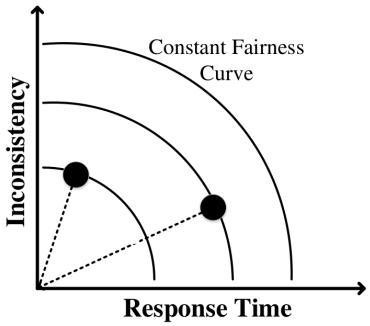


Fig. 3. Fairness space (Chen and El Zarki, 2011).

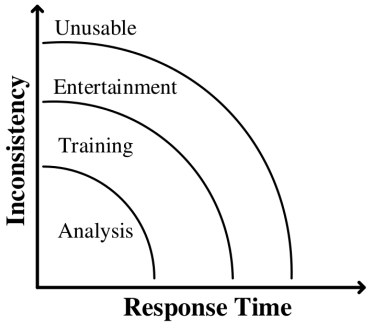


Fig. 4. Fairness zones.

Brun *et al* and Chen and Zarki each approach fairness from the perspective of a multi-dimensional vector space called playability space or fairness space, respectively (Brun et al., 2006; Chen and El Zarki, 2011). Typically these spaces are two-dimensional, with spatial error or inconsistency on one axis and response time on the other as illustrated in Figure 3. Each node in the system is plotted in this space. A node's distance from the origin is termed its playability. The fairness of the simulation is an aggregate property and is defined as the standard deviation of playability measured across all nodes. Millar *et al* alternately defined fairness with respect to cluster cohesiveness in the playability space – systems whose nodes are tightly clustered in the playability space are fair, while those whose nodes are scattered in playability space are not (Millar et al., 2014).

Playability and fairness spaces provide the most general models of fairness in the literature; however, they can be difficult to compute online and no existing update policies take advantage of them. Fairness can be explicitly managed by tuning update policy parameters. Doing so tends to push the spatial error and response time of all nodes towards that of the worst performing node. Thus, the system may well be fair but perform poorly.

Brun *et al* mentions the idea of a playability zone defined by a utility function assigned to each interaction (Brun et al., 2006). This dovetails nicely with Siegfried *et al*'s assertion

that fairness requirements vary by application and purpose. Indeed, requirements for consistency, responsiveness, and fairness can vary within a single application depending on entity interactions (Savery et al., 2010). Extrapolating from this idea and considering the different application domains, the authors view fairness as not just a runtime property impacting users' quality of experience, but also as a design property reflecting a particular system's fitness for purpose. This idea is illustrated in Figure 4.

Here, the fairness space is sub-divided into four equivalence classes based on the simulation's broadly intended purpose. Choices made by the system designers regarding the state update scheduling policy and its parameters directly impact whether the simulation is capable of fulfilling that purpose. As an example, an aerial dogfight simulation intended as an online game only needs to achieve consistency and response time levels sufficient to engage players and keep them from quitting in frustration. If the aim of the simulation is instead training; e.g., to teach basic fighter maneuvering, then the constraints on consistency and response time get tighter. This is necessary to ensure that training outcomes transfer to the real world. Finally, if the aim of the simulation is to test some aspect of the system (e.g., a new gunsight), then constraints on consistency and responsiveness become tighter still in order to minimize confounding factors due to the distributed environment. It is undesirable, for instance, for inconsistency in the entity state data to yield incorrect experimental conclusions or for measurements of an event taken on different system nodes to vary widely.

The state update policy and parameters must be carefully chosen to ensure that the consistency and responsiveness of the simulation remain within constraints defined by its purpose. For instance, use of local lag techniques are acceptable in online games with low interactivity and high consistency requirements such as Pokemon Go. Conversely, use of local lag in highly interactive flight training environments can result in pilot induced oscillations and a failure to meet training requirements due to "fighting the sim." In any case, a loss of perceived fairness results in user disengagement and ultimately an unwillingness to participate in the DVE (Yasui and Ishibashi, 2004; Chen et al., 2009; Beznosyk et al., 2011)

VI. Conclusions

This article presents a review and survey of the distributed virtual environment literature with respect to three key areas: 1) state consistency errors, 2) state update policies, and 3) fairness issues. The choice of a particular consistency measure and state update algorithm can have a direct impact on the system's fairness. For instance, under the TSI error model, small constant spatial errors result in an unbounded measure of inconsistency. A state update policy controlling for fairness might well be required to introduce lag to ensure TSI variation remains small across all simulation nodes. As a consequence, the overall simulation may become unfit for its intended purpose, e.g., if the average inconsistency crossed some maximum threshold. This particular example is not insurmountable; however, it serves to illustrate the larger point – there are complex relationships between consistency, update policy, fairness, and fitness for purpose. Consequently, we conclude by identifying some areas we believe warrant further research.

In practice, distributed virtual environments tend to be designed using optimistic state update policies intended to minimize state inconsistency, e.g., basic dead-reckoning. However, none of the surveyed policies provide upper bounds on the amount of inconsistency they allow. Nor do they provide any *a priori* estimation of the amount of inconsistency likely to be observed in practice. Thus a best-of-breed update policy may well provide minimal inconsistency and still exceed the system's requirements. Research into predictive models of virtual environment consistency is needed to provide designers the tools to adequately choose and tune state update policies.

Additionally, consistency is generally measured with respect to continuous state variables with easily derived rates of change and relatively simple error measures. For instance, nearly all of the error models surveyed are related to entity position information. However, many of the interesting variables in a distributed virtual environment (Did I hit my enemy? Is my enemy dead?) are discrete in nature. While these discrete variables often rely on continuous variables, e.g., collision detection depends on position data, our survey of the literature found very little related to the impact of inconsistent discrete state. Moreover, the consistency of discrete state variables is directly related to the plausibility of outcomes between interacting entities and is a key challenge for distributed virtual environments supporting engineering test and analysis applications.

In general, DVEs cannot provide state consistencies stronger than eventual consistency. This implies there is no upper bound on the divergence in state between any to nodes in the system during the simulation's execution. Consequently, we believe the use of DVEs in test and analysis should be limited to exploratory analysis and analysis of alternatives to minimize the impact of implausible outcomes. Moreover, we believe state update policies for test and analysis applications should be approached from the point of view of measurement calibration – that is, the simulation system should be viewed as an experimental device who's precision must be determined and calibrated before it can be used for meaningful experiments. Our future research includes development of stochastic plausibility models designed to quantify the expected precision of DVEs used to support engineering test and analysis.

References

Aggarwal S, Banavar H, Khandelwal A, Mukherjee S, and Rangarajan S (2004). Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 161–165. ACM.

Aggarwal S, Banavar H, Mukherjee S, and Rangarajan S (2005). Fairness in dead-reckoning based distributed multi-player games. In *Proceedings of 4th ACM SIGCOMM*

workshop on Network and system support for games, pages 1–10. ACM.

Beznosyk A, Quax P, Coninx K, and Lamotte W (2011). Influence of network delay and jitter on cooperation in multiplayer games. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pages 351–354. ACM.

Brun J, Safaei F, and Boustead P (2006). Fairness and playability in online multiplayer games. *Faculty of Informatics-Papers*, page 232.

Cai W, Lee F, and Chen L (1999). An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 82–89. IEEE Computer Society.

Chen K.-T, Huang P, and Lei C.-L (2009). Effect of network quality on player departure behavior in online games. *Parallel and Distributed Systems, IEEE Transactions on*, 20(5):593–606.

Chen P and El Zarki M (2011). Perceptual view inconsistency: An objective evaluation framework for online game quality of experience (qoe). In *Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*, NetGames '11, pages 2:1–2:6, Piscataway, NJ, USA. IEEE Press.

Delaney D, Ward T, and McLoone S (2003). On reducing entity state update packets in distributed interactive simulations using a hybrid model. IASTED.

DIS Steering Committee (1998). IEEE standard for distributed interactive simulation-application protocols. *IEEE Standard*, 1278.

Hodson D. D and Baldwin R. O (2009). Characterizing, measuring, and validating the temporal consistency of live–virtual–constructive environments. *Simulation*, 85(10):671–682.

Hodson D. D and Hill R. R (2013). The art and science of live, virtual and constructive simulation for test and analysis. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, page 1548512913506620.

Kuhl F, Weatherly R, and Dahmann J (1999). *Creating computer simulation systems: an introduction to the high level architecture*. Prentice Hall PTR.

Li Y and Cai W (2010). Consistency aware dead reckoning threshold tuning with server assistance in client-server-based dves. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 2925–2932. IEEE.

Li Y and Cai W (2011). Determining optimal update period for minimizing inconsistency in multi-server distributed virtual environments. In *Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT '11, pages 126–133, Washington, DC, USA. IEEE Computer Society.

Li Z, Cai W, Tang X, and Zhou S (2011). Dead reckoning-based update scheduling against message loss for improving consistency in dves. In *Principles of Advanced and Distributed Simulation (PADS), 2011 IEEE Workshop on*, pages 1–9. IEEE.

Li Z, Cai W, Tang X, and Zhou S (2012). Loss-aware dr-based update scheduling for improving consistency in dves. *Journal of Simulation*, 6(3):164–178.

Li Z, Tang X, Cai W, and Li X (2013). Compensatory dead-reckoning-based update scheduling for distributed virtual environments. *SIMULATION*.

Lui J. C. S (2001). Constructing communication subgraphs and deriving an optimal synchronization interval for distributed virtual environment systems. *Knowledge and Data Engineering, IEEE Transactions on*, 13(5):778–792.

Mauve M (2000). How to keep a dead man from shooting. In *Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 199–204. Springer.

Millar J. R, Hodson D. D, Lamont G. B, and Peterson G. L (2014). Multi-objective optimization of dead-reckoning error thresholds for virtual environments. In *Collaboration Technologies and Systems (CTS), 2014 International Conference on*, pages 562–569. IEEE.

Millar J. R, Hodson D. D, Peterson G. L, and Ahner D. K (2015). Data quality challenges in distributed Live-Virtual-Constructive test environments. *ACM Journal of Data and Information Quality*.

Morse K. L et al. (1996). *Interest management in large-scale distributed simulations*. Information and Computer Science, University of California, Irvine.

Noseworthy J. R (2008). The test and training enabling architecture (TENA) supporting the decentralized development of distributed applications and lvc simulations. In *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*, pages 259–268. IEEE.

Powell E. T and Noseworthy J. R (2012). The test and training enabling architecture (tena). *Engineering Principles of Combat Modeling and Distributed Simulation*, page 449.

Roberts D, Aspin R, Marshall D, Mcloone S, Delaney D, and Ward T (2008). Bounding inconsistency using a novel threshold metric for dead reckoning update packet generation. *Simulation*, 84(5):239–256.

Roberts D, Marshall D, MacLoone S, Delaney D, Ward T, and Aspit R (2005). Exploring the use of local consistency measures as thresholds for dead reckoning update packet generation. In *Distributed Simulation and Real-Time Applications, 2005. DS-RT 2005 Proceedings. Ninth IEEE International Symposium on*, pages 195–202. IEEE.

Savery C, Graham T, and Gutwin C (2010). The human factors of consistency maintenance in multiplayer computer games. In *Proceedings of the 16th ACM international conference on Supporting group work*, pages 187–196. ACM.

Siegfried R, Lüthi J, Herrmann G, and Hahn M (2011). How to ensure fair fight in lvc simulations: Architectural and procedural approaches. *NATO Modelling and Simulation Group*, 87.

Singhal S and Zyda M (1999). *Networked virtual environments: design and implementation*. ACM Press/Addison-

Wesley Publishing Co., New York, NY, USA.

Tang X and Zhou S (2010). Update scheduling for improving consistency in distributed virtual environments. *Parallel and Distributed Systems, IEEE Transactions on*, 21(6):765–777.

Terry D (2013). Replicated data consistency explained through baseball. *Communications of the ACM*, 56(12):82–89.

Yasui T and Ishibashi Y (2004). Consistency and fairness among players in networked racing games: Influence of network delays. In *Proceedings of the ICAT '04 Workshop on VR Applications and Entertainment Technology*, pages 43–47.

Yu Y, Li Z, Shi L, Chen Y.-C, and Xu H (2007). Network-aware state update for large scale mobile games. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 563–568. IEEE.

Zhou S, Cai W, Lee B.-S, and Turner S. J (2004). Time-space consistency in large-scale distributed virtual environments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(1):31–47.

Deriving LVC State Synchronization Parameters from Interaction Requirements

Jeremy R. Millar*, Douglas D. Hodson† and Richard Seymour‡
Air Force Institute of Technology, Wright-Patterson AFB, Dayton OH 45344
Email: *jmillar@afit.edu, †dhodson@afit.edu, ‡rseymour@afit.edu

***Abstract*—Choosing synchronization update parameters for live, virtual, constructive simulations is of particular importance when the simulation is supporting engineering test and evaluation events. Failure to choose these parameters appropriately can lead to substantial data quality problems. This work introduces the notion of plausibility limits for entity interactions based on spatial errors derived from state variable divergence. Moreover, it presents a state update model that provides probabilistic guarantees on meeting interaction requirements expressed as plausibility limits. Finally, it presents an example of these guarantees based on entity state data sampled from a popular online game.**

Live, virtual, and constructive (LVC) simulation technologies are gaining traction within the US Department of Defense (DoD) test and evaluation community as a means of supporting analysis and decision making processes [1], [2], [3]. LVC environments provide opportunities to facilitate testing in joint environments, include a broad spectrum of defense assets not readily available to live test, and reduce test schedules and costs [3].

The DoD defines three primary classes of simulation: 1) live, 2) virtual, and 3) constructive. Live simulations comprise real people operating real equipment in the real world; for instance, traditional military training exercises. Virtual simulations involve real people operating simulated equipment in a simulated environment such as man-in-the-loop flight simulators. Constructive simulations are wholly synthetic; the entities, equipment, and environment are all simulated.

LVC simulations are hybrid simulations combining aspects of live, virtual, and constructive simulations to various degrees [3]. They often involve real people and systems interacting with simulated entities in a virtual environment. From a participant's perspective, it makes no difference if another entity is real or simulated since all interactions occur through a virtual environment. Moreover, the participants are often geographically distributed in order to minimize costs or to leverage unique or rare assets. The virtual environment provides a shared sense of time and space to the simulation's participants, allowing them to interact as if they were actually present in the same location.

A key issue for LVC simulations, as for all distributed virtual environments, is the maintenance of dynamic shared state between participating nodes. The dynamic shared state comprises all of the changing information that must be maintained at each participating site to present a consistent view of the virtual environment. Frequently, the dynamic shared state includes information about who is participating in the simulation, the position of various moving entities, and so forth. Failure to keep the dynamic shared state adequately synchronized across sites destroys the illusion of shared time and place that LVC simulations and other virtual environments require [4].

In general, absolute consistency of dynamic state can be ensured by requiring acknowledgments for state update messages and forcing locked time-step execution of the distributed simulation across all sites. Doing so necessarily reduces the simulation's responsiveness to user input as the user must wait until previous inputs have propagated throughout the entire system before issuing new commands. Thus, there is a fundamental trade-off between state consistency and simulation responsiveness [4], [5]. Lock-step synchronization is not an option for simulations that include live entities; for example, an aircraft must continue moving while state updates propagate, resulting in a loss of absolute consistency.

Typically, consistency requirements are relaxed in order to improve responsiveness. Speculative synchronization protocols such as dead reckoning allow the simulation to respond immediately to new user inputs and continue execution while state updates are propagated. This approach is standardized in IEEE Standard 1287, Distributed Interactive Simulation (DIS) [6].

The degree to which consistency and responsiveness are relaxed as well as the algorithms employed to do so are important decisions in the design of LVC simulations. Some of these decisions are constrained by other requirements; for instance, most DIS compliant simulations employ speculative state synchronization in the form of dead reckoning. Several pertinent questions arise; for instance, how often should state updates be sent? Should user input be delayed to account for network latency? If so, by how much? That is, even within the relevant standards, simulation designers have wide latitude to select synchronization protocols and parameters.

There is little traceability from high-level simulation requirements to low-level synchronization parameters. Often, a simulation is constructed and the synchronization protocol is hand tuned until the system's performance appears correct – most likely by subjective means. In the best case, the state synchronization system meets or exceeds the consistency and responsiveness required to meet the simulation's goals. In the worst case, the (unspecified) consistency and responsiveness targets are not met, and the simulation may behave in subtle and incorrect ways.

