Update Scheduling for Improving Consistency in Distributed Virtual Environments

Xueyan Tang, *Member, IEEE*, and Suiping Zhou, *Member, IEEE*

Abstract—The fundamental goal of distributed virtual environments (DVEs) is to create a common and consistent presentation of the virtual world among a set of computers inter-connected by a network. This paper investigates update scheduling algorithms to make efficient use of network capacity and improve consistency in DVEs. Our approach is to schedule state updates according to their potential impacts on consistency. In DVEs, the perceptions of participants are affected by both the spatial magnitude and temporal duration of inconsistency in the virtual world. Using the metric of time-space inconsistency, we analytically derive the optimal update schedules for minimizing the impact of inconsistency. Based on the analysis, we propose a number of scheduling algorithms that integrate spatial and temporal factors. These algorithms also take into consideration the effect of network delays. The algorithms can be used on top of many existing mechanisms such as dead reckoning. Experimental results show that our proposed algorithms significantly outperform the intuitive algorithms that are based on spatial or temporal factors only.

Index Terms—distributed virtual environment, time-space consistency, state update, scheduling.

1 Introduction

A distributed virtual environment (DVE) allows participants at different locations to communicate and interact with each other in a virtual world [10], [14]. DVEs have been widely used in many areas such as military training [6], collaborative design [1], e-learning [2] and network-based multi-user games [13]. As shown in Figure 1, a DVE normally comprises a group of inter-connected computers (nodes) that simulate a set of entities evolving in a virtual world.

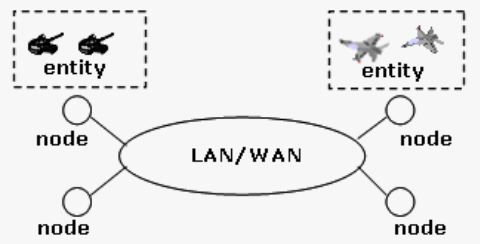


Fig. 1. Distributed virtual environment.

The fundamental goal of DVEs is to create a common and consistent presentation of the virtual world among the participants. When an action is taken by a participant, or the state of an entity changes in the virtual world, all participants should be able to see the change in real time. However, due to the constraints of limited network capacity and message transmission delays, a consistent view among different participants is not guaranteed automatically. If some state updates are not timely disseminated to all nodes, inconsistency is likely to occur in a DVE application. Inconsistent views of the virtual world can seriously affect meaningful interactions among the participants (e.g., it may give some participants advantages over the others in an air combat game). The consistency problem is exacerbated as the size of the system increases. Therefore, an important issue in DVEs is how to make effective use of the limited network capacity to reduce inconsistency.

In this paper, we investigate update scheduling algorithms to efficiently utilize the network capacity for improving consistency in DVEs. Our approach is to schedule state updates according to their potential impacts on consistency. In DVEs, the perceptions of participants are affected by both the spatial magnitude and temporal duration of inconsistency in the virtual world. The participants may not spot inconsistency lasting for short periods even if the spatial difference is relatively large, but they may notice inconsistency lasting for long periods even if the spatial difference is small. To this end, a metric called time-space inconsistency has been proposed and shown to effectively reflect the impact of inconsistency on a participant's perception and decision-making in DVE applications [11].

Using the metric of time-space inconsistency, we analytically derive the optimal update schedules for minimizing the impact of inconsistency. Based on the analysis, we propose a number of scheduling algorithms that integrate spatial and temporal factors. These algorithms also take the effect of network delays into consideration. The proposed algorithms are generic and can be used on top of many existing mechanisms such as dead reckoning [3] and relevance filtering [8] for improving consistency. Experimental results show that our algorithms significantly outperform the intuitive algorithms that are based on spatial or temporal factors only.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 introduces the system model. Two intuitive update scheduling algo-

• *X. Tang and S. Zhou are with the School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798. E-mail: {asxytang, asspzhou}@ntu.edu.sg.*

rithms are presented in Section 4. Section 5 analyzes the optimal update schedules. Based on the analytical results, Section 6 proposes a number of update scheduling algorithms for improving consistency in DVEs. The experimental setup and results are discussed in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

Some work exists on reducing the amount of data exchanged over the network in DVEs. Two widely used techniques are dead reckoning (DR) and relevance filtering. In dead reckoning [3], [7], each node maintains a DR model for each moving entity to extrapolate the entity state between updates. This helps to cut down the number of state updates needed to keep the node's view consistent. Different from dead reckoning, relevance filtering reduces network traffic by disseminating updates only to interested nodes [5], [8]. For example, a participant may just be interested in the state updates of the entities close to its avatar in the virtual environment. Though these techniques are successful in reducing network traffic, the total traffic volume may still be very high for large-scale DVEs with massive number of entities. Our update scheduling algorithms complement these techniques in the sense that they can be used on top of them. When the number of state updates that a node needs to send (as required either by dead reckoning or relevance filtering) exceeds the network capacity, our algorithms determine the priority of disseminating these updates according to their potential impacts on consistency.

Lui [4] analyzed the optimal synchronization interval for dead reckoning with an objective of bounding the maximum phase difference. A priority round-robin algorithm was presented in [20] to reduce the expected spatial difference in entity states. Though this algorithm shared the spirit with an intuitive SPACE algorithm we present in Section 4, it did not take into account the impact of network delays. Similarly, Yu [21] *et al.* studied the optimization of bandwidth allocation for minimizing the expected spatial difference in entity states. However, none of the above studies has considered time-space inconsistency. It has been shown that in addition to spatial magnitude, the temporal duration of inconsistency also affects human perception significantly in DVE applications [11]. Zhou *et al.* [11] defined a metric for characterizing time-space consistency in DVEs but did not propose any update scheduling algorithm for improving time-space consistency. Also relevant to our work is the synchronization of distributed databases. A commonly used consistency metric in database synchronization is freshness, which indicates whether data objects are up-to-date [18]. Olston *et al.* [19] presented a more general metric called divergence. They aimed at minimizing time-averaged divergence between source data objects and cached copies by selectively refreshing modified objects. These metrics, however, are not adequate to characterize the impact of inconsistency in DVEs, which is dependent on both the spatial magnitude and temporal duration of inconsistency. As shall be discussed in Section 3, time-space inconsistency in DVEs is defined over periods known as situations [11] that are driven by the instantaneous spatial magnitude of inconsistency in entity states. Moreover, the above work on database synchronization did not consider the network delays in update dissemination as it is normally negligible compared to the inter-update periods. Nevertheless, the impact of network delays cannot be ignored in DVEs as the entity states are often updated continuously. Due to network delays, inconsistency continues to grow, at possibly different rates for different entities, when state updates are in transmission. Therefore, it is important to take the impact of network delays into account in determining the priority of state updates.

3 System Model

There are generally two types of architectures for organizing the nodes interacting in a DVE. In the client-server architecture [15], [16], [17], [24], all entity states are maintained by one or more nodes called servers. The remaining nodes, known as clients, connect to the servers to send their actions and receive state updates. In the peer-to-peer architecture [12], [13], each node is responsible for maintaining the states of the entities under its control. The node processes local actions and delivers state updates to the other nodes. Without loss of generality, in this paper, we consider update dissemination from a node hosting the entities to another node receiving the state updates. The former node is termed *the server*, which can either be a server in the client-server architecture or a peer in the peer-to-peer architecture. The server periodically refreshes the entity states based on their kinetics and user actions. The period between two successive refreshes is known as a *frame*. The node receiving the updates is termed *the client*. Though we shall focus on one client in this paper, our analysis and proposed methods can easily be applied to multiple clients by conceptually considering the same entity in different clients' views as different entities.

In this paper, we focus on the updates of entity positions in the virtual world, which are a common type of entity states requiring constant updates. Our proposed techniques are generally applicable to updating any numerical entity states provided that a norm function $\|A, B\|$ is defined to measure the difference between two entity states A and B.[1]

We denote the entity positions in the virtual world by multi-dimensional vectors. Let $\vec{S}(t)$ be the position of an entity as a function of time in the server's view, and $\vec{C}(t)$ be the position of the same entity in the client's view. In the simplest update model [4], at each update,

1. We remark that updates to non-numerical entity states (e.g., changing the state of an avatar from "alive" to "dead") normally occur much less frequently. These updates are often disseminated to the client immediately.

the server sends the latest entity position to the client. The client maintains the entity position last received from the server till the next update. Suppose that the network delay between the server and the client is d. Then, the latest update received by the client at time t is the one last sent by the server before time $t-d$. It follows that $\vec{C}(t) = \vec{S}(t_L)$, where $t_L \leq t-d$ is the time of the update last sent by the server before $t-d$. More sophisticatedly, the server can send extra information to the client to help it to predict the entity position between updates. For example, in the first-order dead reckoning (DR) model [3], [11], the server also sends the moving speed of the entity to the client at an update. On receiving the update, the client starts extrapolating the entity position using the DR model. As a result, $\vec{C}(t) = \vec{S}(t_L) + (t-t_L) \cdot \frac{\partial \vec{S}(t)}{\partial t}\big|_{t=t_L}$. The construction of extrapolation models is orthogonal to the focus of this paper and beyond our scope. Without loss of generality, we shall denote by $\mathbb{S}(t)$ the entity state in the server's view at time t (including the entity position and other information such as moving speed needed by the extrapolation model in use if any) and express $\vec{C}(t)$ by

$$\vec{C}(t) = \mathcal{F}(\mathbb{S}(t_L), t-t_L),$$

meaning that the entity position in the client's view is a function $\mathcal{F}$ (determined by the extrapolation model in use) of the entity state in the server's view at the last update before $t-d$, and the time elapsed since this last update. We assume that the server is aware of the extrapolation model used by the client. Thus, the server is able to estimate the entity position in the client's view provided that it knows the network delay. The delay is affected by the network state and traffic level. It can be obtained through the network by means of measurement tools such as King [9] and QStat [25].

