Constructing Communication Subgraphs and Deriving an Optimal Synchronization Interval for Distributed Virtual Environment Systems*

John C.S. Lui
Department of Computer Science & Engineering
The Chinese University of Hong Kong
Shatin, NT
Email:cslui@cse.cuhk.edu.hk

Abstract

In this paper, we consider problems of constructing a communication subgraph and deriving an optimal synchronization interval for a distributed virtual environment system (DVE). In general, a DVE system is a distributed system which allows many clients who are located in different parts of the network to concurrently explore and interact with each other under a high resolution, 3-dimensional, graphical virtual environment. Each client in a DVE system is represented by an avatar in the virtual environment, and each avatar can move and interact freely in the virtual environment. There are many challenging issues in designing a cost-effective DVE system. In this work, we address two important design issues, namely, (a) how to construct a communication subgraph which can efficiently carry traffic generated by all clients in a DVE system and, (b) how to guarantee that each participating client has the same consistent view of the virtual world. In other words, if there is an action taken by an avatar or if there is any change in the state of an object in the virtual world, every participating client will be able to view the change. To provide this consistent view, a DVE system needs to perform synchronization actions periodically. We present several algorithms for constructing a communication subgraph. In the subgraph construction, we try to reduce the consumption of network bandwidth resources or reduce the maximum delay between any two clients in a DVE system. Based on a given communication subgraph, we then derive the optimal synchronization interval so as to guarantee the view consistency among all participating clients. The derivation of the optimal synchronization interval is based on the theory of Markov chains and the fundamental matrix.

Keywords: distributed virtual environment, optimal synchronization interval, Markov chains, fundamental matrix.

*This work is supported in part by the CUHK the RGC Research Grant CUHK4220/01E.

1 Introduction

Recent advances in computer graphics, multimedia systems, parallel/distributed systems and high speed networking technologies enable computer scientists and engineers to build distributed virtual environment (DVE) systems. A DVE is a distributed system that allows many clients who are located in different parts of the network to concurrently explore and interact with each other under a high-resolution, 3-dimensional, graphical virtual environment (or virtual world). Clients who are exploring the virtual environment can (1) extract relevant information about the virtual environment (e.g., by sending database queries to the DVE system and inquiring about the state of any object in the virtual environment), and (2) communicate in real time with other clients who are also exploring the same virtual environment. Like any computer technology, DVE will change the way people work, interact, and share information.

To see how this may affect daily life and business operation, let us consider the following scenario. A group of people, who are located in different parts of the world, need to hold a business meeting to discuss the design and finance issues of a new high-rise office complex. Instead of requiring that all these people meet in a particular place, the meeting will be held under a DVE system. Under this DVE system, those participating in the meeting can interact in a virtual world of the new high-rise office complex that they are proposing to build. In this virtual environment, each participant of the meeting is represented by a 3D object, which is known as an *avatar*. These avatars can walk around in this virtual office building, and in the process, rearrange any 3D object (e.g., rearranging furniture and paintings, or selecting different kinds of wallpaper) in the virtual environment. Any change to a 3D object in this virtual environment will be *visible* to all participants. Also, participants will be able to interact with each other in real time, and they will be able to inquire and receive relevant information about the virtual world that they are exploring.

To design a cost-effective, scalable DVE system, many research issues need to be addressed. For example:

- Designing an efficient backend database engine which can provide high throughput and low response time for any relational, spatial, and temporal query submitted by a client, who wants to know some relevant information about the virtual environment that he/she is exploring.
- Designing an efficient communication protocol which has low network bandwidth consump-

tion and low communication delay for all users in a DVE system.

- Ensuring that each user has a consistent view of the virtual world. In other words, if there is an action taken by any user in the virtual world or if the state of any object in the virtual world is changed, every participating client should be able to view the change either immediately, or with a small and acceptable delay. In order to provide this consistent view, a DVE system needs to periodically perform some form of *synchronization* action.

The first of these research issues (the design of a backend database engine which can efficiently process relational, spatial, and temporal queries submitted by users) was presented in [5]. The authors demonstrated a prototype system known as VINCENT, which can support various database queries for DVE applications. For the second research issue, one can utilize some of the recent work on Internet real time protocols, such as the RSVP[15] and RTP[11], to design a communication protocol for a DVE system that allocates an appropriate amount of network resources for realtime communication. In this paper, we address the third research issue, that is, deriving the optimal synchronization interval so that every client will have a consistent view of the virtual world. Our work is divided into two parts. In the first part, three algorithms are presented for constructing a communication subgraph between all clients in a DVE system. Depending on the network environment as well as the degree of dynamics of the user's membership, we can construct a communication subgraph which has low network bandwidth resource consumption, or a communication subgraph that has a minimum maximal delay between any two clients in a DVE system. In the second part of this paper, we derive the optimal synchronization interval based on the communication subgraph which we constructed. We present the mathematical model and illustrate how we can derive the optimal synchronization interval using the theory of Markov chains and properties of the fundamental matrix[2, 8].