Since consistency is generally relaxed in favor of responsiveness, these errors most often manifest as spatial errors wherein each simulation participant (or node) perceives the same entity at different locations. This breaks the fundamental illusion of a shared space. We propose that the effect of these spatial errors can be mitigated by explicitly accounting for entity interaction requirements while designing the simulations synchronization subsystem.

The contribution of this paper is two-fold. First, it presents a survey of the existing literature on entity interactions and their requirements and defines a formalism for entity interaction requirements based on error tolerances. Second, it presents a method of deriving synchronization parameters from interaction requirements and a model of spatial error in dead reckoning-based simulations.

I. System Model

A distributed LVC simulation system consists of two or more participating nodes connected by a communication network. Each node is responsible for simulating the state of one or more entities based on user inputs, physics models, and interactions with other entities and the environment.

In addition to maintaining the state of its own entities, each node also keeps a replica of the state for every other entity in the simulation. Thus, the simulation state can be viewed as a geographically distributed database with full replication [7]. Crucially, each node only writes to the records associated with its local entities; all other records are read-only. Replica records are updated when a message is received from the owning node. Optimistic consistency maintenance algorithms such as dead reckoning may be employed to improve consistency across replicas.

II. Interactions

Interactions between simulated entities are a defining characteristic of LVC simulations; without interactions, there would be no need to design and built a distributed simulation system. Indeed, the difference between two simulations largely comes down to the set of entities involved and the interactions among them. For instance, an air-to-air combat simulation might include ground units as part of the setting or window dressing, but not allow aircraft to interact with them. Conversion from an air-centric to combined arms simulation is as simple as expanding the set of allowed interactions to include entities on the ground.

In any case, the set of interactions supported by a simulation directly impact it's consistency and responsiveness requirements. Some interactions, such as collision detection or engaging an entity with a high-precision weapon, have a low tolerance for inconsistency in the simulation state. Others, such as tracking a target with a long-range sensor, can tolerate higher levels of inconsistency before erroneous results are generated.

In [8], Itzel et. al. present the notion of the interaction context for a state update. This context captures the type, affected entities, and any dependent interactions for an update message. From this information, system designers can derive consistency and interactivity requirements. For instance, state updates for an entity moving through the environment without affecting with any other entities would have low consistency and interactivity requirements. Updates for two entities engaged in combat, on the other hand, would have high consistency and interactivity requirements.

Similarly, Claypool and Claypool have categorized user actions in online games according to their precision and deadline requirements [9]. Here, precision means the degree of accuracy necessary to complete the (inter)action successfully and is analogous to a consistency requirement. High precision actions show a high sensitivity to state inconsistency, while low precision actions are more tolerant of inconsistent state. Deadline refers to the time required to achieve the final result of an action. The authors show that network latency has a larger impact on game actions with tight precision or deadline requirements.

The foregoing works set the stage for an investigation of interactions, however, they have some limitations when applied to DIS-based LVC simulations. First, both assume a centralized client/server system responsible for processing user actions. Itzel's interactivity and Claypool's deadline requirements are derived from this assumption. Highly interactive actions (those with tight deadlines) require a rapid response from the server processing the actions. This is exactly what is meant by simulation responsiveness [4], [5]. However, peer-to-peer LVC simulations are able to respond immediately to user inputs since no communication with other system nodes is required. With that said, responsiveness does impact an LVC system's ability to meet interaction requirements; Section III explores this relationship.

Second, and more problematically, both Itzel and Claypool characterize the consistency (precision) requirements derived from examining interactions categorically. This makes it difficult to derive concrete system requirements. For example, just how precise must a "high" precision action be? A more formal model of interaction is required to accurately derive consistency and responsiveness requirements from interaction requirements.

Suppose we have a peer-to-peer LVC simulation with n entities denoted e_i, $i = 1 \ldots n$. An *interaction* expresses a state dependency between two entities and is denoted by the tuple

$$I = (e_i, e_j, l) \tag{1}$$

The source of the interaction, e_i, depends on the state of the target entity e_j. Note the dependence is unidirectional. This allows asymmetric interactions to be modeled. For instance, e_i may be a long-range sensor tracking e_j at a sufficiently large distance so that e_j is unaware of e_i. Situations involving two entities that are mutually interacting can be modeled with a pair of interactions I_1 and I_2.

The spatial error tolerance of an interaction is given by l and specifies the maximum spatial error the interaction can support before producing incorrect results. We refer to this maximum error as the *plausibility limit* of an interaction since errors in excess of l are likely to result in implausible results on at least one node of the simulation. Note that we are only concerned with positional state data since nearly every interaction depends on accurate position information. Moreover, positional updates account for nearly all of the network traffic generated by an LVC simulation [10].

The parameters of a particular interaction should be defined by the engineering requirements the simulation is meant to address. For instance, in an aerial refueling simulation, if the boom operator must place the boom within 10 cm of the fuel receptacle to successfully couple the aircraft, then the plausibility limit for that interaction would be 10 cm.

III. Spatial Error Model

Deriving state synchronization parameters from interaction tolerances requires a model for how spatial state inconsistency develops in the first place. The node hosting the target entity e_j of an interaction maintains the true value of its state. If the replica of that state at the node hosting the interaction's source entity deviates from the true value, there is a state inconsistency. Any inconsistency in the position data is called spatial error. This error represents the difference between e_i's perception of e_j's position and e_j's true position.

More formally, and following the definitions in [11], let the *real path*, $\mathcal{R}(t)$, represent the true position of e_j at any time t. $\mathcal{R}(t)$ is a function of the user input and physics models employed by the simulation node n_j hosting e_j. At intervals, n_j sends a message updating the replica of e_j at the source node n_i. These state updates coupled with any dead reckoning algorithms in use at n_i yield the *placed path*, $\mathcal{P}(t)$. This path represents e_i's perception of e_j's position at any time t. The difference between the real and placed paths

$$E(t) = \mathcal{R}(t) - \mathcal{P}(t) \tag{2}$$

is the spatial error at time t.

Moreover, the position of the entity on the placed path after Δt seconds can be approximated using Taylor series expansion [12],

$$\mathcal{R}(t+\Delta t) \approx \mathcal{R}(t) + \frac{\mathrm{d}\mathcal{R}}{\mathrm{d}t}\Delta t + \frac{1}{2}\frac{\mathrm{d}^2\mathcal{R}}{\mathrm{d}t^2}\Delta t^2$$

Letting $s(t) = \mathcal{R}(t)$, $v(t) = \frac{\mathrm{d}\mathcal{R}}{\mathrm{d}t}$, and $a(t) = \frac{\mathrm{d}^2\mathcal{R}}{\mathrm{d}t^2}$ yields the familiar kinematic equation

$$\mathcal{R}(t+\Delta t) \approx s(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$$

where $s(t)$, $v(t)$, and $a(t)$ represent the entity's position, velocity, and acceleration at time t.

The placed path at time $t + \Delta t$ depends on the dead reckoning algorithm in use at the replica node. If no dead reckoning is employed, then

$$\mathcal{P}(t+\Delta t) = \mathcal{P}(t) = s(t)$$

If first-order dead reckoning is employed, then

$$\mathcal{P}(t+\Delta t) = s(t) + v(t)\Delta t$$

Combining the expressions for the real and placed paths yields the following expressions for the spatial error at time $t + \Delta t$:

$$E(t+\Delta t) = v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{3}$$

in the case of no dead reckoning and

$$E(t+\Delta t) = \frac{1}{2}a(t)\Delta t^2 \tag{4}$$

in the case of first-order dead reckoning.

If the entity's maximum velocity and acceleration are known (a safe assumption) and t_{update} is the time the last update was sent, then the spatial error after receiving the update can be bounded above as

$$E(t_{update}+\Delta t) \leq v_{max}\Delta t + \frac{1}{2}a_{max}\Delta t^2 \tag{5}$$

and

$$E(t_{update}+\Delta t) \leq \frac{1}{2}a_{max}\Delta t^2 \tag{6}$$

for the no dead reckoning and first-order dead reckoning cases respectively.

IV. Deriving Synchronization Parameters

Combining the notion of an interaction with the foregoing error model provides a straightforward means of deriving synchronization protocol parameters from high-level engineering requirements. Recall that an interaction is described by

$$I = (e_i, e_j, l)$$

where l specifies the maximum error in e_j's position that yields a correct result for the interaction. Thus, for a simulation employing dead reckoning, we can compute the temporal accuracy interval [13] for a state update in the context of interaction I as

$$\Delta t = \sqrt{\frac{2 * l}{||a_{max}||}} \tag{7}$$

Assuming state updates are sent from e_j to e_i periodically, the quantity Δt specifies the maximum time for which a state update is accurate. After Δt seconds, the update will have been superseded by another and the spatial error will have exceeded the tolerance of interaction I. Note that periodic state updates have been shown to result in optimal state consistency [14]. Thus, our concern is to find the largest interupdate period p that meets the error tolerance for interaction I.

To do so, we note that the temporal accuracy interval is the difference between the time the last update was sent, t_{update} and the time of its use, t_{use}. That is,

$$\Delta t = t_{use} - t_{update}$$

Noting that the current value of the state becomes invalid when the next update is sent and letting d represent the total update propagation delay (accounting for network latency, queuing, and internal delays associated with simulation architecture), the largest accuracy interval is obtained when

$$\Delta t = t_{update} + p + d - t_{update} \tag{8}$$

To summarize, let e_i be an entity interacting with entity e_j. Suppose the interaction has some maximum spatial error tolerance l. Let $||a_{max}||$ be the maximum acceleration of e_j and d be the state update propagation delay. Then the maximum period between state updates from e_j to e_i is given by

$$p = \sqrt{\frac{2 * l}{||a_{max}||}} - d \tag{9}$$

V. Distribution of p

Guarantees on system latency require establishing a messaging policy such that

$$\frac{1}{2}||a_{max}||(p+d)^2 \leq l, \tag{10}$$

where a is the acceleration, p is the inter-update period, d is the delay, and l is the plausibility limit. Assuming ideal system design involves maximizing the inter-update period to minimize the number of messages sent, this research focuses on the plausibility limit in Equation 10 being exactly met, not exceeded, or

$$\frac{1}{2}||a_{max}||(p+d)^2 = l. \tag{11}$$

Solving for p gives results in Equation 9, which is effectively the largest value of p that meets the plausibility limit for specific values of a_{max} and d.

Within these equations, a_{max} and d are independent random variables and l is a constant set by the end users. As independent random variables, the joint distribution of a_{max} and d is the product of their individual distributions, or

$$f_{A,D}(a,d) = f_A(a) f_D(d). \tag{12}$$

In this example problem, $A_{max} \sim$ Double Exponential(0,σ_1^2) and $D \sim$ Shifted Exponential(μ, σ_2^2), so

$$f_{A,D}(a,d) = \frac{1}{2\sigma_1} e^{-\frac{|a|}{\sigma_1}} \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}}. \tag{13}$$

Since p is a function of a_{max}, d, and l from Equation 9, a transformation can be performed on $f_{A,D}(a,d)$ into $f_{P,D}(p,d)$. Let

$$g(p) = p = \sqrt{\frac{2l}{a}} - d, \tag{14}$$

from Equation 9, then

$$g^{-1}(p) = a = \frac{2l}{(p+d)^2}. \tag{15}$$

Using a univariate transformation,

$$f_{P,D}(p,d) = f_{A,D}(g^{-1}(p),d) \left| \frac{\partial a}{\partial p} \right|.$$

Due to the change in monotonicity at $a = 0$, this transformation is actually the sum of two transformations

$$\begin{aligned} f_{P,D}(p,d) &= f_{A,D}(|g^{-1}(p)|,d) \left| \frac{\partial a}{\partial p} \right|, a < 0 \\ &+ f_{A,D}(g^{-1}(p),d) \left| \frac{\partial a}{\partial p} \right|, a \geq 0 \\ &= \frac{1}{2\sigma_1} e^{-\frac{\left|\frac{2l}{(p+d)^2}\right|}{\sigma_1}} \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \left| \frac{\partial}{\partial p} \frac{2l}{(p+d)^2} \right| + \\ &\frac{1}{2\sigma_1} e^{-\frac{\frac{2l}{(p+d)^2}}{\sigma_1}} \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \left| \frac{\partial}{\partial p} \frac{2l}{(p+d)^2} \right| \\ &= \frac{1}{\sigma_1} e^{-\frac{\frac{2l}{(p+d)^2}}{\sigma_1}} \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \frac{4l}{(p+d)^3}. \end{aligned} \tag{16}$$

Integrating over all values of d provides the marginal distribution of p

$$\begin{aligned} f_P(p) &= \int_0^\infty \frac{1}{\sigma_1} e^{-\frac{\frac{2l}{(p+d)^2}}{\sigma_1}} \\ &* \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \frac{4l}{(p+d)^3} \partial d \end{aligned} \tag{17}$$

Recall, p was crafted to be the maximum inter-update period that still meets the plausibility limit. Integrating from 0 to a particular value of p provides the probability that p exceeds the required plausibility limit, or

$$\begin{aligned} \alpha &= \int_0^p \int_0^\infty \frac{1}{\sigma_1} e^{-\frac{\frac{2l}{(p+d)^2}}{\sigma_1}} \\ &* \frac{1}{\sigma_2} e^{-\frac{d-\mu}{\sigma_2}} \frac{4l}{(p+d)^3} \partial d \partial x \end{aligned} \tag{18}$$

Rather than performing a Monte Carlo simulation to find the probability that a given p will meet the plausibility limit, Eqs. 17 and 18 can be computed directly.

VI. Example Application

As an example, we computed the plausibility exceedence probability for World of Warcraft entity position data. Sample data was obtained from the the Game Trace Archive at Delft University of Technology [16]. This data consists of entity position traces sampled at 1 second intervals over a period of 24 hours. Velocity and acceleration data were derived from the position traces and an exponential distribution was fit to the acceleration via maximum likelihood estimation ($\hat{\lambda} = 1.58$). Fig.1 plots a histogram of the entity accelerations and the fitted distribution.

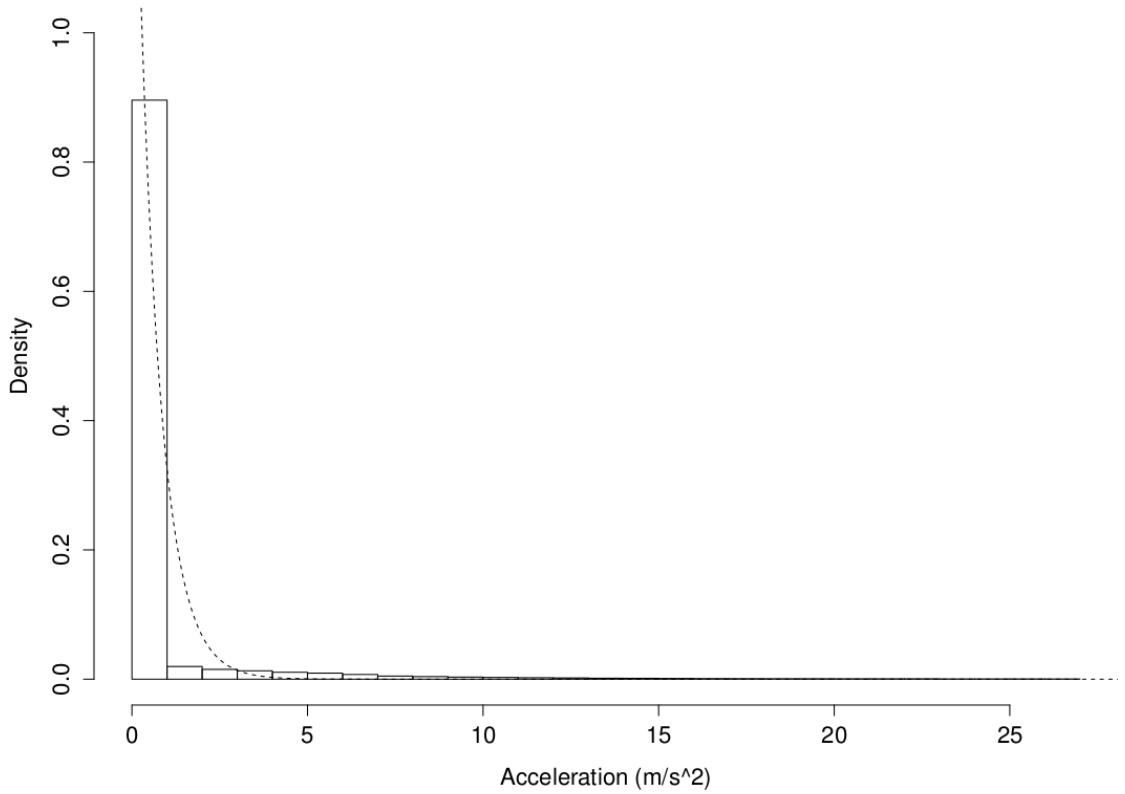


Fig. 1: Probability density of acceleration data sampled from World of Warcraft motion traces [15]. The dotted line is a maximum likelihood fit of an exponential distribution with $\hat{\lambda} = 1.58$.