As discussed earlier, due to network delays and limited network capacity, inconsistency is likely to occur between the server and client views. The effect of inconsistency on a participant's perception has been shown to be highly dependent on a metric called *time-space inconsistency* [11], which combines the spatial difference between entity positions and its temporal duration in evaluating the magnitude of inconsistency. Let $\Delta(t) = \|\vec{C}(t), \vec{S}(t)\|$ be the spatial difference (e.g., Euclidean distance in the virtual world) between $\vec{S}(t)$ and $\vec{C}(t)$ for an entity as a function of time. Let ϵ be a predefined application-dependent parameter that represents the minimum distance discernible by the participants. Suppose that $\Delta(t)$ starts to exceed ϵ at time t_b and persists till time t_e when it falls back below ϵ. Then, the interval $[t_b, t_e]$ is referred to as a *situation* (denoted by $\mathbf{s}$). The time-space inconsistency of the situation is defined as

$$\int_{t_b}^{t_e} \Delta(t)\,\mathrm{d}t.$$

The impact of a situation on a participant's perception generally increases with its time-space inconsistency [11]. Without loss of generality, we define the impact as a non-decreasing function $\mathcal{I}(\cdot)$ of the time-space inconsistency, i.e., the impact of a situation $\mathbf{s}$ is given by

$$\Omega(\mathbf{s}) = \mathcal{I}\Big(\int_{t_b}^{t_e} \Delta(t)\,\mathrm{d}t\Big).$$

The entities of interest to the client may be either the global set maintained at the server or a subset chosen by relevance filtering. Suppose that due to the limitation of network capacity, in each frame, the server is allowed to update only a given number of c entities with the client. When the number of entities relevant to the client exceeds c, the server would not be able to update the states of all these entities with the client. Therefore, the server has to decide which entities to update with the client in each frame. Our goal is to reduce the impact of inconsistency in DVEs by properly scheduling the updates of various entities from the server to the client subject to the constraint of network capacity. We consider two different performance objectives.

- The first objective $\mathcal{O}1$ is to minimize the total impact of all situations over different entities, i.e.,

$$(\mathcal{O}1) \quad \text{minimize} \quad \sum_{\mathbf{s}} \Omega(\mathbf{s}).$$

- The second objective $\mathcal{O}2$ is to minimize the highest impact of all situations over different entities, i.e.,

$$(\mathcal{O}2) \quad \text{minimize} \quad \max_{\mathbf{s}} \Omega(\mathbf{s}).$$

4 Intuitive Update Scheduling Algorithms

4.1 Round-Robin Algorithm

Round-Robin (RR) algorithm repeatedly updates the positions of the entities in circular order. At each frame, the server updates with the client the c entities whose positions have not been updated for the longest periods. The RR algorithm considers only the temporal factor in scheduling. It does not take into account the spatial difference between entity positions in the server and client views. All entities are simply updated with the same frequency. Thus, the RR algorithm is expected to result in large time-space inconsistency for entities that move quickly.

4.2 SPACE Algorithm

To improve consistency, entities with big spatial difference between the server and client views should be given high priorities in update scheduling. SPACE is an intuitive algorithm that schedules the updates based on the instantaneous spatial difference of the entities. Let d be the mean network delay measured between the server and the client. Then, it is anticipated that an update sent by the server at any time t would not affect the client's view until time $t+d$. Therefore, in deciding which entities

to update at time t, the SPACE algorithm estimates the expected spatial difference of the entities at time $t+d$, and selects to update the top c entities in terms of spatial difference.

The estimation of spatial difference should take into consideration all updates that have been sent by the server before t, including those that might still be in transmission at time t due to network delays. To do so, the server maintains, for each entity, the time t_L of the last update and the entity state $\mathbb{S}_L$ in the server's view at the last update. $\mathbb{S}_L$ is expected to be the server state used by the extrapolation model at the client at time $t+d$. Therefore, the entity position in the client's view at time $t+d$ is estimated as $\mathcal{F}(\mathbb{S}_L, t+d-t_L)$. Meanwhile, the server also uses the current state $\mathbb{S}$ of the entity to estimate its position in the server's view at time $t+d$, i.e., $\mathcal{F}(\mathbb{S}, d)$. Then, the expected spatial difference between entity positions in the server and client views at time $t+d$ is given by

$$\|\mathcal{F}(\mathbb{S}_L, t+d-t_L), \mathcal{F}(\mathbb{S}, d)\|.$$

Figure 2 shows the pseudo code for the SPACE algorithm, where the subscripts i in the notations stand for entity i.

```
let t be the current time;
for each entity i do {
    refresh its state S_i based on kinetics and user actions;
    compute expected spatial difference at time t + d as
        ||F(S_{L,i}, t + d − t_{L,i}), F(S_i, d)||;
    }
select the top c entities in terms of spatial difference
    and update them with the client;
for each entity i updated with the client do {
    record t_{L,i} = t;
    record S_{L,i} = S_i;
    }
```

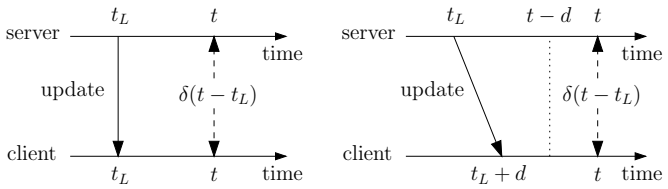
Fig. 2. SPACE algorithm.

The SPACE algorithm, while intuitive, considers only the spatial factor in scheduling. It does not take into account how long the spatial difference has persisted and will persist. As shall be shown by our experimental results in Section 7, the SPACE algorithm does not minimize time-space inconsistency. To reduce time-space inconsistency, it is desirable to consider both temporal and spatial factors in update scheduling.

5 Analysis of Optimal Update Schedules

We now analyze the optimal update schedules in terms of objectives $\mathcal{O}1$ and $\mathcal{O}2$ under some simplifying assumptions. Our analysis is inspired by [19]. The analytical results shall be used to guide the design of update scheduling algorithms in Section 6.

We start by considering a single entity. Assume that in the absence of network delays, after each update, the spatial difference between entity positions in the server and client views grows in the same manner following a non-decreasing function $\delta(\cdot)$ of the time elapsed since the update. Then, at any time t, the spatial difference is given by $\Delta(t) = \delta(t - t_L)$, where t_L is the time of the update last sent by the server before t (see Figure 3(a)). Suppose the network delay between the server and the client is d. In the presence of network delays, an update sent by the server at time t would not take effect in the client's view until time $t+d$. Therefore, the spatial difference is given by $\Delta(t) = \delta(t-t_L)$, where t_L is the time of the update last sent by the server before $t-d$ (see Figure 3(b)). We shall assume that $\delta(d) < \epsilon$, i.e., the spatial difference is brought below the minimum discernible distance when the client updates the entity position on receiving a state update from the server. This is a reasonable assumption because otherwise, the spatial difference would never be smaller than ϵ no matter how frequently the server updates the entity position with the client.



(a) without network delay (b) with network delay

Fig. 3. Impact of network delay on spatial difference.

Recall that only the spatial differences greater than the minimum discernible distance ϵ contribute to time-space inconsistency. To simplify the presentation, we define a new function $\delta^*(\cdot)$ by filtering out the spatial difference less than ϵ in $\delta(\cdot)$, i.e.,

$$\delta^*(t) = \begin{cases} 0 & \text{if } \delta(t) < \epsilon, \\ \delta(t) & \text{if } \delta(t) \geq \epsilon. \end{cases}$$

Then, after the client receives a state update, the growth of the time-space inconsistency for the entity is given by a function $\int_d^\tau \delta^*(t)\,\mathrm{d}t$ of the duration τ since the update is sent out by the server. In the analysis, we assume that the impact of inconsistency goes up more rapidly as time elapses if the server does not update the entity position with the client. That is,

$$\frac{\mathrm{d}\mathcal{I}\Big(\int_d^\tau \delta^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}\tau}$$

is non-decreasing, or

$$\frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^\tau \delta^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}\tau^2} \geq 0.$$

This is plausible since otherwise, it leads to a paradox that a state update by the server with the client (which reduces the instantaneous spatial difference to $\delta(d)$) may increase the growth rate of the impact of inconsistency.

We first show that the total impact of time-space inconsistency and the highest impact of all situations are both minimized when the server updates the entity position with the client periodically.