We start with a brief review of the published literature on DVE systems. In [14], the authors describe many interesting DVE applications, including a prototype system called the Diamond Park, which is a scalable DVE system for a large networking environment. In [10], the authors describe some possible architectures for DVE systems, as well as certain techniques used to reduce the amount of traffic sent across the network, such as the *dead-reckoning* and the *area-of-interest (AOI)* approaches. In [3, 7], the authors illustrate how to design and implement a storage server for different classes of multimedia applications such as video-on-demand and virtual walk-through. This storage server can be used as a basic building block of the DVE applications that we men-

tioned. One example of the virtual world in [13] is a model of the city of Los Angeles, where clients can freely traverse from one location to another location. One limitation of this system, however, is that it only allows one user in its virtual environment at any one time (although different users may explore the same virtual world but under different sessions). Issues of designing a backend database for a DVE system are described in [5], where a backend database engine and query processing techniques are proposed so as to support spatial and relational queries submitted by clients who are exploring the virtual world. In [6], the authors describe and solve the workload partitioning and processor assignment problems for a large scale DVE system.

The outline of the paper is as follows. In Section 2, we present the system model of a DVE system, as well as some synchronization techniques to reduce the synchronization update overhead. In Section 3, we present three algorithms for constructing a communication subgraph for a DVE system. In Section 4, we present our synchronization model and derive the optimal synchronization interval. Experiments are carried out in Section 5 to illustrate the network bandwidth consumption, reduction of transmission delay and the cost of synchronization. Finally, conclusions are given in Section 6.

2 System Model

In this section, we describe the model of a DVE system. Specifically, we formulate the problem of finding a communication subgraph for all participating clients in a DVE system. We also describe various synchronization techniques as well as some approaches to reduce the amount of communication traffic for object synchronization. In Appendix I, we list the common notation we use in this paper.

Since a DVE system can be deployed in a LAN, a private network or in a WAN environment (such as the Internet), we use graph theory notation to describe the underlying general network setting. The network is represented by a connected graph $G = (V, E)$, where V denotes the set of nodes of the network. Assume that $|V| = n$ such that $V = \{v_1, v_2, \ldots, v_n\}$, where v_i is a router or an individual computer that participates in the same DVE session. Letting $\mathcal{V}$ be the set of computers that participate in a DVE session, we have $\mathcal{V} \subseteq V$ and $|\mathcal{V}| = m \leq n$. Let E be the set of edges in the graph G. An edge $e_{i,j} \in E$ is a communication link between v_i,$v_j \in V$. For each communication link $e_{i,j}$, there is an associated upper bound on transmission delay, which is

denoted by $d(e_{i,j}) \geq 0$. One special case occurs when all nodes in V are participating computers in the same LAN. In this case, the graph G is a fully connected graph with $n * (n-1)$ edges, and all these edges have the same upper bound on transmission delay.

Assume that at time t, there are k (where $k \leq m$) clients who are exploring the same virtual environment. Let $C(t) \subseteq \mathcal{V}$ denote the set of these participating DVE clients at time t. It is important to point out that the system model allows a DVE system to have the *dynamic membership property*, that is, the number of participating clients in the same virtual environment can change at any time (e.g., clients can join or leave the virtual world session at any time). Note that this model allows us to present DVE applications which have the inherent dynamic membership characteristic. For example, clients may want to login to a virtual world session and participate in Internet shopping and then leave the session after the transaction is finished. For other DVE applications, membership can be static, for example, the business meeting describe in Section 1.

Given a graph $G = (V, E)$, we need to derive a communication subgraph $G'(t)$. In general, a communication subgraph $G'(t) = (C(t), E')$ is a connected graph where $C(t) \subseteq \mathcal{V}$ and $E' \subseteq E$. All data and control message will be carried on this communication subgraph. Therefore, a given subgraph $G'(t)$ dictates the network bandwidth consumption, as well as the delay between any pair of participating clients. Note that, in general, there can be a large number of possible subgraphs $G'(t)$. Depending on whether a DVE application is operated on a LAN or on a WAN environment and whether a DVE application has the dynamic membership property, we may want to construct a communication subgraph so as to reduce the total network bandwidth consumption or to reduce the maximum delay between any two clients. In Section 3, we propose some approaches for constructing such a communication subgraph $G'(t)$ based on the underlying network (LAN or WAN) as well as the dynamic membership characteristics of a DVE application.