Exceedence probability curves were computed using the acceleration distribution fit and a shifted exponential delay distribution with a nominal delay of 0.035 ms and $\lambda = 0.042$ ms. The delay parameters are based on Verizon's network performance data for cross-country links within the United States. Plausibility limits were set at $l = 10, 1.0$ and 0.1 meters. Exceedence probabilities were calculated for interupdate periods of 0 to 10 seconds. The resulting curves are plotted in Fig.2.

There are several interesting results apparent in this plot. First, interupdate periods for interactions that require loose plausibility limits can be quite large before there is a significant probability of exceeding the limit. Moreover, there is a substantial region of the curve for which the exceedence probability is effectively zero. Second, the likelihood of exceeding the plausibility limit of an interaction rapidly increases as the limits become smaller. Finally, for interactions with small plausibility limits (less than 0.1 meters), there is always some chance of exceeding the limit. This can be problematic for LVC test events requiring high precision results.

VII. Conclusion

In this paper, we have presented a model of error in the positional state variables of LVC simulations based on dead-reckoning and the characteristics of real-time sensor systems, introduced the notion of plausibility limits for entity interactions, and derived a means of computing the probability that a given inter-update period meets a particular plausibility limit. Taken together, these contributions provide a means of probabilistically bounding the error associated with entity motion and consequently estimating the quality of derived data – a key challenge for experimentation in LVC environments [7].

Acknowledgments

This research was supported by the Office of the Secretary of Defense, Operational Test and Evaluation (OSD DOT&E) and the Test Resource Management Center (TRMC) with the Science of Test Research Consortium.

Disclaimer

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

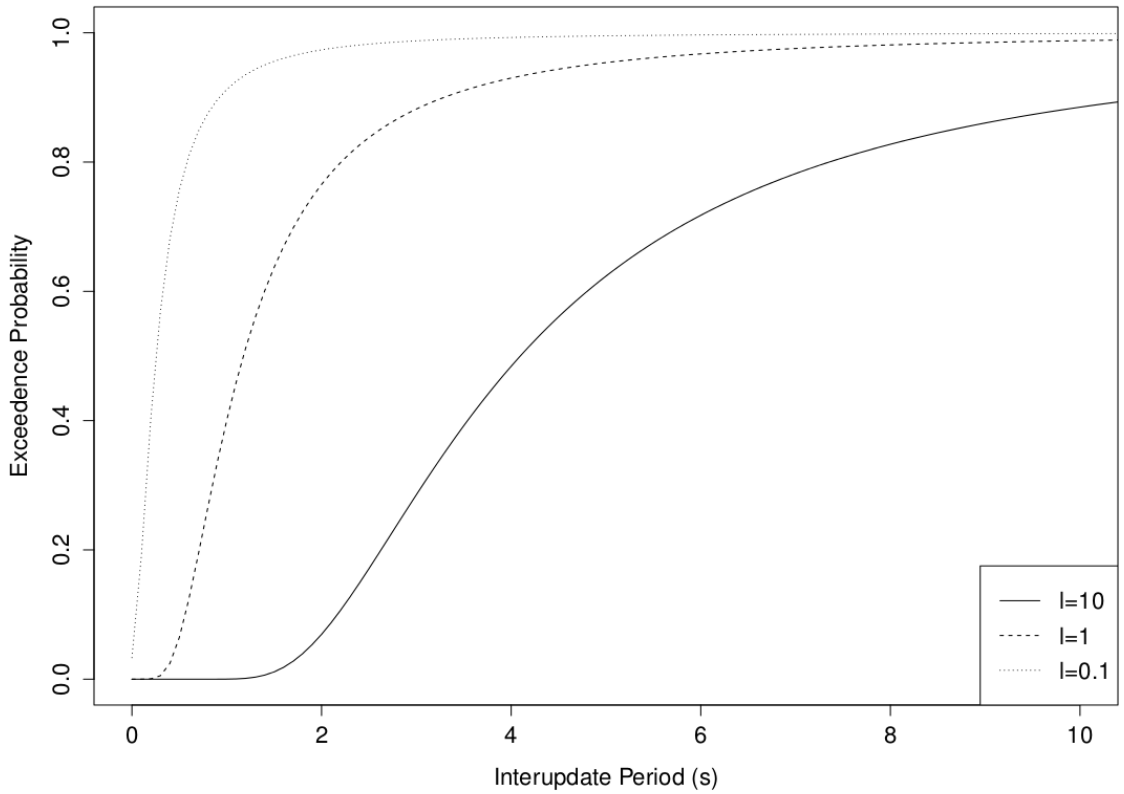


Fig. 2: Exceedence probabilities for plausibility limits of $l = 10, 1.0$ and 0.1 with exponential acceleration and delay distributions ($\hat{\lambda} = 1.58$ and $\hat{\lambda} = 0.7$ respectively).

REFERENCES

[1] E. P. Parker, N. E. Miner, B. P. Van Leeuwen, and J. B. Rigdon, "Testing unmanned autonomous system communications in a live/virtual/constructive environment," *International Test and Evaluation Association Journal (ITEA)*, vol. 30, pp. 513–522, 2009.

[2] B. Van Leeuwen, V. Urias, J. Eldridge, C. Villamarin, and R. Olsberg, "Performing cyber security analysis using a live, virtual, and constructive (lvc) testbed," in *Military Communications Conference, 2010-MILCOM 2010*. IEEE, 2010, pp. 1806–1811.

[3] D. D. Hodson and R. R. Hill, "The art and science of live, virtual, and constructive simulation for test and analysis," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, vol. 11, no. 2, pp. 77–89, 2014.

[4] S. Singhal and M. Zyda, *Networked virtual environments: design and implementation*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999.

[5] D. D. Hodson and R. O. Baldwin, "Performance analysis of live-virtual-constructive and distributed virtual simulations: defining requirements in terms of temporal consistency," 2009.

[6] DIS Steering Committee, "IEEE standard for distributed interactive simulation-application protocols," *IEEE Standard 1278*, 1998.

[7] J. R. Millar, D. D. Hodson, G. L. Peterson, and D. K. Ahner, "Data quality challenges in distributed live-virtual-constructive test environments," *J. Data and Information Quality*, vol. 7, no. 1-2, pp. 2:1–2:3, Apr. 2016. [Online]. Available: http://doi.acm.org/10.1145/2850420

[8] L. Itzel, R. Suselbeck, G. Schiele, and C. Becker, "Specifying consistency requirements for massively multi-user virtual environments," in *IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE)*. IEEE, 2011, pp. 1–2.

[9] M. Claypool and K. Claypool, "Latency can kill: precision and deadline in online games," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*. ACM, 2010, pp. 215–222.

[10] K. L. Morse *et al.*, *Interest management in large-scale distributed simulations*. Information and Computer Science, University of California, Irvine, 1996.

[11] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multi-player games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 2004, pp. 161–165.

[12] D. Hanawa and T. Yonekura, "On the error modeling of dead reckoned data in a distributed virtual environment," in *International Conference on Cyberworlds*. IEEE, 2005, pp. 8–pp.

[13] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*. Springer, 2011.

[14] X. Tang and S. Zhou, "Update scheduling for improving consistency in distributed virtual environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 6, pp. 765–777, 2010.

[15] S. Shen, N. Brouwers, A. Iosup, and D. Epema, "Characterization of human mobility in networked virtual environments," in *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*. ACM, 2014, p. 13.

[16] Y. Guo and A. Iosup, "The game trace archive," in *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, ser. NetGames '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 4:1–4:6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2501560.2501566

Proceedings of the 2016 Winter Simulation Conference
T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka and S.E. Chick, eds.

Sources of Unresolvable Uncertainties in Weakly Predictive Distributed Virtual Environments

Jeremy R. Millar
Communications and Information Directorate
Air Force Institute of Technology
2950 Hobson Way
Wright Patterson AFB, OH 45433

Jason A. Blake
Simulation and Analysis Facility
Air Force Life Cycle Management Center
Bldg. 802, 2302 8th St., Area B
Wright Patterson AFB, OH 45433

Douglas D. Hodson
Department of Electrical and Computer Engineering
Air Force Institute of Technology
2950 Hobson Way
Wright Patterson AFB, OH 45433

J.O. Miller
Raymond R. Hill
Department of Operational Sciences
Air Force Institute of Technology
2950 Hobson Way
Wright Patterson AFB, OH 45433

ABSTRACT

This work expands the notion of unresolvable uncertainties due to modeling issues in weakly predictive simulations to include unique implementation induced sources that originate from fundamental trade-offs associated with distributed virtual environments. We consider these trade-offs in terms of the Consistency, Availability, and Partition tolerance (CAP) theorem to abstract away technical implementation details. Doing so illuminates systemic properties of weakly predictive simulations, including their ability to produce plausible responses. The plausibility property in particular is related to fairness concerns in distributed gaming and other interactive environments.

1 INTRODUCTION

This paper considers two particular kinds of uncertainties that arise in distributed virtual environments and their relationship to weakly predictive simulation systems. We provide a review of uncertainties arising from insufficient knowledge about a system being modeled along with uncertainties arising due to the infrastructure of the experimental apparatus.

We consider the implementation-oriented aspects of distributed architectures from the perspective of the Consistency, Availability and Partition tolerance (CAP) theorem (Redmond and Wilson 2012) to abstract away details. Because the CAP theorem makes a strong statement concerning fundamental trades between three orthogonal aspects of any distributed system that manages a repository of data, it provides a means to reason about the quality of that data as a source from which an analysis often begins. Finally, we conclude with some observations concerning the use of weakly predictive simulated systems to support analysis-focused experiments.

This paper is organized as follows. Section 2 presents a review of uncertainty quantification and sources of uncertainty for computer-based experiment. Modeling human behavior in the presence of insufficient knowledge is the subject of Section 3 which illuminates the motivation to include humans within the simulation environment. Section 4 reviews how unresolvable uncertainties associated with modeling concerns fit within a topology of logical uses for simulation; of particular interest is the 'plausible outcomes' branch. Section 5 reviews fundamental tradeoffs made to create a highly interactive, responsive,

distributed virtual environments and their associated issues with data consistency. We next characterize these issues as a CAP theorem problem. Section 6 highlights the concept of plausibility in relation to dynamic state space consistency issues. We conclude in Section 7 by organizing the two major sources of uncertainty (modeling and architecture) as two sub-branches to 'plausible outcomes' that lead to weakly predictive simulations.

2 UNCERTAINTY QUANTIFICATION

Uncertainty quantification (UQ) is the science of quantitative characterization and reduction of uncertainties in both computational and real world applications. UQ identifies and categorizes different sources of uncertainty with domains of interest. A casual literature search for areas of categorization and quantification of uncertainty reveals active work in finance, economics, manufacturing and even climate change.

The role of simulation is often the identification and modeling of uncertainty to understand its impact or effect on some aspect of system performance or operation. For the domain of computer-based experimentation, (Kennedy and O'Hagan 2001) categorized sources that have grown into a more exhaustive list that can be found here (Wikipedia 2016).

- *Parameter uncertainty*, which comes from the model parameters that are inputs to the computer model (mathematical model) but whose exact values are unknown to experimentalists and cannot be controlled in physical experiments, or whose values cannot be exactly inferred by statistical methods. Examples are the local free-fall acceleration in a falling object experiment, various material properties in a finite element analysis for engineering, and multiplier uncertainty in the context of macroeconomic policy optimization.
- *Parametric variability*, which comes from the variability of input variables of the model. For example, the dimensions of a work piece in a process of manufacture may not be exactly as designed and instructed, which would cause variability in its performance.
- *Structural uncertainty*, aka model inadequacy, model bias, or model discrepancy, which comes from the lack of knowledge of the underlying true physics. It depends on how accurately a mathematical model describes the true system for a real-life situation, considering the fact that models are almost always only approximations to reality. One example is when modeling the process of a falling object using the free-fall model; the model itself is inaccurate since there always exists air friction. In this case, even if there is no unknown parameter in the model, a discrepancy is still expected between the model and true physics.
- *Algorithmic uncertainty*, aka numerical uncertainty, which comes from numerical errors and numerical approximations per implementation of the computer model. Most models are too complicated to solve exactly.
- *Experimental uncertainty*, aka observation error, which comes from the variability of experimental measurements. The experimental uncertainty is inevitable and can be noticed by repeating a measurement for many times using exactly the same settings for all inputs/variables.
- *Interpolation uncertainty*, which comes from a lack of available data collected from computer model simulations and/or experimental measurements. For other input settings that do not have simulation data or experimental measurements, one must interpolate or extrapolate in order to predict the corresponding responses.

Structural and interpolation uncertainty are of particular interest in relation to this work. We consider the modeling of human behavior as analogous to the lack of understanding of the 'underlying true physics' of the system. In other words, we don't completely understand how humans make decisions; because of that, modeling them is an issue.

Uncertainties that arise in distributed virtual environments closely mirror interpolation issues. For our purpose, interpolation has less to do about modeling concerns, but is driven by fundamental trades associated

with the simulation infrastructure that provides the experimental apparatus to conduct an experiment. We contend that both sources of uncertainty contribute to prediction outcomes that are classified as weak.

3 MODELING ISSUES

In modeling and simulation, abstraction is used to determine how details of a system are represented, while maintaining validity with respect to some purpose (Frantz 1995). Despite careful choices by the analyst to mitigate uncertainties, unresolvable uncertainties arise due to insufficient knowledge about a system being modeled (Bankes 1993). While these uncertainties are common across both distributed virtual environments and traditional self-contained discrete-event-based simulations; the implementation of a distributed virtual environment for analysis yields new sources of unresolvable uncertainty.

One particular modeling concern is representing humans and their behavior. Human behavior remains difficult to duplicate reliably; so in traditional simulations, modeling human behavior is often based on probabilistic draws from a defined or known distribution. This modeling approach is problematic when human behavior is not well understood, or if the purpose of the simulation experiment is to study that behavior.

Including humans within the simulated world is one of the key benefits of a distributed virtual environment. The goal of these systems is to create a shared sense of a represented world in which to interact. This leads to higher realism, as the human(s) are no longer models, they are real; but they risk adding noise to the system, especially if they are not clearly within the bounds of the system under study. As anyone with a family can attest, no two humans are exactly alike and even a reliable humans behavior will surprise you. This may be acceptable at family gatherings, but is an unfortunate occurrence for the analyst trying to draw conclusions from data collected from a distributed virtual environment used to conduct a study.

Introducing real humans into a simulation experiment presents challenges; if they are not directly part of the system under study, they can introduce unwanted noise, and if they don't know 'how they should act,' their very presence is considered to be an 'unresolvable uncertainity' or a modeling 'unknown.' This is a very real concern, especially if the system of interest is hypothetical - maybe representing the future or simply does not exist.

4 LOGICAL USES

In light of the fact that our simulations are not perfect reflections of reality, it must be determined how they might be useful. Considering only the types, sources and perceived magnitudes of uncertainty with a given simulation is not enough to make this determination. Only after considering the analytic purpose alongside simulation uncertainties can and educated opinion be formed regarding fitness for analytical use.

In (Dewar et al. 1996), a topology of logical uses for Distributed Interactive Simulation systems is presented (Figure 1). This topology divides the domain of logical uses into two subspaces; the first being experimental stimulus and the second as an analytical aid. The differentiator between these branches is defined by this question; for whose benefit is the study being executed (Dewar et al. 1996)? If a human-in-the-loop is the beneficiary, then the simulation is considered an experimental stimulus. Common examples of this type of simulation can be found in the training and education community where the purpose is to enhance three types of skills (i.e., motor, decision making, and operational skills) or entertainment (Topçu et al. 2016). Additional detail on simulation as experimental stimulus can be found in (Dewar et al. 1996).