Theorem 1: Given a fixed number of updates allowed in a period delineated by two updates of an entity, periodic updates result in the minimum total impact of time-space inconsistency over the period and the minimum highest impact of all situations therein.

Proof: Suppose that two updates of the entity are sent by the server to the client at times $t_0 = 0$ and $t_{k+1} = T$, and the entity is allowed to be updated k times in the period $[0, T]$.

Assume that the server updates the entity position with the client at times $0 < t_1 < t_2 < \cdots < t_k < T$. Since these updates would only affect the client's view between times d and $T + d$ (see Figure 4, where d is the network delay) and hence the spatial difference therein, we shall focus on the time-space inconsistency over the period $[d, T + d]$.

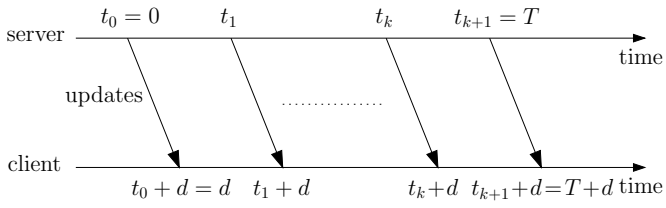


Fig. 4. A period delineated by two updates.

For each $0 \le i \le k$, the spatial difference in the period $[t_i + d, t_{i+1} + d]$ is governed by the update sent by the server at time t_i, i.e.,

$$\Delta(t) = \begin{cases} \delta(t - t_0) & \text{if } t_0 + d \le t < t_1 + d, \\ \delta(t - t_1) & \text{if } t_1 + d \le t < t_2 + d, \\ \cdots & \cdots \\ \delta(t - t_k) & \text{if } t_k + d \le t < t_{k+1} + d. \end{cases}$$

Since $\delta(\cdot)$ is non-decreasing and $\delta(d) < \epsilon$, there is at most one situation in each period $[t_i + d, t_{i+1} + d]$. Therefore, the total impact of time-space inconsistency over the period $[d, T + d]$ is given by

$$\sum_{i=0}^{k} \mathcal{I}\Big(\int_{t_i+d}^{t_{i+1}+d} \delta^*(t - t_i)\, \mathrm{d}t \Big) = \sum_{i=0}^{k} \mathcal{I}\Big(\int_{d}^{t_{i+1}-t_i+d} \delta^*(t)\, \mathrm{d}t \Big).$$

Let $\tau_i = t_{i+1} - t_i$ be the periods between two successive updates, where $i = 0, 1, \cdots, k$. Then, the total impact of time-space inconsistency can be written as

$$\sum_{i=0}^{k} \mathcal{I}\Big(\int_{d}^{\tau_i+d} \delta^*(t)\, \mathrm{d}t \Big).$$

We first prove that for all $i \ne j$,

$$\mathcal{I}\Big(\int_{d}^{\tau_i+d} \delta^*(t)\mathrm{d}t \Big) + \mathcal{I}\Big(\int_{d}^{\tau_j+d} \delta^*(t)\mathrm{d}t \Big) \ge 2 \cdot \mathcal{I}\Big(\int_{d}^{\frac{\tau_i+\tau_j}{2}+d} \delta^*(t)\mathrm{d}t \Big).$$

Without loss of generality, suppose $\tau_i \ge \tau_j$. Since $\dfrac{\mathrm{d}\mathcal{I}\Big(\int_d^{\tau} \delta^*(t)\, \mathrm{d}t \Big)}{\mathrm{d}\tau}$ is non-decreasing, we have

$$\begin{aligned} &\mathcal{I}\Big(\int_{d}^{\tau_i+d} \delta^*(t)\, \mathrm{d}t \Big) - \mathcal{I}\Big(\int_{d}^{\frac{\tau_i+\tau_j}{2}+d} \delta^*(t)\, \mathrm{d}t \Big) \\ &\ge \frac{\tau_i - \tau_j}{2} \cdot \frac{\mathrm{d}\mathcal{I}\Big(\int_d^{\tau} \delta^*(t)\, \mathrm{d}t \Big)}{\mathrm{d}\tau}\Big|_{\tau = \frac{\tau_i+\tau_j}{2}+d} \\ &\ge \mathcal{I}\Big(\int_{d}^{\frac{\tau_i+\tau_j}{2}+d} \delta^*(t)\, \mathrm{d}t \Big) - \mathcal{I}\Big(\int_{d}^{\tau_j+d} \delta^*(t)\, \mathrm{d}t \Big). \end{aligned}$$

Thus,

$$\mathcal{I}\Big(\int_{d}^{\tau_i+d} \delta^*(t)\mathrm{d}t \Big) + \mathcal{I}\Big(\int_{d}^{\tau_j+d} \delta^*(t)\mathrm{d}t \Big) \ge 2 \cdot \mathcal{I}\Big(\int_{d}^{\frac{\tau_i+\tau_j}{2}+d} \delta^*(t)\mathrm{d}t \Big).$$

This implies that balancing two different inter-update periods τ_i and τ_j never increases the total impact of time-space inconsistency. Therefore, the total impact of time-space inconsistency is minimized when $\tau_0 = \tau_1 = \cdots = \tau_k$.

Since $\mathcal{I}(\cdot)$ is non-decreasing, minimizing the highest impact of all situations is equivalent to minimizing the highest time-space inconsistency of them which is given by

$$\max_{0 \le h \le k} \int_{d}^{\tau_h+d} \delta^*(t)\, \mathrm{d}t.$$

Suppose that the situation of the highest time-space inconsistency is produced between two updates at t_i and t_{i+1}, i.e.,

$$\int_{d}^{\tau_i+d} \delta^*(t)\, \mathrm{d}t = \max_{0 \le h \le k} \int_{d}^{\tau_h+d} \delta^*(t)\, \mathrm{d}t.$$

It follows that τ_i must be the longest inter-update period among $\tau_0, \tau_1, \cdots, \tau_k$. Consider any other inter-update period $\tau_j \le \tau_i$. Since $\delta^*(t) \ge 0$, we have

$$\int_{d}^{\tau_j+d} \delta^*(t)\, \mathrm{d}t \le \int_{d}^{\frac{\tau_i+\tau_j}{2}+d} \delta^*(t)\, \mathrm{d}t \le \int_{d}^{\tau_i+d} \delta^*(t)\, \mathrm{d}t.$$

Therefore,

$$\int_{d}^{\frac{\tau_i+\tau_j}{2}+d} \delta^*(t)\, \mathrm{d}t \le \max\Big(\int_{d}^{\tau_j+d} \delta^*(t)\, \mathrm{d}t, \int_{d}^{\tau_i+d} \delta^*(t)\, \mathrm{d}t \Big).$$

This implies that balancing τ_i with another inter-update period τ_j never increases the highest time-space inconsistency of all situations. Therefore, the highest impact of all situations is minimized when $\tau_0 = \tau_1 = \cdots = \tau_k$.

Hence, the theorem is proven. □

Now, we investigate the optimal update schedules for a set of n entities. Assume that the spatial difference between entity i's positions in the server and client views follows a non-decreasing function $\delta_i(\cdot)$ of the time elapsed since its last update. Denote by $\delta_i^*(\cdot)$ the function

resulting from filtering out indiscernible difference in $\delta_i(\cdot)$. The result of Theorem 1 simplifies the problem of finding the optimal update schedules to determining the inter-update periods of the entities. Let the inter-update period of entity i be p_i. Then, entity i's time-space inconsistency between the receipt of two successive updates at the client is given by

$$\int_d^{p_i+d} \delta_i^*(t)\,\mathrm{d}t.$$

Over a sufficiently long period T, the number of updates for entity i can be approximated by $\frac{T}{p_i}$. Thus, the total impact of time-space inconsistency for entity i over the period is

$$\frac{T}{p_i}\cdot\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big).$$

Therefore, the total impact of time-space inconsistency for all entities is

$$\sum_{i=1}^{n}\frac{T}{p_i}\cdot\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big). \tag{1}$$

Recall that the server can update only c entities with the client at each frame. Let f be the length of a frame. Then, p_i's are constrained by

$$\sum_{i=1}^{n}\frac{T}{p_i}=c\cdot\frac{T}{f}. \tag{2}$$

Therefore, the scheduling problem for objective $\mathcal{O}1$ is to minimize (1) subject to constraint (2). This is a constrained-minimum problem. It can be solved by using the Lagrange multiplier theorem.

Theorem 2: The total impact of time-space inconsistency (1) is minimized when every pair of inter-update periods p_i and p_j $(1\le i,j\le n)$ satisfy

$$\begin{aligned}&p_i\cdot\mathcal{I}'\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)\cdot\delta_i^*(p_i+d)-\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)\\ =\;&p_j\cdot\mathcal{I}'\Big(\int_d^{p_j+d}\delta_i^*(t)\,\mathrm{d}t\Big)\cdot\delta_j^*(p_j+d)-\mathcal{I}\Big(\int_d^{p_j+d}\delta_j^*(t)\,\mathrm{d}t\Big),\end{aligned}$$

where $\mathcal{I}'(\cdot)$ is the first-order derivative of $\mathcal{I}(\cdot)$.