Let $G^{'}(t) = \{C(t), E'\}$ denote a derived communication subgraph(details of the subgraph construction will be presented in Section 3). Let let $p_{i,j}$ be a loop-free shortest path between node v_i and v_j where $v_i, v_j \in C(t)$, and $d(p_{i,j})$ be the transmission delay for the path $p_{i,j}$. In other words, if $p_{i,j} = (e_{i,a}, e_{a,b}, \cdots, e_{l,j})$, then $d(p_{i,j}) = d(e_{i,a}) + d(e_{a,b}) + \cdots + d(e_{l,j})$. Letting $d_{max}(G^{'}(t))$ be the maximum delay for the subgraph $G^{'}(t)$, we have

$$d_{max}(G^{'}(t)) \quad = \quad \max_{v_i, v_j \in C(t)} d(p_{i,j}) \tag{1}$$

Once a communication subgraph $G^{'}(t)$ is given, the maximum delay $d_{max}(G^{'}(t))$ between any two clients can be determined by Equation (1). Given the communication subgraph $G^{'}(t)$, we can address the issue of how to maintain a consistent view for every client in the virtual world.

In general, a virtual world is populated by many objects. For example, a piece of furniture in a virtual environment is one kind of object. An avatar, which is a representation of a client in the virtual environment, is another type of object. Each object has a set of attributes, and each attribute has a set of values. One possible attribute is the object's coordinates in the virtual world. Another possible attribute is the directional velocity of a moving object. The *state* of an object at time t is defined by values of the object's attributes at time t. In general, objects in a virtual world can be classified into two categories, *static* and *dynamic*. The values of the attributes of a static object are fixed and do not vary in time. For example, a mountain has its fixed coordinates in the virtual world. The values of the attributes of a dynamic object can be time varying. For instance, a dynamic object like an avatar may traverse from one region to another region in the virtual world, and therefore its coordinates change accordingly. For a given object o_i, let $\mathcal{S}_{o_i} = \{s^1_{o_i}, s^2_{o_i}, \ldots, \}$ be the set representing all possible states of object o_i in a DVE system.

Since all clients who are exploring a virtual world should share the same view, it is important that they should have a *consistent view* of all objects in the virtual world. For example, if a client C_A views an object o_i which is in state $s^*_{o_i}$ (e.g., the coordinates of o_i are $[x, y, z]$), then another client C_B should also view object o_i in state $s^*_{o_i}$ (e.g, object o_i should be at the same coordinates $[x, y, z]$ when seen by C_B). To *visualize* a dynamic object, let $s_{o_i}(t)$ be the state of object o_i at time t. If client c wants to view the dynamic object o_i, then the computer at which client c resides has to *render a sequence of state changes* of object o_i, starting at the same time t. To maintain a consistent view for all clients in a virtual world, this implies that every computer at which clients reside needs to *render* the same sequence of state changes, starting from the same time t. Note that, in general, an absolutely consistent view of a dynamic object may not be possible due to the following reasons:

- Different computers may render the sequence of state changes at different rates. This is due to the fact that different computers may have different workloads and/or different processing power.
- Not all clients can start the rendering of state sequences at the same time. This may be

due to the variation of network delay in delivering the "start-rendering" message to all clients. The purpose of the start-rendering message is that upon receiving this message, each participating client should update the state of the corresponding object.

Let us now give the definition of object synchronization and phase difference.

Definition 1 *Let $c_{o_i}^j(t)$ be the state of object o_i that client C_j is viewing at time t. If $c_{o_i}^l(t) = s_{o_i}^k$ and $c_{o_i}^m(t+\tau) = s_{o_i}^k$, then we say that the phase difference between client C_l and client C_m, denoted by $\Phi(l, m)$, is τ.*

For example, assume that the set of all possible states of object o_i is $\mathcal{S}_{o_i} = \{s_{o_i}^1, s_{o_i}^2, \ldots, s_{o_i}^{10}\}$ and the rendering of a sequence of state changes of object o_i is $Seq = \{s_{o_i}^1, s_{o_i}^3, s_{o_i}^5, s_{o_i}^7, s_{o_i}^9\}$. If client C_l starts the rendering sequence at time 0 and client C_m starts the same rendering sequence at time 3, then the phase difference $\Phi(l, m) = 3$. The *object synchronization procedure* in a DVE system is a process to guarantee that the *absolute* value of the phase difference between any clients and any object in a virtual world is less than or equal to a pre-defined system threshold Φ.

Some of the approaches to maintain object synchronization are:

- **Aggressive Object Synchronization Technique (AOS):**
  Under the AOS approach, each object o_i is assigned to a master process P_{o_i} in a DVE system. For every period with length δt, the process P_{o_i} will broadcast the state of the object o_i to all participating clients in a DVE system. Upon receiving the state information of object o_i, every client will render the state of o_i. Figure 1 illustrates the AOS technique wherein the master process P_{o_i} sends the start-rendering message to all participating clients, then periodically sends a synchronization-message to all participating clients. A major disadvantage of this approach is that there is the resulting high volume of synchronization message traffic, and this leads to a scalability problem in a DVE system. Note that even though the process P_{o_i} broadcasts a synchronization message every δt time units, the synchronization can still be off by $d_{max}(G'(t))$ time units due to the network delay of transmitting the synchronization message.