Simulations where the beneficiaries are others beyond the participants in the simulation, then the system is categorized as an analytic aid (Dewar et al. 1996). This class of simulations will be familiar to most DoD analysts as this is where most constructive analysis tools fit within the topology. The key discriminator among the subclasses of analytic aids is the degree to which they are useful for prediction. Non-predictive uses of DIS do not require a high confidence fit between the simulation output and the real world, these

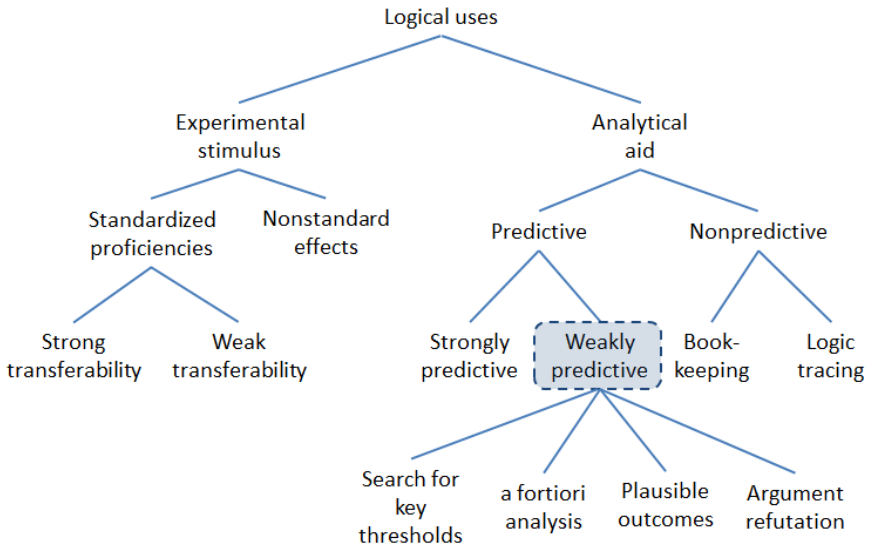


Figure 1: Topology of Logical Uses (Dewar et al. 1996)

uses include bookkeeping activities such as recording events and logic tracing. Predictive models place a higher requirement on the fit between simulation output and the real world system.

Strongly predictive models have a demonstrated capacity to forecast outcomes with a high degree of accuracy (Dewar et al. 1996). Weakly predictive models suffer from moderate to high levels of parametric, structural or unresolvable uncertainties, yet the model still describes some elements of the real system. This category is likely the most populated within the analytic aid class of simulations. Care should be taken to not overstate the predictive credibility of the analysis when using this type of simulation. Often the analyst is forced to use a weakly predictive model due to lack of data resulting from restrictions or nonavailability. Arguably, these system can play a supporting role in relation to experimentation, by generating hypotheses to investigate. In other words, use of weakly predictive simulations will not necessarily generate correct outputs, but might support a research strategy to understand and investigate the dynamics of a system.

Of particular interest is the 'plausible outcomes' branch in this topology, as it relates to unresolvable uncertainties due to modeling unknowns. One source of unknowns can be humans inserted into a virtual environment that really don't know 'how they should act or behave' to fairly represent a system of interest.

5 ARCHITECTURE ISSUES

It is fair to say that UQ is related to understanding known knowns and being tangentially aware of and accounting for, known unknowns. For a distributed virtual environment, this type of unknown or uncertainty arises from the software and/or hardware architecture that provides the experimental apparatus used to conduct an experiment. It is fair to say that we have known unknowns, but how those unknowns impact and influence outputs is not well understood.

Unresolvable uncertainties arise as a result of fundamental trades that must be made when the architecture of a distributed virtual environment is designed. Because these simulations by definition, include, interface and interact with humans and/or hardware, these systems are classified as real-time systems. Real-time systems define performance requirements in terms of timeliness; for example the time it takes to process new inputs to yield outputs is referred to as a response time. As an example, a human operator flying a simulated aircraft would expect (i.e., require) the system to respond to stick and throttle inputs by updating displays within a short period of time (e.g., 100ms).

This requirement is fundamentally at odds with the desire to maintain a consistent representation of the virtual world across all nodes in the distributed simulation. The challenge becomes problematic in

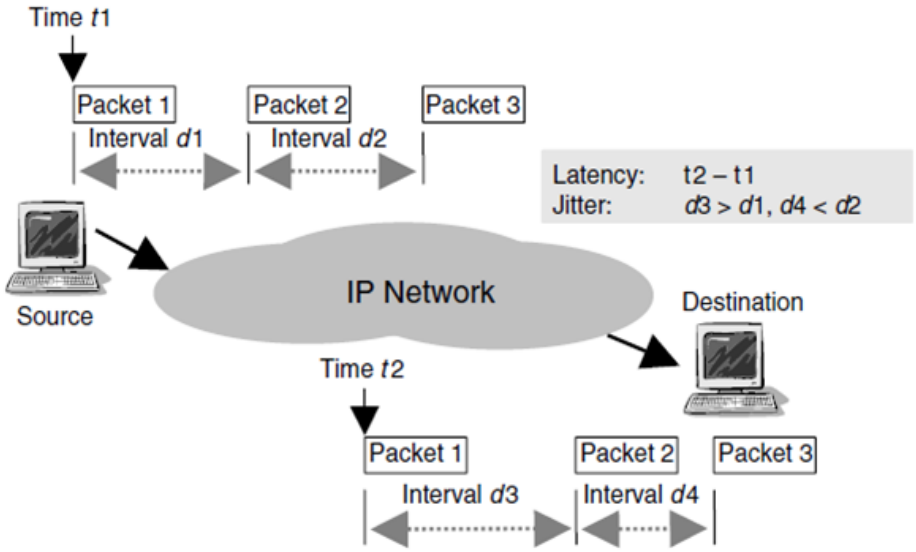


Figure 2: Latency and jitter affects (Armitage et al. 2006)

the presence of significant network latency and jitter. Latency is defined as the time required to deliver a network packet from a sender to receiver, and jitter measures the variation in that delivery time. As Figure 2 shows, latency and jitter affect the reliable delivery and timeliness of state data between nodes in the simulation. If the distributed nodes are connected via a network infrastructure with a relatively high latency, data being used by one simulation might be out of date or 'old' when compared to its current correct value as managed by another node. This inconsistency or error in state data is a distinguishing characteristic of distributed virtual environments and must be recognized and managed (Hodson and Baldwin 2009). The fundamental trade between state consistency and responsiveness is not new, it has existed since the advent of distributed interactive simulation; sometimes it is referred to as the consistency-throughput trade-off (Singhal and Zyda 1999). In the online gaming community, this issue manifests itself as an unfair game (e.g., a dead person shooting). How the concept of 'fairness' is defined, is an area of research, and relates to 'plausibility' or plausible responses which we discuss in Section 6.

If the purpose of the simulation is to serve as an analytical aid to support system study and understanding, then the system itself will most likely be derived from a conceptual model that defines what to represent. The conceptual model is mapped to entities which are created and managed by different simulation nodes (Hodson and Hill 2013). To illustrate this approach, Figure 3(a) presents a notional conceptual model that includes two interacting entities that exchange information related to an interaction. As shown in Figure 3(b), a possible architecture for the virtual environment might be two different simulations or simulators interconnected by a network. Given this arrangement, entity data created and locally managed will need to be shared using a network so that interactive environments can be created for each of the human participants. Figure 3(c) highlights the influence of network latency and jitter on the state data managed by each simulation. For this case, the state data associated with the location of remote entity(s) (i.e., dynamic shared state) most likely will not match their true position as managed by their local hosting application.

5.1 CAP Theorem

The complexities associated with the design and implementation a distributed virtual environment are well known. The underlying issues that create shared state inconsistencies are implementation specific and include the architecture of the software, such as single or multi-threaded, processing priorities, model execution/time advance assumptions, networking latency and jitter, model representation (fidelity), the

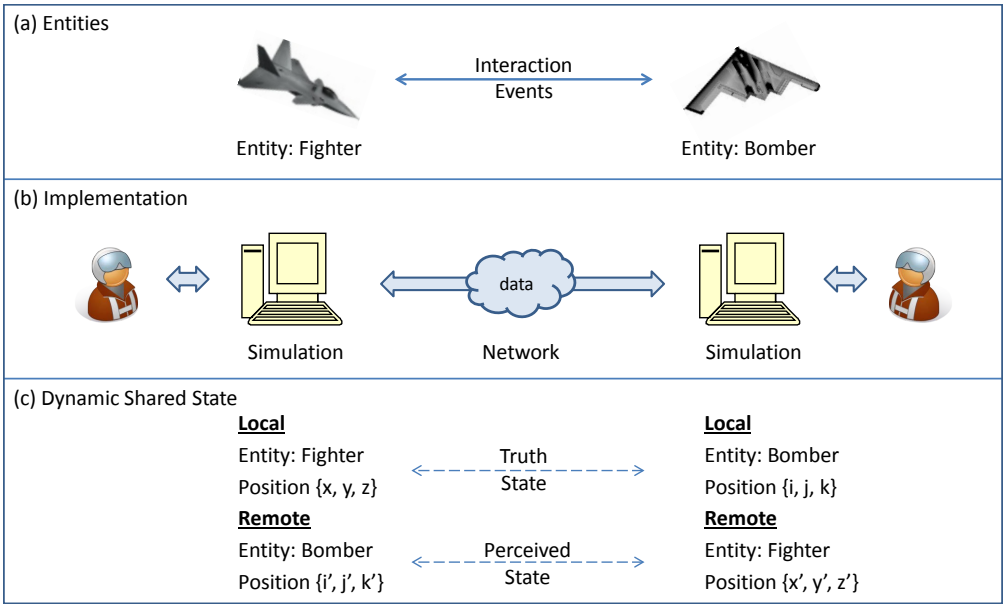


Figure 3: Entities, implementation, and dynamic shared state (Hodson and Hill 2013)

selection of various error threshold parameters, etc. To reason about these complexities, we leverage the CAP theorem - a useful abstraction from the domain of distributed databases.

The CAP theorem holds that you can create a distributed database that is *consistent* (writes are atomic and all subsequent requests retrieve the new value), *available* (the database will always return a value as long as a single server is running), or *partition tolerant* (the system will still function even if server communication is temporarily lost - that is, a network partition), but you can have only two at once (Redmond and Wilson 2012).

In other words, you can create a distributed database system that is consistent and partition tolerant, a system that is available and partition tolerant, or a system that is consistent and available (but not partition tolerant - which basically means not distributed). But it is not possible to create a distributed database that is consistent and available and partition tolerant at the same time (Redmond and Wilson 2012).

Our use of CAP hinges upon viewing the virtual environment in the same light as a distributed database system that attempts to keep its replicated data in sync (Millar et al. 2016). From this viewpoint, models are the consumers of state data that produce outputs. If inputs to the model(s) include dynamic shared state, then outputs may differ.

6 PLAUSIBLE RESPONSES

The CAP theorem has an important consequence for distributed virtual environments related to the plausibility of interaction outcomes as observed at the server and clients. Since virtual environments typically prioritize availability and partition tolerance above consistency, state divergence between the server and client is virtually guaranteed. In the main, this divergence is of little consequence. However, when the server and client must independently compute some result based on the same input variables at the same time, problems with the plausibility of outcomes can arise due to state divergence. For instance, if the interaction of interest is collision detection and the positional state of involved entities is not consistent, the server may detect a collision while the client does not. This problem is exacerbated when the server completes its outcome arbitration – the client's version of the state may well be overwritten by a completely different result, yielding an implausible outcome. Steed and Oliveira refer to this as joint plausibility, i.e., the notion that two or more users accept that they are viewing the same simulation of a shared space (Steed and Oliveira 2009).

Note that plausibility does not imply that the server and client compute identical interaction results. Rather, joint plausibility requires that the computed results be close enough that users are willing to accept the arbitrated results and not reject them out of hand as nonsensical. That is, dead men must not keep shooting (Mauve 2000) due to a divergence in states between client and server. Minimizing state inconsistency between nodes in a distributed virtual environment is a well-studied problem; Delaney *et al* provide an excellent overview of consistency maintenance algorithms (Delaney, Ward, and McLoone 2006b, Delaney, Ward, and McLoone 2006a).

The maximum tolerable state divergence resulting in plausible outcomes is interaction dependent (Itzel et al. 2010). None of the commonly employed consistency maintenance algorithms explicitly account for outcome plausibility, preferring instead to schedule state updates in a fashion designed to minimize inconsistency. However, minimal state inconsistency between nodes does not necessarily imply a plausible outcome.

7 CONCLUSIONS

The main contribution of this research is the unification of two well understood, but separately considered drivers that cause distributed virtual environments to produce at best plausible outcomes which lead to weak prediction. The first relates to modeling aspects of a system that are unknown, the second is a result of the simulation architecture or experimental apparatus used to perform the experiment. The first contributor to plausible outcomes might consistently yield the same agreed upon incorrect result; the second will most likely yield a range of results depending upon which node within the distributed system computes it.

ACKNOWLEDGMENTS

This research was supported by the Office of the Secretary of Defense, Operational Test and Evaluation (OSD DOT&E) and the Test Resource Management Center (TRMC) with the Science of Test Research Consortium.

DISCLAIMER

The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

REFERENCES

Armitage, G., M. Claypool, and P. Branch. 2006. *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. Wiley.

Bankes, S. 1993. "Exploratory Modeling for Policy Analysis". *Operations Research* 41 (3): 435–449.

Delaney, D., T. Ward, and S. McLoone. 2006a. "On consistency and network latency in distributed interactive applications: A survey-part II". *Presence: Teleoperators and Virtual Environments* 15 (4): 465–482.

Delaney, D., T. Ward, and S. McLoone. 2006b. "On consistency and network latency in distributed interactive applications: A surveyPart I". *Presence: Teleoperators and Virtual Environments* 15 (2): 218–234.

Dewar, J. A., S. C. Bankes, J. S. Hodges, T. Lucas, D. K. Saunders-Newton, and P. Vye. 1996. "Credible uses of the distributed interactive simulation (DIS) system". Technical report, Santa Monica, California.

Frantz, F. K. 1995. "A Taxonomy of Model Abstraction Techniques". In *Proceedings of the 27th Conference on Winter Simulation*, WSC '95, 1413–1420. Washington, DC, USA: IEEE Computer Society.

Hodson, D. D., and R. O. Baldwin. 2009. "Characterizing, Measuring, and Validating the Temporal Consistency of Live-Virtual-Constructive Environments". *Simulation: Transactions of The Society for Modeling and Simulation International* 85 (10): 671–682.

Hodson, D. D., and R. R. Hill. 2013. "The Art and Science of Live, Virtual, and Constructive Simulation for Test and Analysis". *Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 11:77–89.

Itzel, L., V. Tuttlies, G. Schiele, and C. Becker. 2010. "Consistency management for interactive peer-to-peer-based systems". In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, 1. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Kennedy, M. C., and A. O'Hagan. 2001. "Bayesian Calibration of Computer Models". *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 63 (3): 425–464.

Mauve, M. 2000. "How to keep a dead man from shooting". In *Interactive Distributed Multimedia Systems and Telecommunication Services*, 199–204. Springer.

Millar, J. R., D. D. Hodson, G. L. Peterson, and D. K. Ahner. 2016, April. "Data Quality Challenges in Distributed Live-Virtual-Constructive Test Environments". *Journal of Data and Information Quality* 7 (1-2): 2:1–2:3.

Redmond, E., and J. R. Wilson. 2012. *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. The Pragmatic Bookshelf.

Singhal, S., and M. Zyda. 1999. *Networked Virtual Environments: Design and Implementation*. Addison Wesley.

Steed, A., and M. F. Oliveira. 2009. *Networked Graphics: Building Networked Games and Virtual Environments*. Elsevier.

Topçu, O., U. Durak, H. Oğuztüzün, and L. Yilmaz. 2016. *Distributed Simulation: A Model Driven Engineering Approach*. Springer.

Wikipedia 2016. "Uncertainty quantification". [Online; accessed 28-April-2016].

AUTHOR BIOGRAPHIES

JEREMY R. MILLAR is a active-duty network operations officer in the United States Air Force. He is the Director, Communications and Information for the Air Force Institute of Technology (AFIT). He holds Bachelors and Masters degrees in Computer Science from the University of Tennessee and AFIT, respectively. He is currently completing a Ph.D. in Computer Science at AFIT. His research interests lie in distributed systems, virtual environments, and machine learning. Major Millar has served in a variety of Air Force assignments including deployments to Iraq, Afghanistan, and Guantanamo Bay, Cuba. His email address is jeremy.millar@afit.edu.