Proof: Since

$$\sum_{i=1}^{n}\frac{T}{p_i}-c\cdot\frac{T}{f}=0,$$

to minimize (1), it is equivalent to minimize

$$\zeta=\sum_{i=1}^{n}\frac{T}{p_i}\cdot\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)+\gamma\cdot\Big(\sum_{i=1}^{n}\frac{T}{p_i}-c\cdot\frac{T}{f}\Big),$$

where γ can be any constant value.

The minimum or maximum value of ζ is reached when for every $1\le i\le n$,

$$\frac{\partial\zeta}{\partial p_i}=0. \tag{3}$$

Since

$$\begin{aligned}\frac{\partial\zeta}{\partial p_i}&=\frac{\mathrm{d}\Big(\frac{T}{p_i}\cdot\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)\Big)}{\mathrm{d}p_i}+\gamma\cdot\Big(-\frac{T}{p_i^2}\Big)\\&=\frac{T}{p_i}\cdot\frac{\mathrm{d}\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i}\\&\quad-\frac{T}{p_i^2}\cdot\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)-\gamma\cdot\frac{T}{p_i^2}\\&=\frac{T}{p_i}\cdot\mathcal{I}'\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)\cdot\delta_i^*(p_i+d)\\&\quad-\frac{T}{p_i^2}\cdot\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)-\gamma\cdot\frac{T}{p_i^2}\\&=\frac{T}{p_i^2}\Big(p_i\cdot\mathcal{I}'\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)\cdot\delta_i^*(p_i+d)\\&\quad-\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)-\gamma\Big),\end{aligned}$$

(3) implies that

$$p_i\cdot\mathcal{I}'\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)\cdot\delta_i^*(p_i+d)-\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)$$

is a constant.

When (3) is satisfied, we have

$$\begin{aligned}\frac{\partial^2\zeta}{\partial p_i^2}&=\frac{T}{p_i}\cdot\frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i^2}\\&\quad-\frac{T}{p_i^2}\cdot\frac{\mathrm{d}\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i}\\&\quad-\frac{T}{p_i^2}\cdot\frac{\mathrm{d}\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i}\\&\quad+2\cdot\frac{T}{p_i^3}\cdot\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)+2\gamma\cdot\frac{T}{p_i^3}\\&=\frac{T}{p_i}\cdot\frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i^2}\\&\quad-2\cdot\frac{T}{p_i^2}\cdot\mathcal{I}'\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)\cdot\delta_i^*(p_i+d)\\&\quad+2\cdot\frac{T}{p_i^3}\cdot\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)+2\gamma\cdot\frac{T}{p_i^3}\\&=\frac{T}{p_i}\cdot\frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i^2}-\frac{2}{p_i}\cdot\frac{\partial\zeta}{\partial p_i}\\&=\frac{T}{p_i}\cdot\frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i^2}.\end{aligned}$$

Note that

$$\frac{\mathrm{d}\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i} = \frac{\mathrm{d}\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}(p_i+d)}\cdot\frac{\mathrm{d}(p_i+d)}{\mathrm{d}p_i} = \frac{\mathrm{d}\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}(p_i+d)},$$

and thus,

$$\frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i^2} = \frac{\mathrm{d}\Big(\frac{\mathrm{d}\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}p_i}\Big)}{\mathrm{d}p_i}$$
$$= \frac{\mathrm{d}\Big(\frac{\mathrm{d}\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}(p_i+d)}\Big)}{\mathrm{d}(p_i+d)}\cdot\frac{\mathrm{d}(p_i+d)}{\mathrm{d}p_i}$$
$$= \frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}(p_i+d)^2}.$$

Since

$$\frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^{\tau}\delta^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}\tau^2}\geq 0,$$

it follows that

$$\frac{\partial^2\zeta}{\partial p_i^2} = \frac{T}{p_i}\cdot\frac{\mathrm{d}^2\mathcal{I}\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)}{\mathrm{d}(p_i+d)^2}\geq 0.$$

Also note that for all $i\neq j$ $(1\leq i,j\leq n)$,

$$\frac{\partial^2\zeta}{\partial p_i\partial p_j}=0.$$

Therefore, ζ is indeed minimized when (3) is satisfied. Hence, the theorem is proven. □

Since $\mathcal{I}(\cdot)$ is non-decreasing, minimizing the highest impact of all situations is equivalent to minimizing the highest time-space inconsistency of them. Therefore, the scheduling problem for objective $\mathcal{O}2$ is to minimize

$$\max_{1\leq h\leq n}\int_d^{p_h+d}\delta_h^*(t)\,\mathrm{d}t \tag{4}$$

subject to constraint (2). We consider the non-trivial case where for every $1\leq h\leq n$, $\delta_h^*(t)$ does not remain at 0 when $t\to\infty$, i.e., the spatial difference between the entity positions in the server and client views would eventually become discernible if the server does not update the entity position with the client. Thus, all inter-update periods p_i's must be finite.

Theorem 3: The highest time-space inconsistency of all situations (4) is minimized when every pair of inter-update periods p_i and p_j $(1\leq i,j\leq n)$ satisfy

$$\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t=\int_d^{p_j+d}\delta_j^*(t)\,\mathrm{d}t.$$

Proof: Let $(p_1,p_2,\cdots,p_n)$ be a set of inter-update periods minimizing (4). Suppose that in this schedule, the situation of the highest time-space inconsistency is produced by entity i, i.e.,

$$\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t=\max_{1\leq h\leq n}\int_d^{p_h+d}\delta_h^*(t)\,\mathrm{d}t.$$

The theorem is trivial if

$$\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t=0.$$

We consider the non-trivial case where

$$\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t>0.$$

Assume that there exists another entity j such that

$$\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t>\int_d^{p_j+d}\delta_j^*(t)\,\mathrm{d}t. \tag{5}$$

Then, there must exist a p_i' where $0<p_i'<p_i$ such that

$$\int_d^{p_i'+d}\delta_i^*(t)\,\mathrm{d}t=\int_d^{p_j+d}\delta_j^*(t)\,\mathrm{d}t.$$

Let

$$\psi=\frac{1}{p_j}+\frac{1}{p_i}.$$

It is obvious that

$$\frac{1}{\psi-\frac{1}{p_i'}}>p_j,$$

and hence,

$$\int_d^{\frac{1}{\psi-\frac{1}{p_i'}}+d}\delta_j^*(t)\,\mathrm{d}t>\int_d^{p_j+d}\delta_j^*(t)\,\mathrm{d}t=\int_d^{p_i'+d}\delta_i^*(t)\,\mathrm{d}t. \tag{6}$$

Now, consider the function

$$\zeta(x)=\int_d^{\frac{1}{\psi-\frac{1}{x}}+d}\delta_j^*(t)\,\mathrm{d}t-\int_d^{x+d}\delta_i^*(t)\,\mathrm{d}t.$$

It follows from (6) that $\zeta(p_i')>0$. On the other hand, (5) implies that $\zeta(p_i)<0$. Therefore, there must exist a $\widetilde{p}_i$ where $p_i'<\widetilde{p}_i<p_i$ such that $\zeta(\widetilde{p}_i)=0$. As a result,

$$\max\Big(\int_d^{\frac{1}{\psi-\frac{1}{p_i}}+d}\delta_j^*(t)\,\mathrm{d}t,\int_d^{\widetilde{p}_i+d}\delta_i^*(t)\,\mathrm{d}t\Big)$$
$$=\int_d^{\widetilde{p}_i+d}\delta_i^*(t)\,\mathrm{d}t\;<\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t$$
$$=\max\Big(\int_d^{p_i+d}\delta_i^*(t)\,\mathrm{d}t,\int_d^{p_j+d}\delta_j^*(t)\,\mathrm{d}t\Big).$$

This implies that setting the inter-update periods of entities i and j at $\widetilde{p}_i$ and $\frac{1}{\psi-\frac{1}{\widetilde{p}_i}}$ respectively reduces the highest time-space inconsistency of all situations compared to setting them at p_i and p_j, which contradicts the optimality of $(p_1,p_2,\cdots,p_n)$.

Hence, the theorem is proven. □

Theorems 2 and 3 imply that in the optimal update schedule for objective $\mathcal{O}1$, all entities should have the same value of

$$p_i \cdot \mathcal{I}'\Big(\int_d^{p_i+d} \delta_i^*(t)\,\mathrm{d}t\Big) \cdot \delta_i^*(p_i+d) - \mathcal{I}\Big(\int_d^{p_i+d} \delta_i^*(t)\,\mathrm{d}t\Big) \quad (7)$$

and in the optimal update schedule for objective $\mathcal{O}2$, all entities should have the same value of

$$\int_d^{p_i+d} \delta_i^*(t)\,\mathrm{d}t. \quad (8)$$

These results indicate that the optimal update schedules are determined by both the network delay d and the growth of spatial difference $\delta_i(t)$. The latter, in turn, depends on the trajectories as well as the moving speeds of entities.