- **Dead Reckoning Synchronization Technique (DRS):**
  In the AOS technique, extremely high network bandwidth will be consumed by sending the

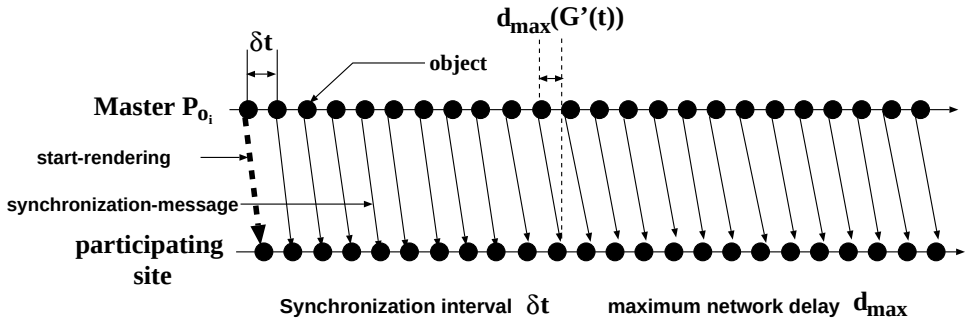


Figure 1: Aggressive Object Synchronization (AOS) Technique with $d_{max}(G')$ being the maximum network delay

synchronization-message. However, it is important to observe that in most cases, the state of an object at the current time step is very *similar* to the state of the object in the previous time step. Therefore, it is not necessary for the process P_{o_i} to send a synchronization message every δt time units. Consider the following example: if a ball in the virtual environment is undergoing a free fall from a 1,000 foot tall building and if the initial position of the ball is known to all clients, then every participating client should be able to calculate the *trajectory* of this ball and render the corresponding sequence of state changes. That is, each participating machine renders the object's coordinates for every time step τ and updates the object's state in its local machine. This is the main principle behind the *dead-reckoning synchronization* (DRS) technique [12]. Under the DRS technique, the initial state as well as the equation governing the trajectory of the object are available to all participating clients. Each of the participating client's machine performs the local computation and updates its local display. The DRS technique is illustrated in Figure 2, where the start-rendering message encodes the object's initial position and directional velocity.

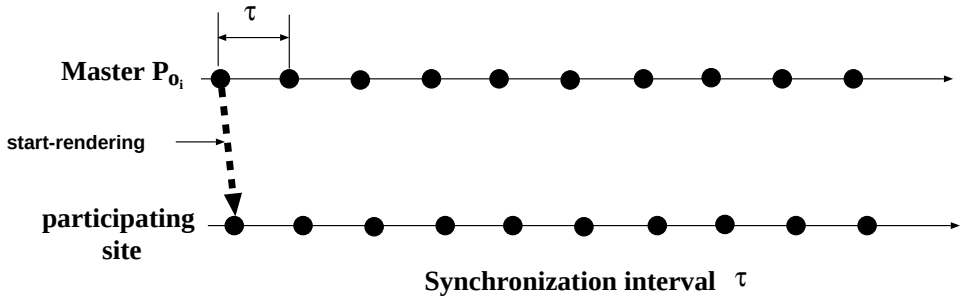


Figure 2: Dead Reckoning Synchronization Technique for which only the start-rending message is sent

- **Dead Reckoning with Periodic Synchronization Message (DRPS):**

  Note that even if the DRS technique is used, it is still necessary for the master process P_{o_i} of object o_i to send a synchronization message from time to time. This situation occurs if there is an external event that will affect the trajectory of the object o_i. For example, a participating client catches the free-falling ball before it reaches the floor. Another reason for sending a synchronization message periodically is the fluctuation of workload at participating computers. Some computers may not be able to render the next state of the object on time, and if this is not adjusted, it will create a view inconsistency problem in the long run. Therefore, the DRPS technique employs the dead-reckoning synchronization technique and every synchronization intervals with length $n\tau$, a "synchronization-message" is sent. Upon receiving the synchronization message, each computer will immediately update the state of the object (or, in effect, reset the phase difference to zero). The DRPS technique is depicted in Figure 3 (with $d_{max}(G'(t))$ being the maximum network delay). Note that, in general, the larger the