JASON A. BLAKE is a Systems Analysis Engineer for the U.S. Air Force Simulation and Analysis Facility (SIMAF) and a Ph.D. student in the Operations Research Department at the Air Force Institute of Technology. He has a B.S. in Industrial and Systems Engineering from The Ohio State University and a M.S. in Operations Research from the Air Force Institute of Technology. His research interests include combat modeling, distributed simulation and design and analysis of experiments. He can be reached at jason.blake.3@us.af.mil.

DOUGLAS D. HODSON is an Assistant Professor of Software Engineering with the Air Force Institute of Technology. He received a B.S. in Physics from Wright State University in 1985, and both an M.S. in Electro-Optics in 1987 and an M.B.A. in 1999 from the University of Dayton. He completed his Ph.D. at the Air Force Institute of Technology in 2009. He has over 25 years of experience in the domain of modeling and simulation and has a research interest in characterizing the consistency of shared simulation state data in terms of its temporal properties to estimate Live-Virtual-Constructive and Distributed Virtual Simulations performance, cloud computing and modeling quantum key distribution systems. He is the technical lead developer for the open-source OpenEaagles simulation framework that has been used to develop a wide variety of standalone and distributed simulation applications. He is also a DAGSI scholar

and a member of Tau Beta Pi. His email address is douglas.hodson@afit.edu.

J.O. MILLER is a 1980 graduate of the U.S. Air Force Academy (USAFA) and retired from the Air Force as a Lt. Colonel in January 2003. In addition to his undergraduate degree from USAFA, he received an MBA from the University of Missouri at Columbia in 1983, his M.S. in Operations Research from the Air Force Institute of Technology (AFIT) in 1987, and his Ph.D. in Industrial Engineering from The Ohio State University in 1997. He is an Associate Professor of Operations Research in the Department of Operational Sciences at AFIT. His research interests include combat modeling, computer simulation, and ranking and selection. He can be reached at john.miller@afit.edu.

RAYMOND R. HILL is a Professor of Operations Research with the Air Force Institute of Technology. He has a Ph.D. in Industrial and Systems Engineering from The Ohio State University. His research interests include applications of simulation, applied statistical modeling, mathematical modeling, decision analytical methods, the design and analysis of heuristic optimization methods, and agent-based modeling. He is an Associate Editor for Quality Engineering, Naval Research Logistics, Military Operations Research, the Journal of Defense Modeling and Simulation, and the Journal of Simulation. He was a proceedings co-chair for both the 2008 and 2009 Winter Simulation Conferences and was the General Chair for the 2013 conference. His email address is rayrhill@gmail.com.

CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 0000; **00**:1–18
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Optimizing Update Scheduling Parameters for Distributed Virtual Environments Supporting Operational Test

Jeremy R. Millar[1] and Douglas D. Hodson[2]*and Gilbert L. Peterson[2] and Darryl K. Ahner[3]

[1]*Air Force Institute of Technology, 2950 Hobson Way, Wright-Patterson AFB, Dayton OH, 45344*
[2]*Department of Electrical and Computer Engineering, Air Force Institute of Technology, 2950 Hobson Way, Wright-Patterson AFB, Dayton OH, 45344*
[3]*Department of Operational Science, Air Force Institute of Technology, 2950 Hobson Way, Wright-Patterson AFB, Dayton OH, 45344*

SUMMARY

Distributed virtual environments provide a shared sense of time and space to geographically distributed users. They depend on a limited ability to ensure system nodes at different locations maintain consistent state data, and to the extent the system cannot support this goal, suffer from a number of undesireable effects. This paper presents system models and algorithms designed to find optimal update scheduling parameters that minimize the effects of inconsistent simulation state data. The first model is concerned with ensuring distributed virtual environments present a fair experience to all users while simultaneously providing adequate levels of system performance. In the second model we introduce the concept of plausibility limits and address their use in ensuring all participants in an interaction see the same result. Copyright © 0000 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: optimization, distributed virtual environments, fairness, lvc simulation

1. INTRODUCTION

Distributed virtual environments (DVEs) are real-time, man-in-the-loop, geographically distributed simulations supporting interaction between users through shared senses of time and place [29]. Applications of distributed virtual environments are diverse: examples include entertainment and gaming [2, 1], distributed training [15], and analytical test and evaluation [27, 32, 18]. Maintaining a shared sense of time and place for these applications requires synchronizing the state of dynamic entities across system nodes. However, the inclusion of human actors imposes soft real-time constraints on the simulation's response to user inputs. Failure to meet these constraints often results in user dissatisfaction and disuse of the system [7, 4]. Moreover, distributed virtual environments must frequently tolerate the loss or addition of system nodes without failure or undue performance impact. Meeting responsiveness and partition tolerance requirements necessitates a relaxation of absolute state consistency [5].

The use of weakly consistent state in distributed virtual environments leads to a number of challenges, particularly in relation to the outcome of user interactions in the shared space. Since the state of the world (e.g., entity positions) may differ between nodes, each participant in an

*Correspondence to: Department of Electrical and Computer Engineering, Air Force Institute of Technology, 2950 Hobson Way, Wright-Patterson AFB, Dayton OH, 45344. E-mail: douglas.hodson@afit.edu

interaction may perceive different outcomes, manifesting as a sense of unfairness. For instance, a player engaged in combat with another may perceive a series of definite hits and a kill while his target perceives a series of misses and returns fire [22]. In analytical applications, weak state consistency contributes to uncertainty in data measurements and difficulty in quantifying error [24].

The Department of Defense is increasingly turning to virtual environments to support operational test and evaluation due to their ability to provide access to threat densities and assets that are otherwise unavailable, e.g., foreign equipment or low-density, high-demand assets such as stealth aircraft. In this domain, ensuring the plausibility of entity interaction outcomes is crucially important to ensure that valid conclusions are drawn from the test event. Additionally, spatial state data is of prime importance since nearly all aspects of the target operations chain (Find, Fix, Track, Target, Engage, and Assess) depend on accurate positional information. While there are far more state elements to a modern DVE than positional information, this data is of utmost importance to our application domain, e.g., live-virtual-constructive simulation and experimentation in a military context. This is because the outcome of nearly all relevant interactions (target detection and tracking, weapons effects, damage models, etc) depend on accurate positioning. Inaccuracies in positional state data can yield different outcomes for geographically distributed users potentially invalidating the training or test event. Finally, ensuring a "fair fight" is necessary to keep operators from unintentionally skewing experimental results.

There is little traceability from high-level simulation requirements to low-level synchronization parameters. Often, a simulation is constructed and the synchronization protocol is hand tuned until the system's performance appears correct – most likely by subjective means. In the best case, the state synchronization system meets or exceeds the consistency and responsiveness required to meet the simulation's goals. In the worst case, the (unspecified) consistency and responsiveness targets are not met, and the simulation may behave in subtle and incorrect ways.

Since consistency is generally relaxed in favor of responsiveness, these errors most often manifest as spatial errors wherein each simulation participant (or node) perceives the same entity at different locations. This breaks the fundamental illusion of a shared space. Designers of distributed virtual environments typically employ an optimistic consistency maintenance algorithm such as dead-reckoning [29] to minimize the effects of weakly consistent state. In addition, the state update scheduling algorithm must be carefully chosen and tuned to maximize state consistency without overloading network resources or missing real-time deadlines. This paper presents two algorithms designed to provide optimal update scheduling parameters.

The first algorithm is a multi-objective optimization for virtual environments employing dead-reckoning-based state updates. It provides the optimal dead-reckoning error threshold in terms of response time, state consistency, and fairness. Here, fairness is defined in terms of a cluster cohesion metric that ensures all simulation users see similar levels of state consistency and response time. This work was originally presented in [25].

The second algorithm extends the previous work to virtual environments employing periodic state updates and is based on the notion of interaction contexts [19] and plausibility limits. We introduce the notion of plausibility limits as the maximum tolerable state inconsistency that allows all participants in an interaction to reach the same conclusion or outcome. Using object position as the state of interest, a statistical model of consistency is derived in terms of update period and spatial error. Solution of the associated optimization problem yields the largest period between updates meeting the plausibility limit with a specified level of confidence.

Given some mild assumptions about network latency and user behavior, these algorithms provide designers of distributed virtual environments a means to estimate system performance *a priori* to ensure fitness for purpose, e.g., training or test and evaluation. For applications involving test and analysis, the level of uncertainty associated with system state can be quantified and tuned to ensure measurements are within tolerance.

DOI: 10.1002/cpe

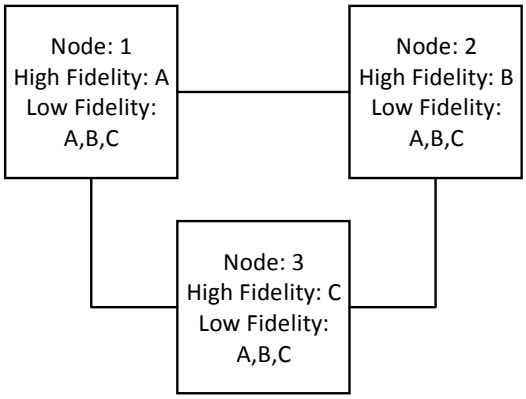


Figure 1. A virtual environment consisting of three nodes and three entities. Each node maintains an authoritative high fidelity model for its local entity and low fidelity models for all entities in the system.

2. OPTIMAL DEAD-RECKONING

IEEE Standard 1278, Distributed Interactive Simulation (DIS) [13], provides a common protocol and messaging standard for communicating between nodes in a virtual environment. While there are other interoperability standards available (e.g., HLA [11] or TENA [26]), DIS defines the data and semantics of operation. It is considered a de facto standard for defense-oriented virtual environments (i.e., networked simulators) and its consistency maintenance mechanisms are widely used in networked games [30]. Consequently, we restrict our attention to dead-reckoning algorithms as defined by the DIS standard.

The DIS standard defines a predictive consistency maintenance protocol called *dead-reckoning*. Under dead-reckoning, each node maintains a low-fidelity model for each remote entity in the system in addition to the high-fidelity models for its hosted entities. Figure 1 depicts a virtual environment consisting of three nodes and three entities. Each node provides an authoritative, high-fidelity model for one or more entities. Additionally, each node maintains a low-fidelity model of all other entities in the system. Crucially, each node also maintains a low-fidelity model of its own local entity.

The low-fidelity models allow a node to update entity positions between state updates using predictive dead-reckoning algorithms. Low-fidelity models typically operate using simplified dynamics such as first order kinematics. Note that all nodes execute the same dead-reckoning model. State updates are sent by a node whenever the divergence (i.e., difference) between the position of the high-fidelity and low-fidelity models of its hosted entities exceeds a pre-determined threshold. This threshold is the key parameter controlling the dead-reckoning algorithm.

Choosing an appropriate error threshold is a system-dependent design decision. Generally speaking, lower thresholds yield better consistency. However, improved consistency comes at the cost of increased network traffic. Depending on network characteristics such as available bandwidth, it is possible to overwhelm the network and increase system response time (that is, the time it takes for all nodes to see an update). Additionally, as network load increases, consistency can actually decrease as well [21].

2.1. Fairness

Consistency and response time are local properties, that is, they are measured pair-wise. Thus the consistency measured between nodes 1 and 2 with respect to entity A in Figure 1 might well be different than that measured between nodes 2 and 3. Global properties are also of interest, particularly the notion of fairness [6, 8]. A system is fair if no user has an advantage over others due to consistency or response time.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

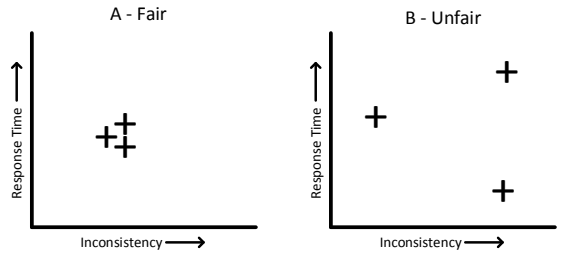


Figure 2. Fairness plots for two three node virtual environments. Each + symbol represents a node's location in two-dimensional fairness space. The nodes in system A are tightly clustered with similar consistency and responsiveness values. Therefore, system A is fair. Conversely, system B is unfair since the nodes are widely dispersed in fairness space.

Fairness is measured by projecting each node participating in a virtual environment into a two-dimensional fairness space with consistency as one dimension and system response time as the other. A cluster cohesion measure such as within-class scatter is computed for all nodes. Low scatter indicates that the nodes are tightly clustered in fairness space. Thus, each node has similar consistency levels and response times and no participant experiences a significant advantage or handicap. On the other hand, a high scatter value indicates that nodes have dissimilar consistency values and response times. This affords some participants advantages in terms of state consistency or response while handicapping others.

Figure 2 illustrates these ideas for two three-node virtual environments labeled system A and system B. Note that the horizontal axis measures *inconsistency* so that consistency (i.e., difference between states) degrades as one moves away from the origin. The vertical axis measures system response time, i.e., the time required for a state update from one node to reach all other nodes. For both dimensions lower values (closer to the origin) are more desirable. Each '+' symbol indicates an individual node's position in fairness space based on average consistency level and response time.

For system A, the nodes are clustered fairly tightly, indicating a fair system. Each participating node has a similar consistency level and response time. Thus no participant has a distinct advantage in terms of better information about the environment or more rapid environmental response. Conversely, the nodes in system B are widely dispersed in fairness space. This indicates an unfair system. One node has a distinct advantage in terms of data consistency, one has an advantage in response time, and one is severely handicapped in both dimensions.

A system can be fair while exhibiting poor performance with respect to data consistency or response time. Similarly, a system with generally good performance can be unfair so long as at least one node has sufficiently different performance characteristics. Consequently, it is incumbent upon system designers to consider fairness in addition to the more traditional trade-offs between consistency and responsiveness.

2.2. Multi-Objective Model

In order to optimize the dead-reckoning error threshold, we need to define and compute the following quantities:

1. average inconsistency,
2. average response time,
3. and fairness.

2.2.1. Computing Inconsistency The virtual environment community has settled on two major inconsistency measures:

1. spatial inconsistency (variously termed spatial error, export error, etc),
2. and time-space inconsistency [35],

DOI: 10.1002/cpe

Spatial inconsistency is simply the difference between the local, dead-reckoning estimate of an entity's position and its true, high-fidelity position. Time-space inconsistency is the spatial inconsistency integrated over a time period to account for the fact that even small errors can be meaningful if they last long enough. Of the two, spatial inconsistency is the more common, largely because it is simple to compute. Additionally, choosing thresholds for time-space inconsistency can be non-intuitive since the value no longer corresponds to a simple error. For these reasons, we consider spatial inconsistency as our measure of interest for this research. However, the optimization techniques employed here are applicable regardless of the specific inconsistency measure. Indeed, they may well make time-space inconsistency more attractive by eliminating manual input of the threshold value.

We compute the average spatial inconsistency as follows: let $P_i(t)$ be the true position of entity i at time t. Let $P_i^j(t)$ be the position of entity i as represented by node j at time t according to its dead-reckoning model. Then the average (pairwise) inconsistency with respect to entity i at node j is given by

$$\frac{1}{T}\sum_{t=1}^{T}|P_i(t) - P_i^j(t)| \tag{1}$$

Averaging Equation 1 over all entities for a particular node j gives the average spatial inconsistency experienced by node j, i.e.,

$$\frac{1}{N}\sum_{i=1,i\neq j}^{N}\frac{1}{T}\sum_{t=1}^{T}|P_i(t) - P_i^j(t)| \tag{2}$$

Computing Equation 2 for all nodes and averaging provides the average global system inconsistency associated with remote entity positions, i.e.,

$$\frac{1}{N^2T}\sum_{j=1}^{N}\sum_{i=1,i\neq j}^{N}\sum_{t=1}^{T}|P_i(t) - P_i^j(t)| \tag{3}$$

We desire to minimize this inconsistency measure.

2.2.2. Computing Response Time Local response time is associated with the time it takes to process user inputs. We consider the response time associated with propagating state updates from a given node to all other nodes in the system. In order to account for queuing effects in the implemented software system itself, as well as all network-induced latencies, this value should be measured in an end-to-end fashion. That is, the clock begins when the sending application executes the send operation and not when the operating system and network hardware actually place the bits on the wire. Similarly, it ends when the receiving application (not host or operating system) has received the data.