6 Consistency Aware Update Scheduling Algorithms

In the above analytical results, p_i refers to the inter-update period of entity i. If we replace the inter-update period of an entity by the time elapsed since its last update and consider the entity's behavior since this last update, (7) and (8) can be rewritten as

$$(t-t_L)\cdot\mathcal{I}'\Big(\int_{t_L+d}^{t+d}\Delta^*(x)\,\mathrm{d}x\Big)\cdot\Delta^*(t+d) - \mathcal{I}\Big(\int_{t_L+d}^{t+d}\Delta^*(x)\,\mathrm{d}x\Big), \quad (9)$$

and

$$\int_{t_L+d}^{t+d}\Delta^*(x)\,\mathrm{d}x, \quad (10)$$

where t is the current time, t_L is the last update time before t, d is the network delay, and $\Delta^*(x)$ is the spatial difference at time x filtered by the minimum discernible distance ϵ, i.e.,

$$\Delta^*(x) = \begin{cases} 0 & \text{if } \Delta(x) < \epsilon, \\ \Delta(x) & \text{if } \Delta(x) \geq \epsilon. \end{cases}$$

The analytical results imply that for objective $\mathcal{O}1$ (or objective $\mathcal{O}2$), the update scheduling algorithm should attempt to minimize the differences of (9) (or (10)) over different entities at the times when the server updates them with the client. To do so, the server can select to update the entities with the highest values of (9) (or (10)) at each frame.

To demonstrate these design principles, in this section, we present a number of update scheduling algorithms for improving consistency based on two different impact functions $\mathcal{I}_1(x) = x$ and $\mathcal{I}_2(x) = x^2$. Function $\mathcal{I}_1$ represents that the impact of time-space inconsistency is linearly proportional to time-space inconsistency. Under $\mathcal{I}_1$, objective $\mathcal{O}1$ degenerates to minimizing the total time-space inconsistency, and (9) can be rewritten as

$$(t-t_L)\cdot\Delta^*(t+d) - \int_{t_L+d}^{t+d}\Delta^*(x)\,\mathrm{d}x. \quad (11)$$

In this special case, the impact of time-space inconsistency is proportional to the time-averaged divergence in [19]. The update scheduling algorithm proposed for objective $\mathcal{O}1$ under $\mathcal{I}_1$ shall be called TotalTS.

Function $\mathcal{I}_2$ represents a scenario under which the impact of inconsistency grows superlinearly with time-space inconsistency: situations of large time-space inconsistency have disproportionally greater impacts than situations of small time-space inconsistency. Under $\mathcal{I}_2$, objective $\mathcal{O}1$ degenerates to minimizing the 2-norm of time-space inconsistency over all situations, and (9) can be rewritten as

$$2(t-t_L)\cdot\int_{t_L+d}^{t+d}\Delta^*(x)\,\mathrm{d}x\cdot\Delta^*(t+d) - \Big(\int_{t_L+d}^{t+d}\Delta^*(x)\,\mathrm{d}x\Big)^2. \quad (12)$$

The update scheduling algorithm proposed for objective $\mathcal{O}1$ under $\mathcal{I}_2$ shall be called 2NormTS.

For both impact functions $\mathcal{I}_1$ and $\mathcal{I}_2$, objective $\mathcal{O}2$ remains equivalent to minimizing the highest time-space inconsistency of all situations (i.e., the inconsistency of the most noticeable situation). Thus, the update scheduling algorithm proposed for objective $\mathcal{O}2$ shall be called MaxTS.

The key to the TotalTS, 2NormTS and MaxTS algorithms for reducing time-space inconsistency is the estimation of (11), (12) and (10) respectively for each entity, which basically involves estimating the following two items:

$$\Delta^*(t+d),$$

and

$$\int_{t_L+d}^{t+d}\Delta^*(x)\,\mathrm{d}x.$$

The item $\Delta^*(t+d)$ refers to the expected spatial difference of the entity at d time later, where t is the current time and d is the mean network delay measured between the server and the client. $\Delta^*(t+d)$ is estimated using the same method as that in the SPACE algorithm of Section 4.2. That is, the server maintains, for each entity, the time t_L of the last update and the entity state $\mathbb{S}_L$ in the server's view at the last update. Together with the current state $\mathbb{S}$ of the entity, the expected spatial difference at time $t+d$ is computed as

$$\|\mathcal{F}(\mathbb{S}_L, t+d-t_L), \mathcal{F}(\mathbb{S}, d)\|,$$

which provides an estimate of $\Delta^*(t+d)$ after filtering out indiscernible difference.

The item $\int_{t_L+d}^{t+d}\Delta^*(x)\,\mathrm{d}x$ refers to the time-space inconsistency in the period $[t_L+d, t+d]$, where t_L is the time of the last update and t is the current time. Note that the server refreshes and updates entity positions periodically in frames rather than completely continuously. Let f be the length of a frame. Without loss of generality, assume that the network delay $d = l \cdot f$, where l is an integer. Suppose that the server last updates an entity with the client at the time that is w frames prior to the current frame, i.e., $t = t_L + wf$. Then, we approximate

$\int_{t_L+d}^{t+d} \Delta^*(x)\,\mathrm{d}x$ by

$$f \cdot \sum_{j=l+1}^{w+l} \Delta^*(t_L + jf) \tag{13}$$

As shown in Figure 5, the above summation can be divided into two portions $A + B$, where

$$A = \sum_{j=l+1}^{w} \Delta^*(t_L + jf),$$

and

$$B = \sum_{j=w+1}^{w+l} \Delta^*(t_L + jf) = \sum_{j=1}^{l} \Delta^*(t + jf).$$

Portion A refers to the accumulated spatial difference since l frames after the last update, and portion B refers to the total spatial difference of the next l frames in the future. When $l \geq w$, A is null and equals 0.

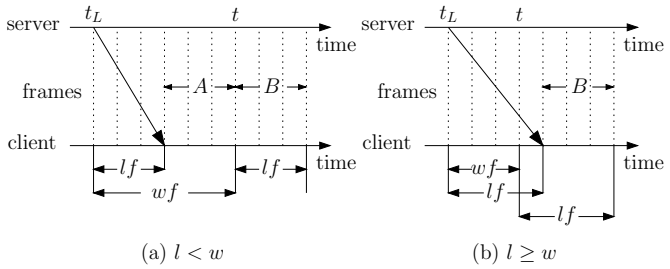


Fig. 5. Division of expression (13).

Portion A can be incrementally maintained by the server. At each frame, if the server updates the entity with the client, it resets A to 0. Otherwise, the server checks the duration since the last update of the entity. If the duration is less than or equal to l frames, A remains 0. If the duration is longer than l frames, A is increased by the spatial difference at the current frame if it is discernible, which is estimated as

$$\|\mathcal{F}(\mathbb{S}_L, t - t_L), \mathcal{F}(\mathbb{S}, 0)\|.$$

Portion B, on the other hand, estimates the spatial difference of the entity in the future. For each $1 \leq j \leq l$, $\Delta^*(t + jf)$ is estimated as

$$\|\mathcal{F}(\mathbb{S}_L, t + jf - t_L), \mathcal{F}(\mathbb{S}, jf)\|,$$

if it is discernible.

In the TotalTS algorithm, at each frame, the server selects to update the top c entities in terms of the estimated value of (11). Figure 6 shows the pseudo code, where the subscripts i in the notations stand for entity i. Similarly, in the 2NormTS and MaxTS algorithms, the server updates with the client the top c entities in terms of the estimated values of (12) and (10) respectively. The pseudo code of the 2NormTS and MaxTS algorithms is omitted due to space limitations.

```
let t be the current time;
for each entity i do {
    refresh its state S_i based on kinetics and user actions;
    if t − t_{L,i} > lf and ‖F(S_{L,i}, t − t_{L,i}), F(S_i, 0)‖ ≥ ε
        set A_i = A_i + ‖F(S_{L,i}, t − t_{L,i}), F(S_i, 0)‖;
    if ‖F(S_{L,i}, t + d − t_{L,i}), F(S_i, d)‖ ≥ ε
        set D_i = (t − t_{L,i}) · ‖F(S_{L,i}, t + d − t_{L,i}), F(S_i, d)‖;
    else
        set D_i = 0;
    set B_i = 0;
    for each 1 ≤ j ≤ l do {
        if ‖F(S_{L,i}, t + jf − t_{L,i}), F(S_i, jf)‖ ≥ ε
            set B_i = B_i + ‖F(S_{L,i}, t + jf − t_{L,i}), F(S_i, jf)‖;
        }
    }
select the top c entities in terms of D_i − A_i − B_i
    and update them with the client;
for each entity i updated with the client do {
    record t_{L,i} = t;
    record S_{L,i} = S_i;
    reset A_i = 0;
    }
```

Fig. 6. TotalTS algorithm.

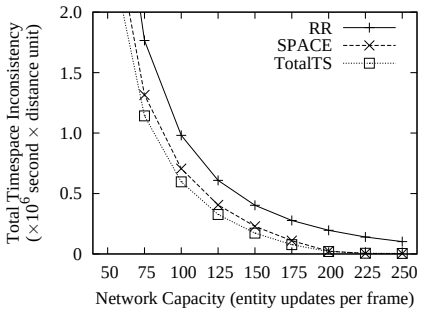
We remark that the TotalTS, 2NormTS and MaxTS algorithms can be tailored to handle scenarios where the entity set of interest to the client change dynamically due to relevance filtering. When a new entity becomes relevant to the client in a frame, the server sends an update to the client immediately for initializing the entity state in the client's view. Meanwhile, the remaining network capacity in the frame is used for updating existing entities with the client. The entities with the highest estimated values of (11), (12) or (10) are updated with the client. On the other hand, when an existing entity is no longer relevant to the client, the server simply stops considering the entity in update scheduling.