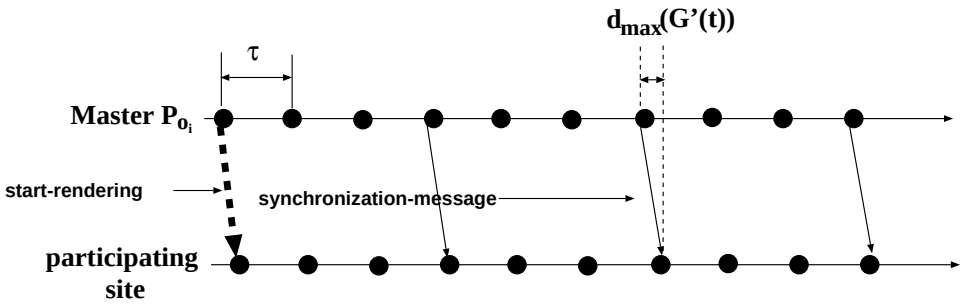


Figure 3: Dead Reckoning Synchronization Technique with Synchronization Message

value of the synchronization interval (e.g., $n\tau$), the larger the degree of the phase difference between any two clients. Thus for large enough n the view consistency between these two clients may become unacceptable. On the other hand, if the synchronization message is sent too often (equivalently, the value of the synchronization interval is too small), a DVE system needs to send many synchronization messages to all participating clients and therefore consumes more network bandwidth. In Section 4, we illustrate how to derive an optimal synchronization interval.

3 Construction of communication subgraph

In this section, we first describe various issues which affect design choices of communication subgraphs. Depending on the performance criteria, we present several algorithms for constructing a communication subgraph. Based on the derived communication subgraph, we can obtain the maximum end-to-end delay between any two clients in a DVE system. We then use this maximum end-to-end delay as an input parameter for maintaining object synchronization.

As we have discussed in Section 2, the underlying network is represented by a connected graph $G = (V, E)$. Given the current client set $C(t)$, we need to find a connected subgraph $G'(t) = (V', E')$ such that $C(t) \subseteq V' \subseteq V$ and $E' \subseteq E$ where V' is a set that contains the current client set, and possibly any node (or router) in the graph G that makes the subgraph connected.

Since there are many possible communication subgraphs $G'(t)$, we may want to optimize the subgraph construction based on one of the following metrics:

- minimize the maximum *end-to-end delay* between any two clients in $C(t)$, or
- minimize the total *bandwidth consumption* of the communication subgraph by minimizing the total edge cost of the subgraph where the edge cost represents the propagation delay of the corresponding edge.

The first metric ensures that there is a high degree of object synchronization in a DVE system. As described in the previous section, absolute object synchronization may not be possible due to the non-negligible network delay of sending the "start-rendering" message. Therefore, reducing the maximum end-to-end delay between any two clients implies reducing the phase difference of any object in a DVE system. The second metric ensures that the number of packets in transit across the communication links (or edges) is minimized and therefore reduces the network bandwidth resource consumption. Note that reducing the number of edges in the graph G may not reduce the average number of transit messages because some edges may have a longer propagation delay than another edge. Formally, minimizing the maximum end-to-end delay requires finding a connected subgraph $G^*(t)$ such that:

$$d_{max}(G^*(t)) = \min_{G'(t)} \left\{ \max_{v_i, v_j \in C(t)} \{d(p_{i,j})\} \right\} \quad (2)$$

where $d(p_{i,j})$ is the delay in the communication path between client v_i and client v_j, for $i \neq j$. For the second criteria of subgraph construction, let $N_{G'(t)}(\tau)$ denote the total number of synchronization messages in transit across edges at time τ for the communication subgraph $G'(t)$. In this work, finding a subgraph that minimizes the network bandwidth consumption refers to finding a connected subgraph $G^*(t)$ such that :

$$\lim_{\tau \to \infty} \frac{\int_{x=0}^{\tau} N_{G^*(t)}(x)}{\tau} \quad = \quad \min_{G'(t)} \left\{ \lim_{\tau \to \infty} \frac{\int_{x=0}^{\tau} N_{G'(t)}(x)}{\tau} \right\} \tag{3}$$

where the limit of the above expression represents the average number of total synchronization messages which are in transit across some communication edges in the corresponding subgraph $G'(t)$.

In this paper, we consider two factors that may affect the design metrics of the communication subgraph: 1) the underlying networking environment, and 2) the type of membership of DVE clients. We classify the underlying networking environment into two major classes: LAN and WAN environments. Under a LAN environment, the transmission bandwidth is high and the data transmission is usually reliable and has a small propagation delay. For a DVE system to operate in a LAN environment, the number of participants is usually small. Thus, it is possible to use a centralized algorithm to construct a communication subgraph so as to minimize the maximum delay or to minimize the bandwidth consumption. On the other hand, a WAN is a large networking environment which consists of many routers and sub-networks. The communication delay between any two clients is non-negligible. The transmission bandwidth of a WAN environment, as compared to the transmission bandwidth of a LAN environment, is usually scarce and expensive. Moreover, the number of DVE clients in a WAN environment can range from tens to thousands. Due to the size of the network and the number of possible participating DVE clients, we need a distributed algorithm for the construction of a communication subgraph.