Response time can be calculated as follows: let t_{ij} be the amount of time required to send an update from node i to node j. Then the response time is given by

$$\max_j t_{ij}, j = 1 \ldots N, j \neq i \tag{4}$$

where N is the total number of nodes in the system. Note that this value may vary with time since it depends on environmental factors such as network load. For simplicity, we assume this value is constant.

Averaging Equation 4 over all nodes provides a measure of system response time, i.e.,

$$\frac{1}{N}\sum_{i=1}^{N}\max_j t_{ij}, j = 1 \ldots N, i \neq j \tag{5}$$

We seek to minimize this response time.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

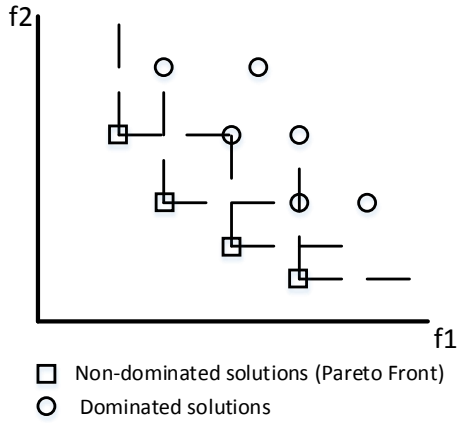


Figure 3. Pareto front, dominated solutions, and non-dominated solutions for a bi-objective minimization problem.

2.2.3. Computing Fairness Equations 2 and 4 provide a means of locating each node in a two-dimensional fairness space. System fairness is computed as the cohesion of the resulting data cluster. Let the vector f_i be the location in fairness space of node i. Then the system fairness is given by

$$\sum_{i=1}^{N}(f_i - c)^2 \tag{6}$$

where c is the centroid of the N fairness locations f_i. Minimizing this value corresponds to a tighter grouping in fairness space.

2.2.4. Multi-Objective Error Threshold Problem We are now in a position to define selection of the dead-reckoning error threshold as a multi-objective optimization problem. Let x be the spatial error threshold. Let $\vec{f} = (f_1 f_2 f_3)$, where f_1 is given by Equation 3, f_2 is given by Equation 5, and f_3 is given by Equation 6. Then we wish to find

$$\min_x f(\vec{x}) \text{ s.t. } BW - BW_{max} \leq 0 \tag{7}$$

where BW is the system bandwidth requirement based on the number of state update messages sent and BW_{max} is the system's maximum available bandwidth.

2.3. Solving the Multi-Objective Error Threshold Problem

In general, there is not a single solution to the multi-objective optimization problem defined by Equation 7. Instead, a set of solutions characterizing the trade-offs between individual objectives is obtained. This notion is formalized through the concepts of Pareto dominance and Pareto optimality.

Definition 1 (Pareto Dominance)
Without loss of generality, assume a multi-objective minimization problem. A solution x dominates solution y if $f_i(x) \leq f_i(y)\ \forall i$ and $\exists j$ such that $f_j(x) < f_j(y)$. Pareto dominance is denoted $x \preceq y$.

Definition 2 (Pareto Optimality)
A solution x is Pareto optimal if $\neg\exists\ x' \preceq x$; that is, if no other solution dominates x.

A set of Pareto optimal solutions is called a Pareto optimal set and its image in objective space is called the Pareto front. In solving multi-objective optimization problems, we seek to find or approximate the Pareto optimal set and its associated trade-offs represented by the Pareto front.

DOI: 10.1002/cpe

Figure 3 presents these concepts graphically for a bi-objective minimization problem. Square dots represent non-dominated solutions on the Pareto front. Round dots represent dominated solutions. The dotted lines represent dominance areas – a solution denoted by a square dot dominates any solution above and to its right. Although not drawn, this relationship holds for solutions not on the Pareto front as well.

Solutions to multi-objective optimization problems should lie on or as close as possible to the Pareto Front. Additionally, solutions should cover a broad section of the Pareto front. Multi-objective evolutionary algorithms are a preferred means of solving multi-objective optimization problems because they can find multiple Pareto optimal solutions in a single run. Additionally, multi-objective evolutionary algorithms are able to handle concavity and discontinuity on the Pareto front [10] making them ideal for exploring the trade-off space. Implementation of a solver for the error threshold problem requires two fundamental subsystems: a multi-objective optimization routine, and a simulation of the virtual environment. We built the optimization portion of our solver on the JMetal [14] multi-objective optimization framework. JMetal is a Java-based framework providing abstractions for problems, algorithms, and experiments. It includes a large number of multi-objective optimization algorithms as well as standard benchmark problems. Additionally, JMetal provides an experimental framework capable of multiple independent runs and basic statistics gathering. We have extended JMetal with an implementation of the error threshold problem. This extension evaluates candidate solutions by invoking a virtual environment simulator, reading its output, and computing values for each objective function.

A simulation of the distributed virtual environment was developed using the OMNeT++ [33] discrete event simulation framework. The simulation is structured as a hierarchical set of interacting modules that communicate via timed messages defining the events in the system. Runtime libraries are provided to manage the simulation infrastructure (e.g., the future events list, event scheduling, etc). Extensions provide a variety of network nodes and protocols to assist developers.

For each evaluation, our solver generates a network description file describing the node types, network topology, and parameters to simulate. With the exception of the dead-reckoning error threshold to evaluate, the contents of this file are fixed. The simulator is invoked with the error threshold under consideration and run for a configurable number of time steps. It outputs trajectory and message log files for each entity in the virtual environment.

The trajectory log file for each entity includes its true position at each time step. It also includes, for each time step, the perceived location of all other entities in the system. Taken as a whole, this data allows us to reconstruct the true and perceived locations for all pairs of entities at all times.

The message log file for each entity records the start time for each message sent as well as the time each incoming message was received. Taking these data as a whole allows us to compute maximum response times for each state update.

2.4. Model Validation

Validation of the multi-objective approach to setting error thresholds requires a particular virtual environment for investigation. This environment should be deterministic with well-understood decision models and dynamics for each entity. Additionally, all entities should be simulated to allow for statistically significant numbers of trials and long simulations. Finally, the dynamics of each entity should depend on one or more other entities and should be complex enough to provide interesting data.

We leverage Reynolds' boids model of flocking behavior [28] to provide a simple system with complex enough dynamics to generate an interesting Pareto front. The model defines flocking behavior as an emergent system property based on individual behaviors. Entities called boids move through a virtual space in three dimensions much like a flock of birds flying. The behavior of each boid is highly coupled to all other boids as each seeks to align its motion with its neighbors, steer towards the center of its neighbors, and avoid collisions. These simple steering behaviors allow complex flocking to emerge without explicitly designing it into the system.

To investigate the shape of the error threshold Pareto front for the boids, a series of single factor experiments were undertaken. The goals of these experiments are to: 1) demonstrate the validity

DOI: 10.1002/cpe

Table I. Algorithm parameters.

Parameter	NSGA-II	SPEA2	MCTS
Population size	100	100	100
Archive size	100	100	100
Max evaluations	500	500	500
Crossover probability	0.9	0.9	-
Crossover distribution index	20.0	20.0	-
Mutation probability	1.0	1.0	-
Mutation distribution index	20.0	20.0	-
Exploration coefficient	-	-	$\frac{1}{\sqrt{2}}$

Table II. Wilcoxon rank-sum test results for Experiment 1.

	SPEA 2	MCTS
NSGAII	▲	▲
SPEA2		▲

of the multi-objective optimization approach to determining dead-reckoning error thresholds, 2) ascertain the shape and location of the Pareto front for a representative virtual environment, and 3) compare the performance of the NSGA-II [12], SPEA2 [36], and MCTS [34] multi-objective optimization algorithms on the error threshold problem.

Two experiments were run using slightly different configurations. In the first, a 3 node fully-connected boids network was established. 30 runs were made for each of the NSGA-II, SPEA2, and MCTS algorithms. The simulation ran for 60,000 steps for each candidate solution. All network parameters (e.g., propagation delay, jitter, etc) were fixed and homogeneous. This leads to a constant response time based solely on the network's propagation delay and makes Equation 5 irrelevant. Therefore, a simple count of messages sent was substituted for Equation 5 with a goal of minimizing total traffic. This is a reasonable thing to do as it serves as a surrogate or model for aggregate traffic which is associated with system scalability.

The second experiment used a 5 node fully-connected boids network with time-varying network characteristics. Each link was given a constant propagation delay of 500 ms and a fixed bandwidth of 1.5 Mbps. For each transmission, jitter was sampled from a truncated normal distribution with mean of 100 ms and variance of 60 ms. Link saturation was modeled with a simple queuing mechanism – messages are held until the link becomes available. Retransmits and dropped packets were not modeled. Response time was measured as the second objective. 10 runs were made for each algorithm with 60,000 steps per simulation invocation.

Table I lists the parameters used for each algorithm. Note that crossover, mutation, and selection operators refer to built-in operators provided by JMetal. The exploration coefficient for MCTS sets a trade-off between exploration of new tree branches and exploitation of known good branches. For multi-objective problems, there should be a coefficient for each objective. Since little was known *a priori* about the structure of the search space, all objectives use the same coefficient value.

Figure 4 plots the approximate Pareto fronts returned by NSGA-II, SPEA2, and MCTS in objective space for Experiment 1. All three algorithms achieve good convergence and diversity and show a distinct Pareto front. As expected, the best values cluster near the origin. There are well-defined trade-offs between inconsistency and message traffic and inconsistency and fairness. Low volumes of messaging result in high inconsistency and poor fairness.

Algorithm performance was compared using the hypervolume quality indicator [10]. The hypervolume indicator measures how much of the objective space is dominated by the solutions in a given set. Consequently, it provides a good indicator of both convergence to the Pareto front and diversity. The hypervolume was calculated for each algorithm run ($N = 30$). Algorithms were compared using the Wilcoxon rank-sum test against the null hypotheses that the median samples were drawn from the same distribution. The Wilcoxon results indicate NSGA-II outperforms both

DOI: 10.1002/cpe

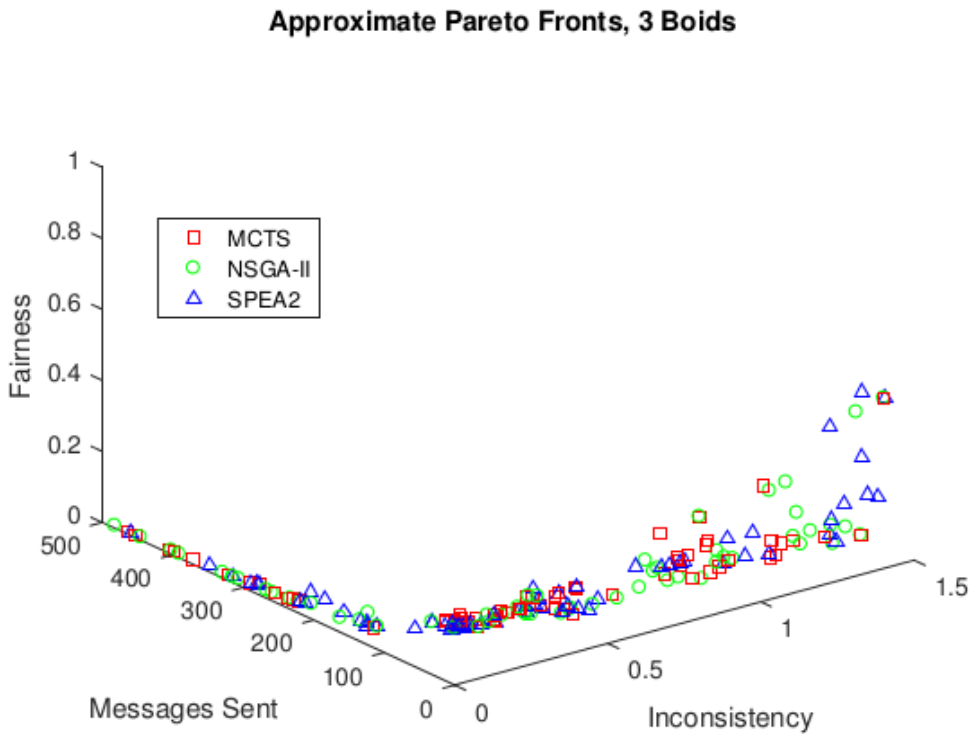


Figure 4. Scatter plot showing the approximate Pareto fronts returned by NSGA-II, SPEA2, and MCTS for Experiment 1.

Table III. Mann-Whitney rank-sum test results for Experiment 2.

	SPEA2	MCTS
NSGA-II	-	▲
SPEA2		-

SPEA2 and MCTS. Additionally, one-way ANOVA indicates statistically significant differences in the sample medians ($p = 0.0,\ \alpha = 0.05$).

Figure 5 plots the approximate Pareto fronts returned by NSGA-II, SPEA2, and MCTS in objective space for Experiment 2. Figures 6 to 8 plot the planar projections of the data in Figure 5. All three algorithms achieve good convergence and diversity and show a distinct Pareto front.

Algorithm performance was again compared with respect to the hypervolume indicator. Due to the small sample size ($N = 10$), the Mann-Whitney rank-sum test was used instead of the Wilcoxon rank-sum test. Results are tabulated in Table III. No statistical difference was found between NSGA-II and SPEA2 or between SPEA2 and MCTS. However, NSGA-II was found to outperform MCTS ($p = 0.0493,\ \alpha = 0.05$).

Both experiments show well-defined Pareto fronts indicating trade-offs between inconsistency, response time, and fairness. Additionally, there are definite lower limits to performance in any of these dimensions.

For example, Experiment 2 clearly shows that there is a minimum achievable inconsistency and that as inconsistency approaches this value, the response time increases dramatically. This drives an attendant degradation in fairness. This result is important since it implies that there are diminishing returns as one approaches the theoretical minimum for inconsistency (see Figures 6 and 7).

Fortunately, there is a wide area of acceptable performance with low inconsistency, low response time, and reasonable fairness. However, there is wide variability in fairness in this region driven primarily by changing message latency due to jitter and congestion. It should be noted; however, that while response times remain low in this region, the amount of state updates sent becomes

DOI: 10.1002/cpe

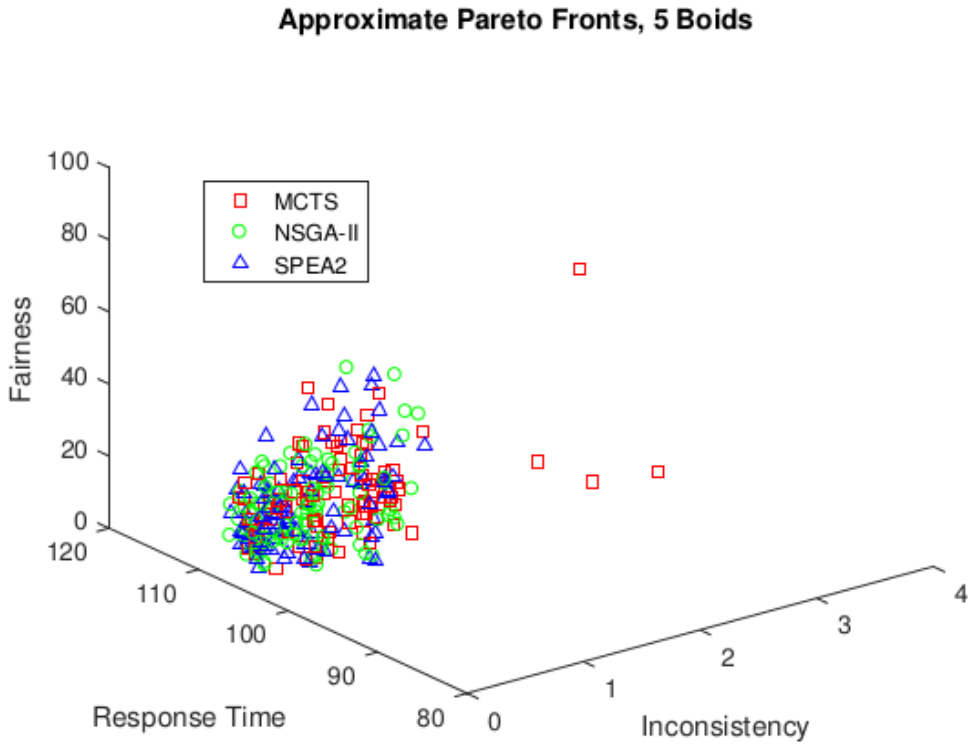


Figure 5. Approximate Pareto fronts achieved by NSGA-II, SPEA2, and MCTS for Experiment 2.