7 Experimental Evaluation

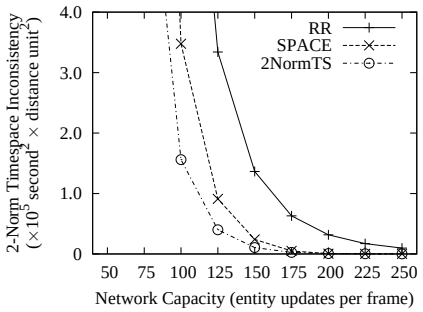
7.1 Experimental Setup

We have conducted a wide range of simulation experiments to evaluate the proposed update scheduling algorithms. The virtual world in our experiments contained a total of 10000 entities moving in a two-dimensional space. We simulated a server that hosts the virtual world and a client that maintains a local view of the virtual world. The length of a frame was set at 0.025 second, i.e., there were 40 frames per second. The network capacity between the server and the client was set to allow the server to update a given number of c entities with the client at each frame out of the 10000 entities simulated. The value of c was varied to produce different levels of network capacity constraints.

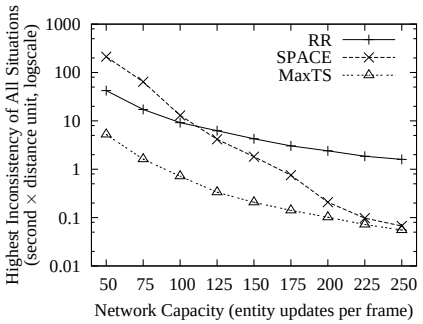
The client implemented the first-order dead reckoning model described in Section 3 to extrapolate entity positions between updates received from the server. To facilitate controlling how actual entity movements differ from the prediction model, the entities were set to move around circles in the virtual world. The radii of the



(a) Total time-space inconsistency.



(b) 2-norm time-space inconsistency.



(c) Highest time-space inconsistency of all situations.

Fig. 7. Time-space inconsistency vs. network capacity.

circles and the rotational speeds of the entities provide two tuning knobs for adjusting the extent to which entity movements at the server differ from the extrapolation model used by the client. Specifically, the radii of the circles determine how the trajectories of the entities deviate from the straight lines predicted by the first-order dead reckoning model. Smaller radii result in more significant deviations. The rotational speeds, on the other hand, determine how fast the entities move along their trajectories. By default, the radii of the circles were randomly assigned from a uniform distribution between 0.1 and 9.9 distance units. The rotational speeds of the entities were randomly assigned from a uniform distribution between 0.01π and 0.09π per second. These distribution parameters were varied widely in our experiments to study the impact of heterogeneity in entity movements (see Section 7.3). Given a radius r and a rotational speed ω, the actual moving speed of an entity is $r \cdot \omega$. During the simulation, we changed the moving speeds of the entities periodically by reassigning their rotational speeds from the uniform distribution once every minute of simulated time. The minimum discernible distance was set at 0.1 distance unit.

In our experiments, the network delays of transmitting messages between the server and the client were modeled by a shifted exponential distribution with a probability density function $f(x) = \lambda e^{-\lambda(x-\mu)}$ $(x \geq \mu)$. This model was shown to well approximate the distribution of packet delays in the Internet [22]. By default, we set $\mu = 0.95 \cdot d$ and $\lambda = 20/d$, where d is the mean network delay. We also tested network delays with larger variance by setting $\mu = 0.5 \cdot d$ and $\lambda = 2/d$ (see Section 7.4). A logical clock mechanism [23] was implemented for the client to filter out-of-order message receipts so that an entity position in the client's view was always extrapolated based on the most recent update message among those received. The default mean network delay d was set at 0.1 second and the performance impact of network delays and their measurement accuracy is investigated in Section 7.4.

We simulated the Round-Robin (RR), SPACE, TotalTS, 2NormTS and MaxTS algorithms for update scheduling.

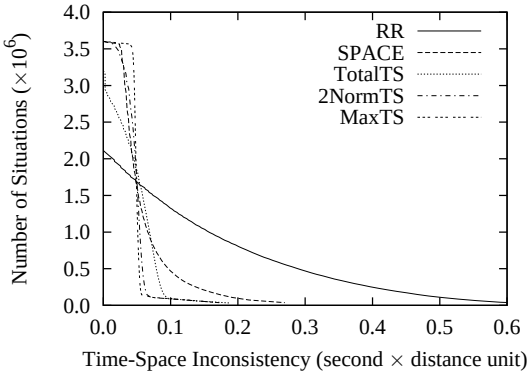


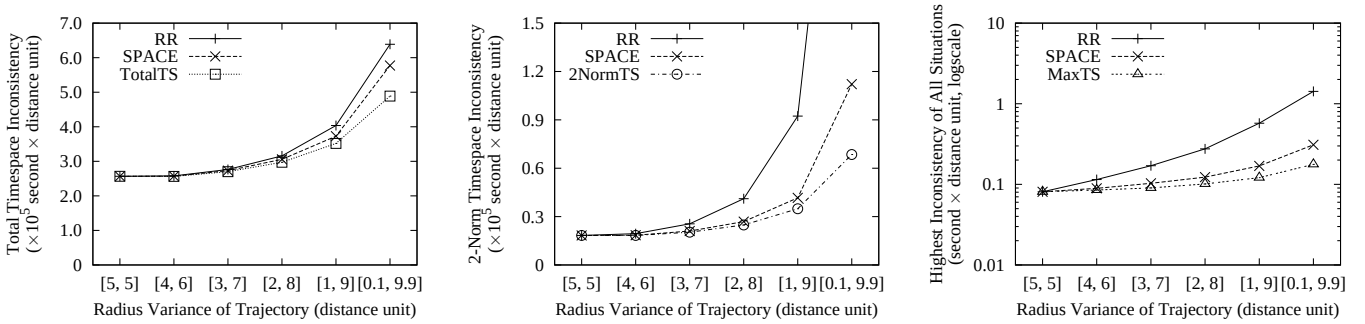
Fig. 8. Distribution of time-space inconsistencies of all situations.

Each experiment run started with synchronized server and client views and simulated the virtual world for a period of 12 minutes. The first 2 minutes of simulated time was considered the warm up period for bringing the system into a relatively steady state. Statistics were collected for the remaining 10 minutes of simulated time. We measured the spatial difference between entity positions in the server and client views at each frame. Time-space inconsistency, as defined in Section 3, was computed over all entities for performance comparison.

7.2 Impact of Network Capacity Constraints

Figure 7 shows time-space inconsistency as a function of the network capacity between the server and the client. As can be seen, when the network capacity allows more than 200 entities to be updated at each frame, the tested scheduling algorithms perform similarly in general. This is because all entities can be updated frequently in this case and thus inconsistency is low. Lower network capacity makes the capacity constraint more serious, thereby increasing time-space inconsistency. Figure 7 shows that the proposed TotalTS, 2NormTS and MaxTS algorithms significantly outperform the RR and SPACE algorithms when the server is allowed to update less than 200 entities with the client at each frame.

To assist in explaining the performance trends, we plot in Figure 8 the distribution of time-space inconsistencies



(a) Total time-space inconsistency. (b) 2-norm time-space inconsistency. (c) Highest time-space inconsistency of all situations.

Fig. 9. Time-space inconsistency vs. radius variance of trajectory.

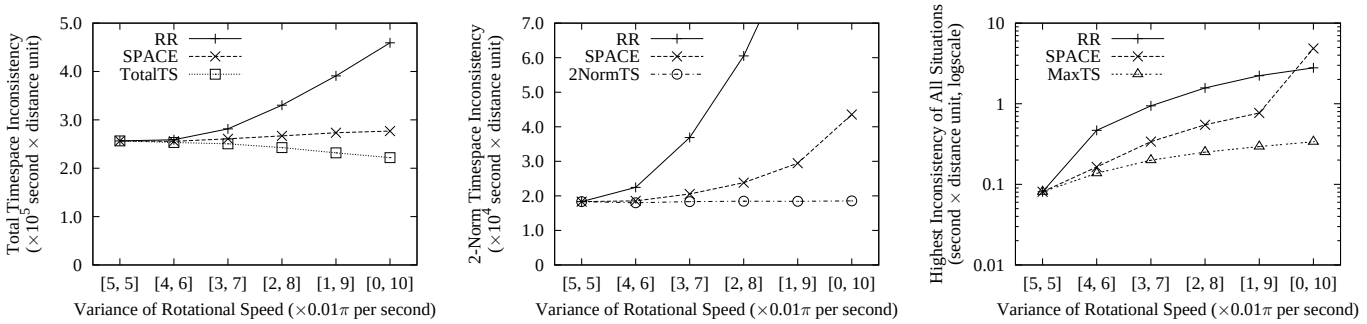
of all situations when the network capacity allows 150 entities to be updated at each frame. A point (x, y) on the curve means that a total of y situations encountered have time-space inconsistency greater than x. Recall that the RR algorithm performs the same number of updates for each entity regardless of their moving speeds and trajectories. Though this helps to cut down the spatial difference of slowly moving entities, a large number of situations with high time-space inconsistency would be generated by fast moving entities. Figure 8 shows that more than 600000 situations have time-space inconsistency greater than 0.25 second $\times$ distance unit. Thus, the RR algorithm shows poor performance in terms of the total, 2-norm and highest time-space inconsistencies of all situations (see Figure 7).