Another factor that influences design choices of communication subgraphs is the type of membership. As discussed in Section 2, we classify the membership of DVE clients into two types, static membership and dynamic membership. Under static membership, a DVE system can construct a communication subgraph before admitting all participating clients. At the end of the DVE session, all resources allocated for a communication subgraph can be deallocated. For dynamic membership, a communication subgraph is created at the beginning of a DVE session. Whenever a client joins or leaves the virtual environment session, a DVE system may have to re-construct a

new communication subgraph. Usually, this construction can be based on the previous subgraph, so that the subgraph construction can be accomplished in a short period of time.

Consideration the above factors for the construction of a communication subgraph, we discuss three subgraph construction algorithms. They are: 1) the minimum diameter subgraph (MDS), 2) the core-based tree (CBT), and 3) the spanning tree (ST).

Minimum diameter subgraph (MDS):

A minimum diameter subgraph (MDS) is a natural choice when we want to construct a multicast communication subgraph, since an MDS provides a guarantee on the delay bound between any two nodes. An MDS ensures that, for every pair of nodes, there exists a path between these two nodes which has a length that is less than or equal to the diameter of the graph. In the context of our DVE application, the diameter corresponds to the maximum delay between any pair of clients. Since every DVE client can be both a sender and a receiver of synchronization messages, as a consequence, the diameter provides a delay bound for all possible synchronization message transmissions. Formally, the diameter of a graph $G = (V, E)$ is

$$\text{diameter}(G) = \max_{v_i, v_j \in V} d(p_{v_i,v_j})$$

where p_{v_i,v_j} is a shortest path between v_i and v_j and $d(p_{v_i,v_j})$ is the delay associated with this path. Therefore, to obtain the diameter of a graph, we have to consider all pairs' shortest paths of a graph and then take the maximum value. One way to find an MDS of a connected graph $G = (V, E)$ is to consider a subset of edges $E' \subseteq E$ that forms a connected subgraph $G' = (V', E')$, where $C(t) \subseteq V'$ and the diameter of G' is equal to the diameter of G. It is important to note that, given the diameter of a graph G, the derived MDS G' may not necessarily be unique.

Aside from finding an MDS of a connected graph G, one interesting question is whether we can find an MDS which also has minimal total edge cost (i.e., find an MDS which is also a minimum spanning tree of the graph G) so as to minimize the network bandwidth consumption. To answer this question, we have the following theorem:

Theorem 1 *Let $G = (V, E)$ be a connected graph, and for each edge $e \in E$ let $d(e) \in \mathbb{R}$ be the edge cost. The problem of finding a spanning subgraph $G' = (V, E')$ for the graph G such that diameter(G') = diameter(G) and the sum of the cost of all edges in E' does not exceed a real number L is NP-complete.*

Proof: Please refer to [9]. ∎

Because the problem is NP-complete, we relax the constraint of minimizing the total edges cost, in order to find a minimum diameter subgraph (MDS) in polynomial time. In general, to find an MDS of a graph G, we perform the following steps:

1. Find all shortest paths between all pairs of vertices in $C(t)$.
2. Take the union of all nodes and edges from these shortest paths and form a new node set V' and new edge set E'.
3. For the new node set V' and edge set E', we form the minimum diameter subgraph $G' = (V', E')$.

The above steps give a general idea of the construction of an MDS from a given connected graph G. Note that for a DVE application, the set of participating clients $C(t)$ may not be equal to V, the set of nodes in the network. Therefore, we want to construct a subgraph $G' = (V', E')$ which satisfies the following:

1. $C(t) \subseteq V'$ and $V' \subseteq V$. The first condition ensures that all participating clients are included in the communication subgraph. The second condition ensures that some nodes, although they are not participating clients, may be included in the subgraph so as to provide the connectivity.
2. To construct a minimum diameter subgraph, we have to ensure that diameter(G') = diameter(G). Therefore, to derive G', we want to minimize the largest delay among all shortest paths between any two clients, that is,

$$\min\{\max_{v_i, v_j \in C(t)} d(p'_{v_i,v_j})\}$$

where p'_{v_i,v_j} is a shortest path between client v_i and client v_j with respect to the subgraph G'.

The algorithm for finding an MDS is given below.