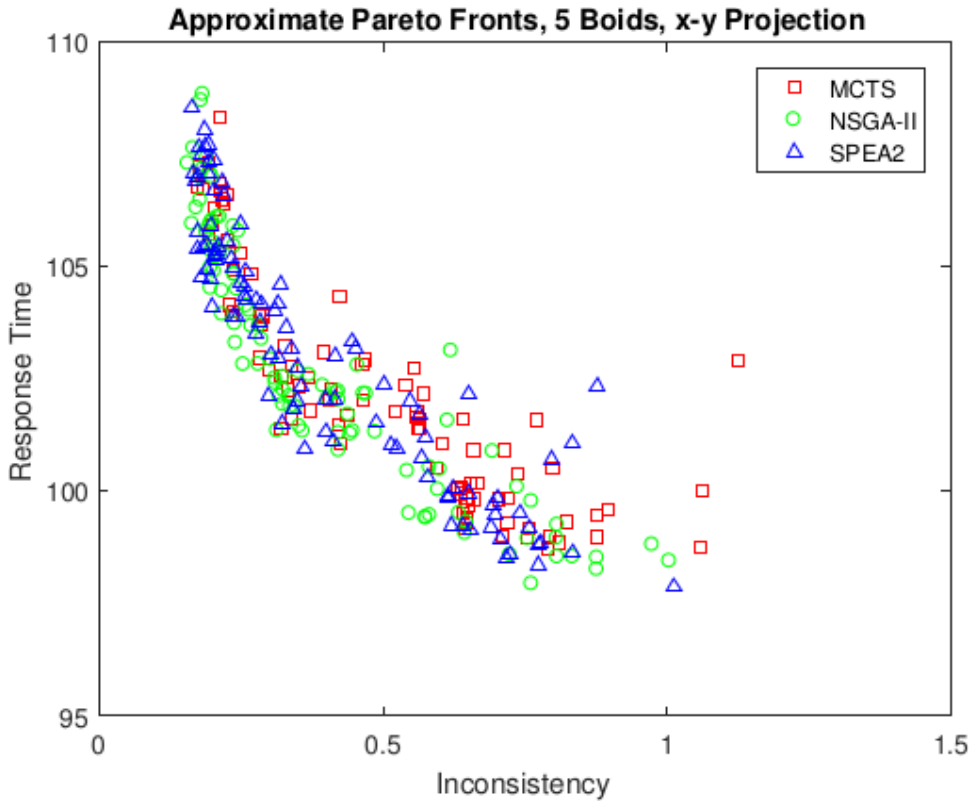


Figure 6. Projection of the Pareto front on the inconsistency-response time plane for Experiment 2.

DOI: 10.1002/cpe

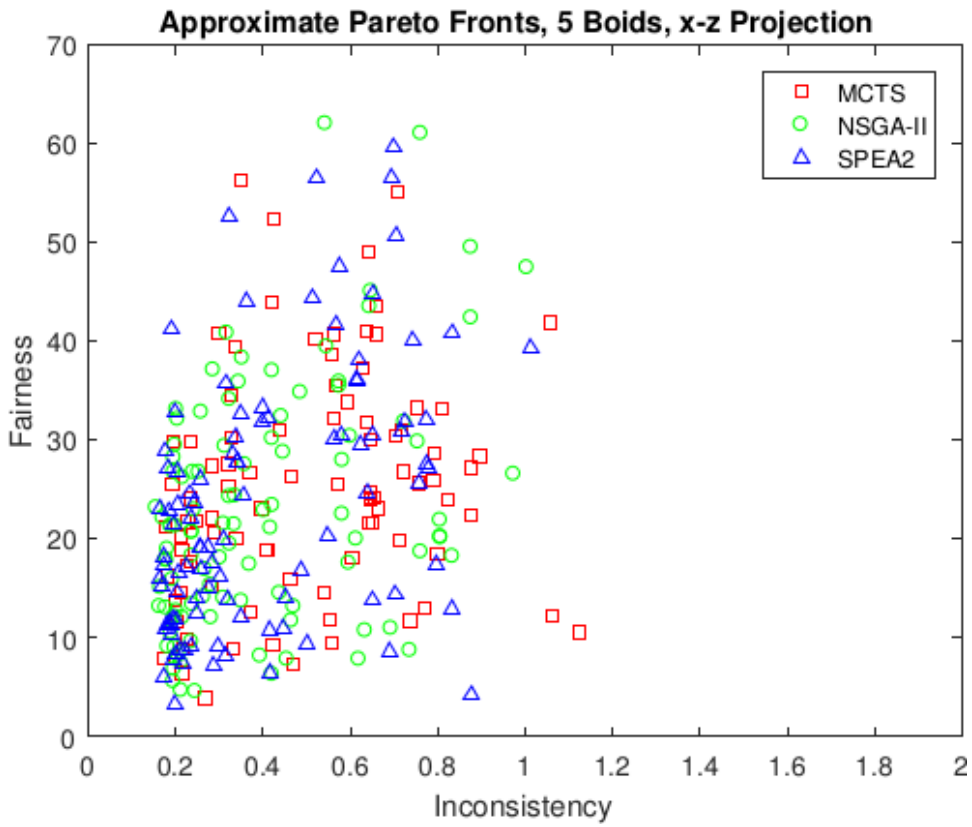


Figure 7. Projection of the Pareto front on the inconsistency-fairness plane for Experiment 2.

rather large as borne out by Experiment 1. The systems under test in this work are small; real-world systems include many more entities and nodes. Thus, while response time may not become a design constraint, overall message volume may well limit scalability. Additionally, while not modeled here, one should also expect response time to increase as message volume increases due to network routing.

3. OPTIMIZING PLAUSIBILITY

Interactions between simulated entities are a defining characteristic of DVEs; without interactions, there would be no need to design and built a distributed simulation system. Indeed, the difference between two simulations largely comes down to the set of entities involved and the interactions among them. For instance, an air-to-air combat simulation might include ground units as part of the environment or virtual world, but not allow aircraft to interact with them. Conversion from an air-centric to combined arms simulation might be as simple as expanding the set of allowed interactions to include entities on the ground.

In any case, the set of interactions supported by a simulation directly impact its consistency and responsiveness requirements. Some interactions, such as collision detection or engaging an entity with a high-precision weapon, have a low tolerance for inconsistency in the simulation state. Others, such as tracking a target with a long-range sensor, might tolerate higher levels of inconsistency before erroneous results are generated.

In [19], Itzel et. al. present the notion of the interaction context for a state update. This context captures the type, affected entities, and any dependent interactions for an update message. From this information, system designers can derive consistency and interactivity requirements. For instance, state updates for an entity moving through the environment without affecting any other entities

DOI: 10.1002/cpe

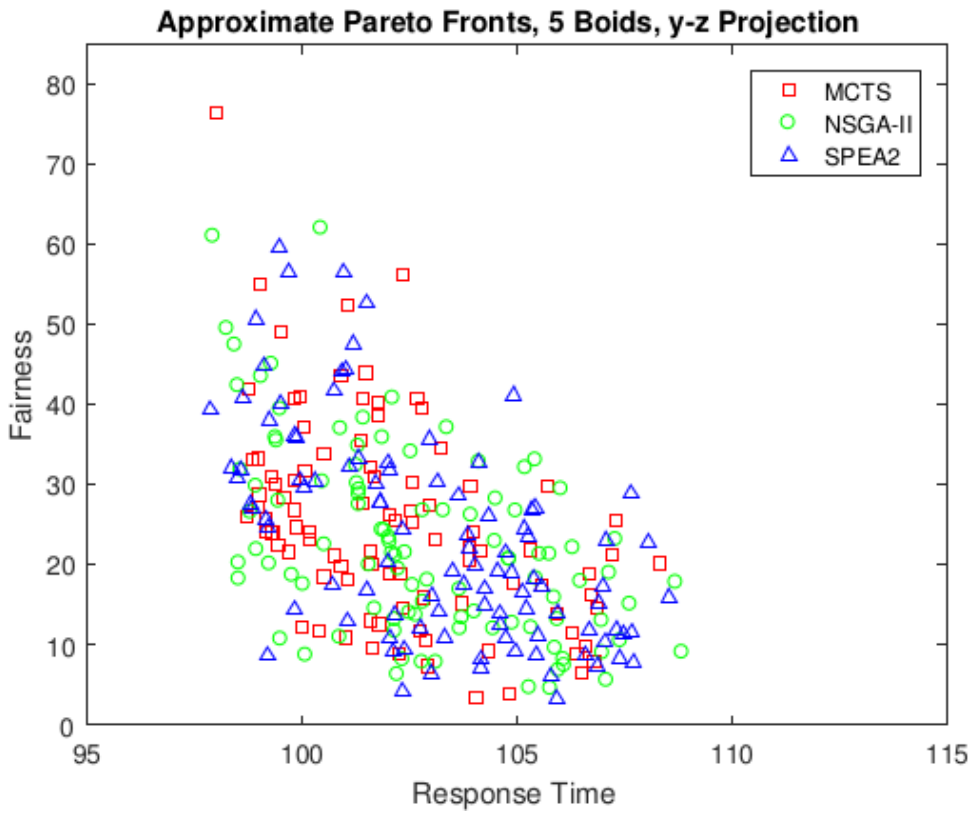


Figure 8. Projection of the Pareto front on the response time-fairness plane for Experiment 2.

would have low consistency and interactivity requirements. Updates associated with two entities engaged in combat, on the other hand, would have more stringent consistency and interactivity requirements.

Similarly, Claypool and Claypool have categorized user actions in online games according to their precision and deadline requirements [9]. Here, precision means the degree of accuracy necessary to complete the (inter)action successfully and is analogous to a consistency requirement. High precision actions show a high sensitivity to state inconsistency, while low precision actions are more tolerant of inconsistent state. Deadline refers to the time required to achieve the final result of an action. The authors show that network latency has a larger impact on game actions with tight precision or deadline requirements.

The foregoing works set the stage for an investigation of interactions, however, they have some limitations when applied to DIS-style (i.e., peer-to-peer or serverless) LVC simulations. First, both assume a centralized client/server system responsible for processing user actions. Itzel's interactivity and Claypool's deadline requirements are derived from this assumption. Highly interactive actions (those with tight deadlines) require a rapid response from the server processing the actions. This is exactly what is meant by simulation responsiveness [29, 17]. However, peer-to-peer LVC simulations are able to respond immediately to user inputs since no communication with other system nodes is required. With that said, responsiveness does impact a DVE's ability to meet interaction requirements; Section 3.1 explores this relationship.

Second, and more problematically, both Itzel and Claypool characterize the consistency (precision) requirements derived from examining interactions categorically. This makes it difficult to derive concrete system requirements. For example, just how precise must a "high" precision action be? A more formal model of interaction is required to accurately derive consistency and responsiveness requirements from interaction requirements.

DOI: 10.1002/cpe

Suppose we have a peer-to-peer DVE with n entities denoted e_i, $i = 1 \ldots n$. We define an *interaction* to be a state dependency between two entities denoted by the tuple

$$I = (e_i, e_j, L) \tag{8}$$

The source of the interaction, e_i, depends on the state of the target entity e_j. Note the dependence is unidirectional. This allows asymmetric interactions to be modeled. For instance, e_i may be a long-range sensor tracking e_j at a sufficiently large distance so that e_j is unaware of e_i. Situations involving two entities that are mutually interacting can be modeled with a pair of interactions I_1 and I_2.

With this in mind, we define the plausibility limit L as the maximum spatial error the interaction can support before participating entities disagree on the outcome. Our focus here is on spatial data since nearly all interactions of interest in an operational test scenario are position-dependent. However, we note that the concept of a plausibility limit is applicable to any continuous state data for which a rate of change can be defined.

The parameters of a particular interaction should be defined by the engineering requirements the simulation is meant to address. For instance, in an aerial refueling simulation, if the boom operator must place the boom within 10 cm of the fuel receptacle to successfully couple the aircraft, then the tolerance for that interaction would be 10 cm. Similarly, if certain entities are included in a simulation as a backdrop to define the context, then there is little consequence if their positions are inconsistent. Thus, interactions such as rendering their position to the user's display would have a high plausibility limit.

3.1. Spatial Error Model

Deriving state synchronization parameters from interaction tolerances requires a model for how spatial state inconsistency develops in the first place. The node hosting the target entity e_j of an interaction maintains the true value of its state. If the replica of that state at the node hosting the interaction's source entity deviates from the true value, there is a state inconsistency. Any inconsistency in the position data is called spatial error. This error represents the difference between e_i's perception of e_j's position and e_j's true position.

More formally, and following the definitions in [3], let the *real path*, $\mathcal{R}(t)$, represent the true position of e_j at any time t. $\mathcal{R}(t)$ is a function of the user input and physics models employed by the simulation node n_j hosting e_j. At intervals, n_j sends a message updating the replica of e_j at the source node n_i. These state updates coupled with any dead reckoning algorithms in use at n_i yield the *placed path*, $\mathcal{P}(t)$. This path represents e_i's perception of e_j's position at any time t. The difference between the real and placed paths

$$E(t) = \mathcal{R}(t) - \mathcal{P}(t) \tag{9}$$

is the spatial error at time t.

Moreover, the position of the entity on the placed path after Δt seconds can be approximated using Taylor series expansion [16],

$$\mathcal{R}(t + \Delta t) \approx \mathcal{R}(t) + \frac{\mathrm{d}\mathcal{R}}{\mathrm{d}t}\Delta t + \frac{1}{2}\frac{\mathrm{d}^2\mathcal{R}}{\mathrm{d}t^2}\Delta t^2$$

Letting $s(t) = \mathcal{R}(t)$, $v(t) = \frac{\mathrm{d}\mathcal{R}}{\mathrm{d}t}$, and $a(t) = \frac{\mathrm{d}^2\mathcal{R}}{\mathrm{d}t^2}$ yields the familiar kinematic equation

$$\mathcal{R}(t + \Delta t) \approx s(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$$

where $s(t)$, $v(t)$, and $a(t)$ represent the entity's position, velocity, and acceleration at time t.

The placed path at time $t + \Delta t$ depends on the dead reckoning algorithm in use at the replica node. If no dead reckoning is employed, then

$$\mathcal{P}(t + \Delta t) = \mathcal{P}(t) = s(t)$$

DOI: 10.1002/cpe

If first-order dead reckoning is employed, then

$$\mathcal{P}(t + \Delta t) = s(t) + v(t)\Delta t$$

Combining the expressions for the real and placed paths yields the following expressions for the spatial error at time $t + \Delta t$:

$$E(t + \Delta t) = v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{10}$$

in the case of no dead reckoning and

$$E(t + \Delta t) = \frac{1}{2}a(t)\Delta t^2 \tag{11}$$

in the case of first-order dead reckoning.

If the entity's maximum velocity and acceleration are known (a safe assumption) and t_{update} is the time the last update was sent, then the spatial error after receiving the update can be bounded above as

$$E(t_{update} + \Delta t) \leq v_{max}\Delta t + \frac{1}{2}a_{max}\Delta t^2 \tag{12}$$

and

$$E(t_{update} + \Delta t) \leq \frac{1}{2}a_{max}\Delta t^2 \tag{13}$$

for the no dead reckoning and first-order dead reckoning cases respectively.

3.2. Deriving Synchronization Parameters

Combining the notion of an interaction with the foregoing error model provides a straightforward means of deriving synchronization protocol parameters from high-level engineering requirements. Recall that an interaction is described by

$$I = (e_i, e_j, L)$$

where L specifies the maximum error in e_j's position that yields a correct result for the interaction. Thus, for a simulation employing dead reckoning, we can compute the temporal accuracy interval [20] for a state update in the context of interaction I as

$$\Delta t = \sqrt{\frac{2 * L}{a_{max}}} \tag{14}$$

Assuming state updates are sent from e_j to e_i periodically, the quantity Δt specifies the maximum time for which a state update is accurate. After Δt seconds, the update will have been superseded by another and the spatial error will have exceeded the tolerance of interaction I. Note that periodic state updates have been shown to result in optimal state consistency [31]. Thus, our concern is to find the largest interupdate period p that meets the error tolerance for interaction I.