By scheduling updates based on the instantaneous spatial difference of the entities, the SPACE algorithm substantially reduces the number of situations with high time-space inconsistency. Figure 8 shows that only about 50000 situations have time-space inconsistency greater than 0.25 second $\times$ distance unit. However, SPACE does not take into account how long the spatial difference has persisted and will persist. An entity with small spatial difference hardly gets a chance to be updated by SPACE. As a result, the spatial difference may persist for a long period, producing high time-space inconsistency. Therefore, SPACE is not effective in reducing time-space inconsistency. As seen from Figure 7, at a network capacity that allows 150 entities to be updated per frame, the total, 2-norm and highest time-space inconsistencies of all situations in SPACE are 1.33, 2.27 and 8.81 times as much as those in the proposed TotalTS, 2NormTS and MaxTS algorithms respectively. Figure 7(c) also shows that at a severely constrained network capacity of 50 entity updates per frame, SPACE performs even worse than RR by a factor of 5 in terms of the highest time-space inconsistency of all situations (note that the y-axis is in logscale).

Our proposed TotalTS, 2NormTS and MaxTS algorithms integrate temporal and spatial factors in scheduling. Thus, they are able to reduce time-space inconsistency substantially. Figure 8 shows that TotalTS and 2NormTS hardly produce any situation of time-space inconsistency over 0.25 second $\times$ distance unit, and MaxTS completely eliminates situations of time-space inconsistency greater than 0.25 second $\times$ distance unit. Therefore, as shown in Figure 7, TotalTS, 2NormTS and MaxTS cut down time-space inconsistency significantly compared to the RR and SPACE algorithms.

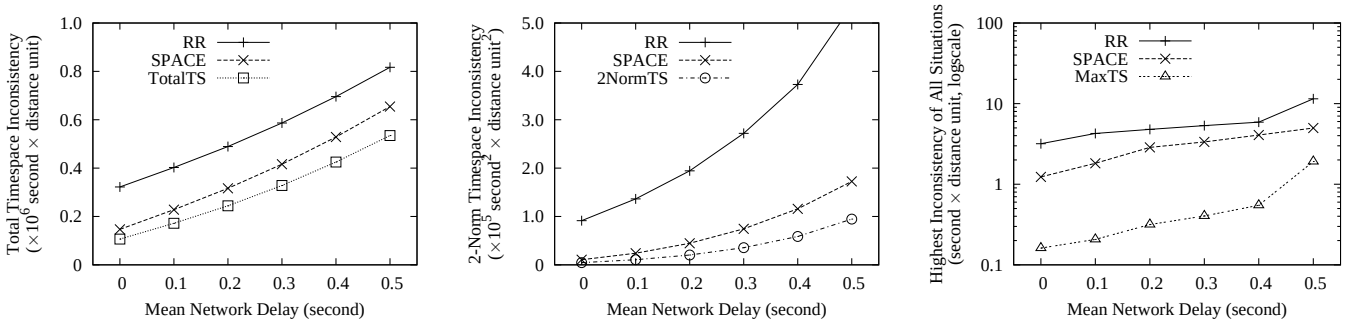
7.3 Impact of Heterogeneity in Entity Movements

We first examine the impact of heterogeneity in the trajectories of the entities. To this end, we set the moving speeds of all entities at 0.25π distance unit per second. The radii of entity trajectories were randomly assigned from uniform distributions over different ranges: [5, 5] (i.e., the trajectories of all entities have the same radius), [4, 6], [3, 7], [2, 8], [1, 9] and [0.1, 9.9] (distance unit). Given a radius r of the trajectory, the rotational speed of an entity was set to $0.25\pi/r$ per second. For example, if the radius is 5 distance units, the rotational speed is then 0.05π per second. Figure 9 shows the performance results for a network capacity that allows 150 entities to be updated per frame. Experimental results with other network capacities have similar performance trends and are not shown here due to space limitations. Since the entities move at the same speed, when the trajectories of all entities have the same radius, entity movements are homogeneous. In this case, as seen from the leftmost points in Figure 9, the RR, SPACE and proposed update scheduling algorithms perform the same because they all update various entities with the same frequency. When the radii of entity trajectories are different, the trajectories deviate, to different extents, from the straight lines predicted by the first-order dead reckoning model. Smaller radii give rise to more significant deviations. As a result, the spatial difference between entity positions in the server and client views grows in different manners for different entities even if the entities move at the same speed. Thus, it is important for the network capacity to be properly allocated to updating different entities for reducing inconsistency. As seen from Figure 9, the proposed TotalTS, 2NormTS and MaxTS algorithms outperform the RR and SPACE algorithms



(a) Total time-space inconsistency. (b) 2-norm time-space inconsistency. (c) Highest time-space inconsistency of all situations.

Fig. 10. Time-space inconsistency vs. variance of rotational speed.



(a) Total time-space inconsistency. (b) 2-norm time-space inconsistency. (c) Highest time-space inconsistency of all situations.

Fig. 11. Time-space inconsistency vs. mean network delay.

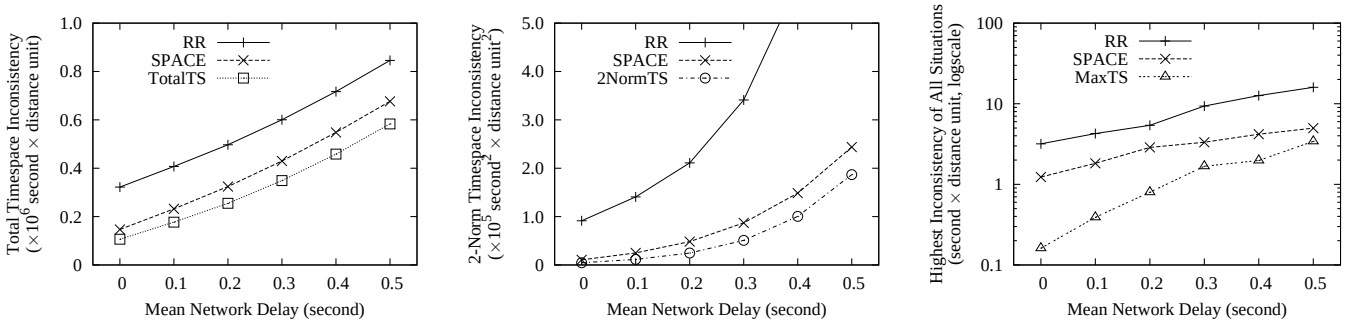
in terms of time-space inconsistency when the radii of entity trajectories are different. The improvement of the proposed algorithms becomes more significant with increasing variance in the radii of entity trajectories.

We then examine the impact of heterogeneity in the moving speeds of the entities. To do so, we set the radii of all entity trajectories at 5 distance units. The rotational speeds of the entities were randomly assigned from uniform distributions over different ranges: $[0.05\pi, 0.05\pi]$ (i.e., all entities move at the same speed), $[0.04\pi, 0.06\pi]$, $[0.03\pi, 0.07\pi]$, $[0.02\pi, 0.08\pi]$, $[0.01\pi, 0.09\pi]$, $[0, 0.1\pi]$ (per second). Figure 10 shows the performance results for a network capacity that allows 150 entities to be updated per frame. It is seen that when all entities move at the same speed (the leftmost points in Figure 10), the RR, SPACE and proposed update scheduling algorithms perform the same since all of them update various entities at the same frequency. When the moving speeds of the entities are different, the spatial difference between entity positions in the server and client views grows at different rates for different entities. If an entity moves faster than others, its spatial difference grows more rapidly. Therefore, it is critical to assign proper update frequencies to different entities for reducing inconsistency. As shown in Figure 10, the proposed TotalTS, 2NormTS and MaxTS algorithms outperform the RR and SPACE algorithms in terms of time-space inconsistency when the entities move at different speeds. The improvement of the proposed algorithms becomes greater with increasing variance in the moving speeds of the entities.