1 $V' \leftarrow \emptyset$
2 $E' \leftarrow \emptyset$
3 **for** each pair of clients $v_i, v_j \in C(t)$ **do**
4 find a shortest path p'_{v_i,v_j} between v_i and v_j
5 where p'_{v_i,v_j} is a set of edges that constitutes the shortest path;
6 **for** each edge $e_s \in p'_{v_i,v_j}$ **do**
7 $E' \leftarrow E' \cup \{e_s\}$
8 **Let** v_{s1} and v_{s2} be two vertices that are the endpoints of the edge e_s
9 $V' \leftarrow V' \cup \{v_{s1}\} \cup \{v_{s2}\}$
10 **end for**
11 **end for**
12 The MDS is $G' = (V', E')$

Theorem 2 *The MDS algorithm given above guarantees that G' is a minimum diameter subgraph.*

Proof: Let us prove the above claim by contradiction. Assume that the constructed subgraph G' based on the above algorithm is not a minimum diameter subgraph and there exists another connected graph G'' which has a smaller diameter. Let p'_{v_i,v_j} be the longest shortest path between node v_i and node v_j in the connected graph G', in other words, diameter$(G') = d(p'_{v_i,v_j})$. We also let p''_{v_i,v_j} be a shortest path between node v_i and v_j in graph G''. Since p''_{v_i,v_j} may not be the longest shortest path in G'', we have $d(p''_{v_i,v_j}) \leq$ diameter(G''). Because we assume that G' is not an MDS, we have the following inequality:

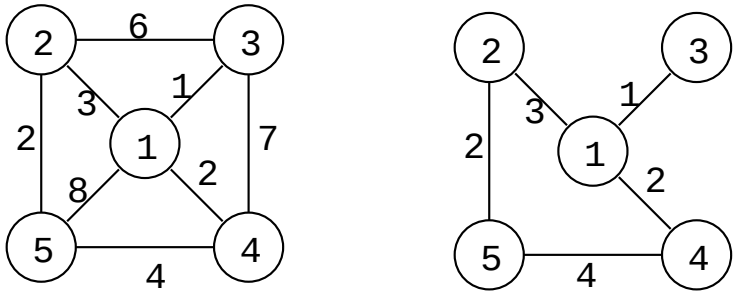
$$d(p''_{v_i,v_j}) \leq \text{diameter}(G'') < \text{diameter}(G') = d(p'_{v_i,v_j}).$$

However, based on the proposed algorithm, the path p'_{v_i,v_j} is guaranteed to be the shortest path between node v_i and v_j in graph G (e.g., line 4 and line 5 ensure this condition). Consequently, it is not possible to find another path p''_{v_i,v_j} which has a lower value than p'_{v_i,v_j} and therefore a contradiction occurs. ∎

One important point about the proposed algorithm is that the total edge cost of the derived communication subgraph G' may be high. As stated above, to minimize the total edge cost in G' is an NP-complete problem. However, we can try to reduce the total edge cost of the connected

graph G' by the following heuristic. Try to delete some edges from G' such that, for each edge deletion, the diameter of G' remains unchanged and the resulting graph is still connected. If the removal of edge e will cause the diameter of G' to increase, we do not remove this edge. We can apply this heuristic a finite number of times, e.g., try to remove a finite number of edges from G' so as to have a communication subgraph that has a minimum diameter and, at the same time, a reasonable total edge cost.

Figure 4 shows an example of finding a minimum diameter subgraph. Figure 4(a) is a graph G that represents a model of a network in which all nodes are participating nodes. Figure 4(b) is an MDS G' corresponding to the connected graph G in (a). It is worthwhile to mention that the MDS may contain cycles, and usually the total cost of edges is larger than the total cost of edges in the corresponding minimum spanning tree.



(a) (b)

Figure 4: An MDS example: (a) the original graph G, (b) the corresponding MDS G'

Core-based Tree (CBT):

The core-based tree (CBT) was proposed in [1] and is intended to provide a general framework for subgraph construction in a large scale network where clients are located in different parts of the Internet. The key features of CBT are: 1) there is a designated node called the *core*; 2) the path between any node to the core node must be a shortest path; 3) the routing policy for each node (or router) is efficient and easy to implement (e.g., based on the current Internet routing protocol); 4) the core-based tree is constructed in an incremental manner, such that any change to the client membership imposes only small changes on the communication subgraph. However, it is important to point out that the CBT does not guarantee a minimum diameter spanning tree. It only guarantees that the path between any node and the core is the shortest.

A CBT is constructed in a distributed, incremental manner. The construction procedure of CBT

is as follows: initially, a core is chosen from the set of participating clients, either manually or via a bootstrap mechanism as discussed in [1]. The core node is the only node of the communication subgraph at this stage. Then, each participating client will send a `JOIN_REQUEST` message to the core (the IP address of the core node is advertised and is well-known). This join message is sent through a shortest path from the participating client's node to the destination core, which can accomplished via the existing Internet routing protocol. Along this shortest path, the join message may reach a node which is either part of the current communication subgraph (e.g., a node which has already joined the CBT) or a node that is not part of the tree. If the join message reaches a node which is a part of the multicast tree, the forwarding process will stop and the incoming link will be added to the *forwarding cache* of the visit node. In this case, the visit node will send an acknowledgment message (`JOIN_ACK`) back to the participating client's node via the incoming interface. On the other hand, if the join message reaches a node which is *not* part of the multicast tree, the visit node will redirect the message to the next hop along the shortest path toward the core node and will cache the incoming interface and the incoming node in temporary storage. After that, the visit node waits for an acknowledgment message. Once an acknowledgment is received, the node adds the incoming interface to the forwarding cache and redirects the acknowledgment to all nodes listed in temporary storage. Also, it sets the node which sent the acknowledgment to be its parent node. Using this scheme, each participating client can reach the core node using the shortest path.