To do so, we note that the temporal accuracy interval is the difference between the time the last update was sent, t_{update} and the time of its use, t_{use}. That is,

$$\Delta t = t_{use} - t_{update}$$

Noting that the current value of the state becomes invalid when the next update is sent and letting d represent the total update propagation delay (accounting for network latency, queuing, etc), the largest accuracy interval is obtained when

$$\Delta t = t_{update} + p + d - t_{update} \tag{15}$$

DOI: 10.1002/cpe

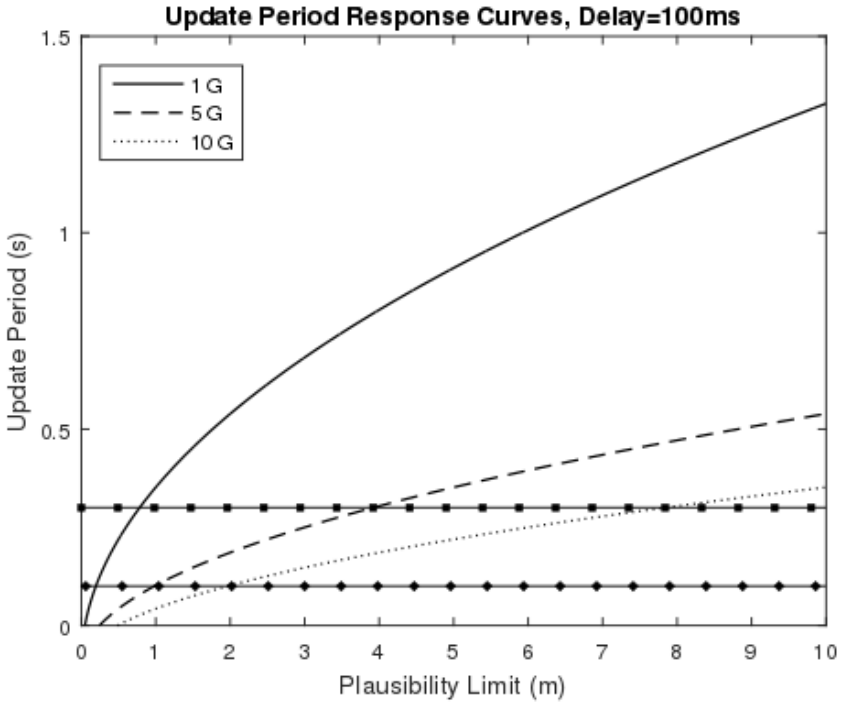


Figure 9. Update period as a function of plausibility limit with a delay of 100 ms and maximum acclerations of 1, 5, and 10 G. Blue and red lines indicate 100 ms and 300 ms update periods respectively.

To summarize, let e_i be an entity interacting with entity e_j. Suppose the interaction has some maximum plausibility limit L. Let a_{max} be the maximum acceleration of e_j and d be the state update propagation delay. Then the maximum period between state updates from e_j to e_i is given by

$$p = \sqrt{\frac{2 * L}{a_{max}}} - d \tag{16}$$

3.3. Model Validation

Validation of the optimal update period model was conducted using a two-node simulated DVE. The server node hosts a single entity employing a random acceleration motion model. The client node maintains a replica of the server's entity and employs local dead reckoning as a consistency control algorithm. Updates consist of the current position and velocity of the entity and are sent from the server to the client at periodic intervals.

Figure 9 plots the predicted optimal update periods as a function of the plausibility limit for entities with a maximum acceleration of 1, 5, and 10 times the force of gravity. The predicted response curves assume an entity undergoing constant maximum acceleration and therefore indicate a worst-case scenario. The horizontal lines indicate update periods of 100 ms (diamonds) and 300 ms (squares). These are the maximum limits recommended by the DIS standard for tightly-coupled and loosely-coupled simulations. Cross-referencing these lines with the response curves indicate the maximum obtainable plausibility limit for an entity undergoing constant maximum acceleration.

In the case of an entity whose maximum acceleration is 10 G (typical of military aircraft), the maximum plausibility limit for any interaction I is 2 m. Any interaction that requires a maximum spatial deviation less than 2 m cannot be guaranteed to provide a correct outcome. This is often manifested as a divergence in interaction outcomes for each participating entity. For example, a

DOI: 10.1002/cpe

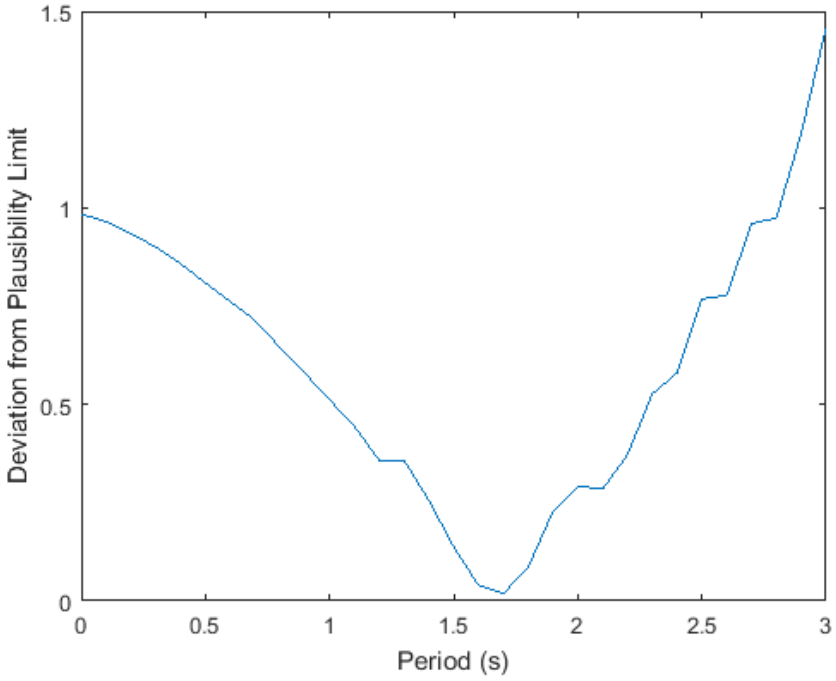


Figure 10. Deviation between the 95% upper tolerance bound ($1 - \alpha = 0.05$) and plausibility limit $L = 1.0$ for an entity with normally distributed acceleration $N(0, 1)$.

collision detection routine requiring accuracy of 1 m may result in one entity observing a collision while the other does not.

As noted, the model provides a worst-case prediction of the update period required to obtain a plausibility limit of L. In real DVEs, entities rarely undergo sustained maximum accelerations. In fact, entity acceleration is practically zero the vast majority of the time. If the requirement for meeting the plausibility limit is relaxed in a statistical fashion, then the required update period can be quite a bit higher than that predicted by the optimal model. That is, if the spatial error is allowed to exceed the limit some fraction of the time, then tolerance limits can be computed at a specified level of confidence. The guarantee provided by Equation 16 then becomes something similar to "with confidence $1 - \alpha$, the spatial error will be within the plausibility limit L 95% of the time." More formally, we have the following optimization problem:

$$\min \sqrt{\frac{2 * L}{a_{max}}} - d - L \tag{17}$$

Figure 10 plots the deviation between the 95% upper tolerance bound ($1 - \alpha = 0.05$) for an entity moving according to a normally distributed acceleration profile ($a\ N(0, 1)$, $a_{\max} = 9.8$) and a plausibility limit $L = 1.0$ with an update propagation delay of 100 ms. The predicted update period according to Equation 16 is 0.3518 s. On the other hand, relaxing the consistency constraints and solving the minimization problem given in Equation 17 yields an update period of 1.75 s. This larger period ensures that 95% of the spatial error over the course of the simulation will be less than or equal to $L = 1.0$ with 95% confidence.

Finally, we have conducted subsequent experiments assessing the likelihood of meeting plausibility limits using motion models derived from World of Warcraft trace data [23]. These experiments indicate the probability of exceeding a "loose" plausibility limit ($L \geq 1.0$) is quite low for real-world motion models. However, as the plausibility limit decreases, the exceedence probability reaches a non-zero minimum – approximately 4% for an interupdate period of 0 ms and 20% for an interupdate period of 100 ms with $L = 0.1$.

DOI: 10.1002/cpe

4. CONCLUSIONS

This paper presented two algorithms designed to provide optimal update scheduling parameters. The first algorithm is a multi-objective optimization for virtual environments employing dead-reckoning-based state updates. It provides the optimal dead-reckoning error threshold in terms of response time, state consistency, and fairness. Here, fairness is defined in terms of a cluster cohesion metric that ensures all simulation users see similar levels of state consistency and response time.

The second algorithm applies to virtual environments employing periodic state updates and is based on the notion of interaction contexts and plausibility limits. Plausibility limits are defined as the maximum tolerable state inconsistency that allows all participants in an interaction to reach the same conclusion or outcome. Using object position as the state of interest, a statistical model of consistency is derived in terms of update period and spatial error. Solution of the associated optimization problem yields the largest inter-update period meeting the plausibility limit with a specified level of confidence.

Both algorithms provide insight into the performance of distributed virtual environments. Optimizing dead-reckoning for fairness, consistency, and response time ensures that all participants have a similar experience and that no user has a systemic advantage. Optimizing only for fairness tends to yield DVE systems that perform only as well as the worst-performing node and can be frustrating for users. Treating the problem as a multi-objective problem ensures that all users see similar levels of responsiveness and consistency. Treating fairness in terms of cluster cohesion ensures that all users are equally handicapped. Importantly, it is possible for the DVE to be fair while failing to perform adequately. This is of particular concern for DVEs supporting applications such as engineering test and evaluation. In these cases, a different optimization must be pursued to ensure fitness for purpose and allow some sense of system calibration. Optimizing the update period in terms of plausibility limits fulfills this need.

Plausibility limits are derived directly from the interactions supported by a DVE and provide a means for ensuring that all parties see a similar outcome for any given interaction despite variances in the state data at each node. Borrowing from real-time systems theory, the maximum period that supports a particular limit can be derived from a simple model of spatial error. This can be done either absolutely or in a statistical sense.

Together, these algorithms provide systems designers a means to provide both a “fair fight” and plausible outcomes for virtual environments supporting operational test scenarios. This is a necessary first step to quantifying the uncertainty associated with experiments conducted on virtual environments and ensuring valid conclusions are drawn from the resulting data. Future work includes continued study into the properties of plausibility limits and their utility for defining the capabilities of distributed virtual environments with particular application to military simulation, test, and analysis.

REFERENCES

1. Second Life. http://secondlife.com.
2. World of Warcraft. http://www.worldofwarcraft.com.
3. Sudhir Aggarwal, Hemant Banavar, Amit Khandelwal, Sarit Mukherjee, and Sampath Rangarajan. Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 161–165. ACM, 2004.
4. Anastasiia Beznosyk, Peter Quax, Karin Coninx, and Wim Lamotte. Influence of network delay and jitter on cooperation in multiplayer games. In *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*, pages 351–354. ACM, 2011.
5. Eric A Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.
6. Jeremy Brun, Farzad Safaei, and Paul Boustead. Fairness and playability in online multiplayer games. *Faculty of Informatics-Papers*, page 232, 2006.
7. Kuan-Ta Chen, Polly Huang, and Chin-Laung Lei. Effect of network quality on player departure behavior in online games. *Parallel and Distributed Systems, IEEE Transactions on*, 20(5):593–606, 2009.
8. Peng Chen and Magda El Zarki. Perceptual view inconsistency: An objective evaluation framework for online game quality of experience (qoe). In *Proceedings of the 10th Annual Workshop on Network and Systems Support for Games*, NetGames ’11, pages 2:1–2:6, Piscataway, NJ, USA, 2011. IEEE Press.

DOI: 10.1002/cpe

9. Mark Claypool and Kajal Claypool. Latency can kill: precision and deadline in online games. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 215–222. ACM, 2010.
10. Carlos A Coello Coello, Gary B Lamont, and David A Van Veldhuisen. *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007.
11. Judith S Dahmann, Richard M Fujimoto, and Richard M Weatherly. The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*, pages 142–149. IEEE Computer Society, 1997.
12. Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.
13. DIS Steering Committee. IEEE standard for distributed interactive simulation-application protocols. *IEEE Standard*, 1278, 1998.
14. J. J. Durillo and A. J. Nebro. jMetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.
15. Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, Inc, 2000.
16. Dai Hanawa and Tatsuhiro Yonekura. On the error modeling of dead reckoned data in a distributed virtual environment. In *Cyberworlds, 2005. International Conference on*, pages 8–pp. IEEE, 2005.
17. Douglas D Hodson and Rusty O Baldwin. Performance analysis of live-virtual-constructive and distributed virtual simulations: defining requirements in terms of temporal consistency. 2009.
18. Douglas D Hodson and Raymond R Hill. The art and science of live, virtual and constructive simulation for test and analysis. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, page 1548512913506620, 2013.
19. Laura Itzel, R Suselbeck, Gregor Schiele, and Christian Becker. Specifying consistency requirements for massively multi-user virtual environments. In *Haptic Audio Visual Environments and Games (HAVE), 2011 IEEE International Workshop on*, pages 1–2. IEEE, 2011.
20. Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer, 2011.
21. Damien Marshall, Séamus McLoone, Tomás Ward, and Declan Delaney. Does reducing packet transmission rates help to improve consistency within distributed interactive applications? 2006.
22. Martin Mauve. How to keep a dead man from shooting. In *Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 199–204. Springer, 2000.
23. Jeremy R. Millar, Douglas D. Hodson, Gilbert L. Peterson, and Darryl K. Ahner. Data quality challenges in distributed live-virtual-constructive test environments. *ACM Journal of Data and Information Quality*, To appear.
24. Jeremy R Millar, Douglas D Hodson, and Richard Seymour. Deriving lvc state synchronization parameters from interaction requirements. In *Distributed Simulation and Real Time Applications (DS-RT), 2016 IEEE/ACM 20th International Symposium on*, pages 85–91. IEEE, 2016.
25. J.R. Millar, D.D. Hodson, G.B. Lamont, and G.L. Peterson. Multi-objective optimization of dead-reckoning error thresholds for virtual environments. In *Collaboration Technologies and Systems (CTS), 2014 International Conference on*, pages 562–569, May 2014.
26. J Russell Noseworthy. The test and training enabling architecture (TENA) supporting the decentralized development of distributed applications and lvc simulations. In *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*, pages 259–268. IEEE, 2008.
27. Eric Paul Parker, Nadine Elizabeth Miner, Brian Peter Van Leeuwen, and James Brian Rigdon. Testing unmanned autonomous system communications in a live/virtual/constructive environment. *International Test and Evaluation Association Journal (ITEA)*, 30:513–522, 2009.
28. Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987.
29. Sandeep Singhal and Michael Zyda. *Networked virtual environments: design and implementation*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.
30. Anthony Steed and Manuel Fradinho Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments*. Elsevier, 2009.
31. Xueyan Tang and Suiping Zhou. Update scheduling for improving consistency in distributed virtual environments. *Parallel and Distributed Systems, IEEE Transactions on*, 21(6):765–777, 2010.
32. Brian Van Leeuwen, Vincent Urias, John Eldridge, Charles Villamarin, and Ron Olsberg. Performing cyber security analysis using a live, virtual, and constructive (lvc) testbed. In *Military Communications Conference, 2010-MILCOM 2010*, pages 1806–1811. IEEE, 2010.
33. Andras Varga. Omnet++. In *Modeling and Tools for Network Simulation*, pages 35–59. Springer, 2010.
34. Weijia Wang, Michèle Sebag, et al. Multi-objective monte-carlo tree search. In *Asian conference on machine learning*, 2012.
35. Suiping Zhou, Wentong Cai, Bu-Sung Lee, and Stephen J Turner. Time-space consistency in large-scale distributed virtual environments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(1):31–47, 2004.
36. Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm, 2001.

DOI: 10.1002/cpe

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
09/14/2017	Dissertation	Sep 2012 - Sep 2017

4. TITLE AND SUBTITLE
A Stochastic Model of Plausibility in Live-Virtual-Constructive Environments

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Millar, Jeremy R., Maj, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
Wright-Patterson AFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT-ENG-17-S-015

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution Statement A. Approved for Public Release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES
This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT
Distributed live-virtual-constructive simulation promises a number of benefits for the test and evaluation community. However, geographically distributed systems are subject to fundamental state consistency limitations that make assessing the data quality of live-virtual-constructive experiments difficult. This research presents a data quality model based on the notion of plausible interaction outcomes. This model accounts for the lack of consistency in distributed real-time systems and offers system designers a means of estimating data quality and fitness for purpose. Experiments with trace data validate the plausibility model and exceedance probability estimates.

15. SUBJECT TERMS
Distributed Simulation; LVC; Plausibility

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Douglas D. Hodson, AFIT/ENG
U	U	U	UU	123	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565 x4719