7.4 Impact of Network Delays

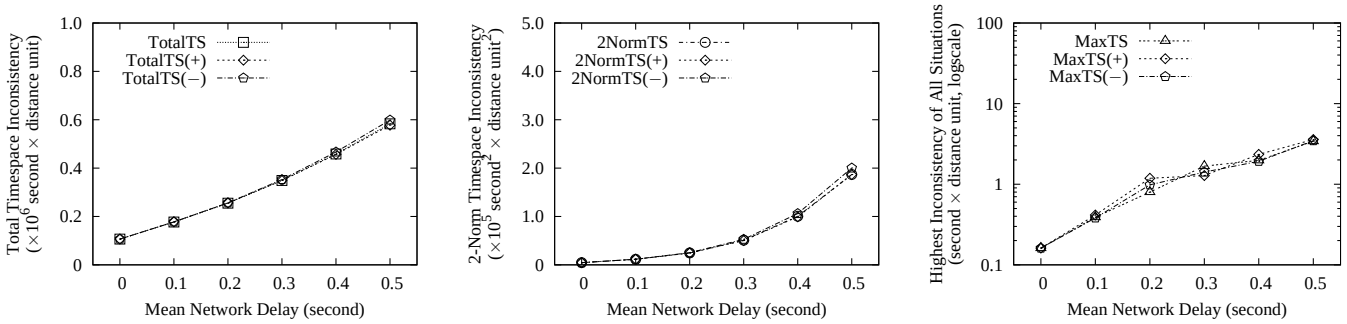
Figure 11 shows time-space inconsistency as a function of mean network delay. Due to space limitations, we report only the experimental results for a default network capacity that allows the server to update 150 entities with the client at each frame. Similar performance trends have been observed in the experimental results with other network capacities.

Due to network delays, when state updates arrive at the client, the actual entity positions at the server would have changed. As a result, time-space inconsistency is affected by network delays and generally increases with the delays. The relative performance of the update scheduling algorithms remains similar over a wide range of network delays. As seen from Figure 11, TotalTS outperforms RR by 35% – 67% and outperforms SPACE by 18% – 28% in terms of total time-space inconsistency; 2NormTS normally outperforms RR and SPACE by an order of magnitude in terms of 2-norm time-space inconsistency; MaxTS outperforms RR by an order of magnitude and outperforms SPACE by about 50% in terms of highest time-space inconsistency of all situations.



(a) Total time-space inconsistency. (b) 2-norm time-space inconsistency. (c) Highest time-space inconsistency of all situations.

Fig. 12. Time-space inconsistency vs. mean network delay (large delay variance).



(a) Total time-space inconsistency. (b) 2-norm time-space inconsistency. (c) Highest time-space inconsistency of all situations.

Fig. 13. Time-space inconsistency vs. mean network delay (large delay variance, imperfect network delay knowledge).

Then, we simulated network delays with larger variance by setting $\mu = 0.5 \cdot d$ and $\lambda = 2/d$ in the delay model, where d is the mean network delay. Figure 12 shows the performance results for different update scheduling algorithms. When the variance of network delays is large, the periods between successive updates received by the client are more diverse, which increases the chance of producing situations with higher time-space inconsistency. Therefore, as seen by comparing Figures 11 and 12, time-space inconsistency generally increases with the variance of network delays. The relative performance of the update scheduling algorithms in Figure 12 is similar to that in Figure 11. The proposed TotalTS, 2NormTS and MaxTS algorithms continue to outperform RR and SPACE even in the presence of large delay variance.

So far, we have assumed that the server has the knowledge of mean network delay to the client. Finally, we study the effect of imperfect network delay knowledge. In this set of experiments, network delays were also set with large variance (i.e., $\mu = 0.5 \cdot d$ and $\lambda = 2/d$). We tested two scenarios in which the mean network delay was overestimated and underestimated relatively by 50% in the proposed TotalTS, 2NormTS and MaxTS algorithms. These two scenarios are identified by "$+$" and "$-$" in Figure 13 (e.g., TotalTS$(+)$/TotalTS$(-)$ denotes the TotalTS algorithm that overestimates/underestimates the mean network delay relatively by 50% in its calculation). As seen from Figure 13, the TotalTS, 2NormTS and MaxTS algorithms are not very sensitive to the inaccuracy in the knowledge of network delays.

8 Conclusion

In this paper, we have investigated update scheduling algorithms for reducing time-space inconsistency in DVEs. Based on the theoretical analysis of optimal update schedules, we have proposed three scheduling algorithms TotalTS, 2NormTS and MaxTS with the objectives of minimizing the total, 2-norm and highest time-space inconsistencies of all situations respectively. These algorithms integrate spatial and temporal factors in scheduling and take into consideration network delays by estimating future inconsistency due to the delays. The algorithms are generic in that they can be used on top of many existing mechanisms such as dead reckoning. Experimental results show that: (1) the proposed algorithms significantly outperform the intuitive algorithms that are based on spatial or temporal factors only; (2) the improvement of the proposed algorithms over the intuitive algorithms generally increases with the heterogeneity in entity movements; (3) the proposed algorithms are not very sensitive to inaccurate estimation of network delays.

ACKNOWLEDGMENTS

This work was supported in part by the Singapore National Research Foundation under Grant NRF2007IDM-IDM002-052.

REFERENCES

[1] J. Dias, R. Galli, A. Almeida, C. Belo and J. Rebordao, "mWorld: A Multiuser 3D Virtual Environment", *IEEE Computer Graphics and Applications*, Vol. 17, No. 2, pp. 55-65, February 1997.

[2] T. Nitta, K. Fujita, and S. Cono, "An Application of Distributed Virtual Environment to Foreign Language Education", *Proc. 30th IEEE Frontiers in Education Conference*, October 2000.

[3] S. Goel and K. Morris, "Dead Reckoning for Aircraft in Distributed Interactive Simulation", *Proc. AIAA Flight Simulation Technology Conference*, 1992.

[4] J. Lui, "Constructing Communication Subgraphs and Deriving an Optimal Synchronization Interval for Distributed Virtual Environment Systems", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 13, No. 5, pp. 778-792, Sep/Oct 2001.

[5] M. Bassiouni, M. Chiu, M.Loper, and M. Garnsey, "Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation", *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 3, pp. 293-331, July 1997.

[6] D. Miller and J. Thorpe, "SIMNET: The Advant of Simulator Networking", *Proceedings of the IEEE*, Vol. 83, No. 8, pp. 1114-1123, August 1995.

[7] S. Singhal and D. Cheriton, "Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality", *Presence: Teleoperators and Virtual Environments*, Vol. 4, No. 2, pp. 169-193, 1995.

[8] K. Morse, L. Bic, and M. Dillencourt, "Interest Management in Large-Scale Virtual Environments", *Presence: Teleoperators and Virtual Environments*, Vol. 9, No. 1, pp. 52-68, Winter 2000.

[9] K. Gummadi, S. Saroiu, and S. Gribble, "King: Estimating Latency between Arbitrary Internet End Hosts", *Proc. 2nd ACM SIGCOMM Workshop on Internet Measurment*, pp. 5-18, November 2002.

[10] S. Singhal and M. Zyda, "Networked Virtual Environments: Design and Implementation", Addison-Wesley, Reading, MA, 1999.

[11] S. Zhou, W. Cai, B.-S. Lee, and S. Turner, "Time-Space Consistency in Large-Scale Distributed Virtual Environments", *ACM Transactions on Modeling and Computer Simulation*, Vol. 14, No. 1, pp. 31-47, January 2004.

[12] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet", *IEEE Network*, Vol. 13, No. 4, pp. 6-15, April 1999.

[13] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-Peer Support for Massively Multiplayer Games", *Proc. IEEE INFOCOM'04*, March 2004.

[14] R. Waters and J. Barrus, "The Rise of Shared Virtual Environments", *IEEE Spectrum*, Vol. 34, No. 3, pp. 20-25, March 1997.

[15] J. Lui and M.F. Chan, "An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, No. 3, pp. 193-211, March 2001.

[16] P. Morillo, J. Orduna, M. Fernandez, and J. Duato, "Improving the Performance of Distributed Virtual Environment Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, No. 7, pp. 637-649, July 2005.

[17] B. Ng, R. Lau, A. Si, and F. Li, "Multiserver Support for Large-Scale Distributed Virtual Environments", *IEEE Transactions on Multimedia*, Vol. 7, No. 6, pp. 1054-1065, December 2005.

[18] A. Labrinidis and N. Roussopoulos, "Update Propagation Strategies for Improving the Quality of Data on the Web", *Proc. VLDB'01*, September 2001.

[19] C. Olston and J. Widom, "Best-Effort Cache Synchronization with Source Cooperation", *Proc. ACM SIGMOD'02*, June 2002.

[20] C. Faisstnauer, D. Schmalstieg, and W. Purgathofer, "Priority Scheduling for Networked Virtual Environments", *IEEE Computer Graphics and Applications*, Vol. 20, No. 6, pp. 66-75, Nov/Dec 2000.

[21] Y. Yu, Z. Li, L. Shi, Y.-C. Chen, and H. Xu, "Network-Aware State Update for Large Scale Mobile Games", *Proc. IEEE ICCCN'07*, August 2007.

[22] A. Corlett, D. I. Pullin, and S. Sargood, "Statistics of One-Way Internet Packet Delays", *Proc. 53rd IETF*, March 2002.

[23] C. Fidge, "Logical Time in Distributed Computing Systems", *IEEE Computer*, Vol. 24, No. 8, pp. 28-33, August 1991.

[24] D. Ta and S. Zhou, "Efficient Client-to-Server Assignments for Distributed Virtual Environments", *Proc. IEEE IPDPS'06*, April 2006.

[25] QStat – Real-Time Game Server Statistics, http://www.qstat.org/, 2008.