Once the CBT is formed, multicast service can begin. When a client sends a multicast message, it first reads the contents its *forwarding cache*, which is a list of its neighbor nodes in the CBT. Therefore, to multicast a message, a client can simply send the message to all nodes listed in its forwarding cache. When a node receives a message, it forwards the message to all the outgoing interfaces listed in the forwarding cache, excluding the incoming interface from which the message came. In this way, the message can reach every leaf node of the entire multicast tree.

The algorithm of CBT construction for our DVE application is given below. Initially, the CBT consists of one core node only, and we assume that this core is one of the clients $v_c \in C(t)$. The initial CBT is $G' = (V', E')$, where $V' = \{v_c\}$ and $E' = \emptyset$. Temporary storage is associated with each node v, and we denote it by tmp_v. The temporary storage of each node is empty at the beginning of the CBT construction procedure, i.e. $tmp_v = \emptyset, \forall v \in V'$. A client v who wants to join a DVE can send a `JOIN_REQUEST` message to the adjacent node v' which is along the

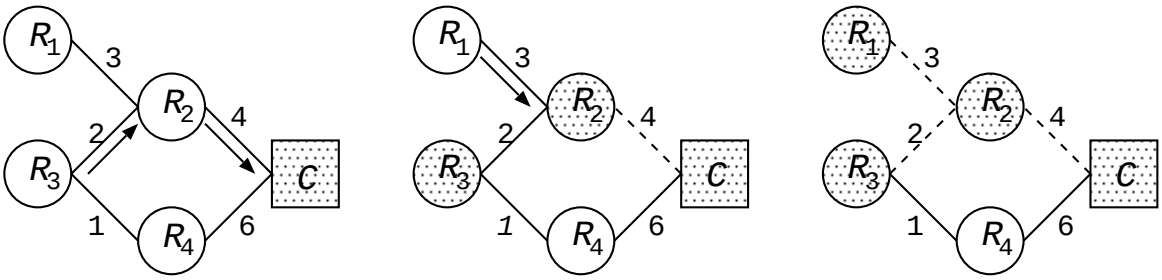


Figure 5: An example of CBT construction

shortest path from v to the core node v_c. When node v' receives a `JOIN_REQUEST` from node v, the following procedure will be executed:

---

if $v' \in V'$ **then**
    add v to the forwarding cache;
    reply with a `JOIN_ACK` message to v;
else
    $tmp_{v'} \leftarrow tmp_{v'} \cup \{v\}$;
    send a `JOIN_REQUEST` message to the first node which is along the shortest path from v' to the core v_c;

When node v receives a `JOIN_ACK` message from node v', the following procedure will be executed:

$V' \leftarrow V' \cup \{v\}$;
$E' \leftarrow E' \cup \{e\}$, where e is the edge connecting v and v';
add v' to the forwarding cache in v;
set v' to be the parent of v;
reply with a `JOIN_ACK` to all the neighboring nodes $u \in tmp_v$;
clear the temporary storage tmp_v;

An example of the CBT construction is depicted in Figure 5. Figure 5(a) shows a network topology of one core node C and four additional nodes R_1, R_2, R_3 and R_4. The number associated with each edge is the cost (i.e., propagation delay) of the edge. The shaded nodes and the edges with dotted lines are part of the multicast tree. In this example, node R_3 wants to join the CBT, and afterward node R_1 wants to join the CBT. Initially, core node C is the only node of the CBT. First,

node R_3 sends a `JOIN_REQUEST` message toward the core, as shown by the arrows in Figure 5(a). Intermediate node R_2 receives the request message and processes it. Since R_2 is not part of the multicast tree, it stores R_3 on its forwarding cache, and then R_2 sends a request message toward C. Core node C replies to R_2 with an acknowledgment and in turn R_2 sends an acknowledgment to R_3. Because of the acknowledgment, C is added to the forwarding cache of R_2 and C is marked as the parent of R_2. Similarly, R_2 is added to the forwarding cache of R_3 and R_2 is marked as the parent of R_3. On completion of the above process, the forwarding caches of R_2, R_3 and C are: $\{R_3, C\}$, $\{R_2\}$ and $\{R_2\}$ respectively. Also, R_2 and R_3 are now part of the multicast tree as depicted in Figure 5 (b). Now, consider the case for which node R_1 wants to join the multicast tree. It sends a join request message to R_2. Since R_2 is already part of the multicast tree, R_2 replies to R_1 directly and adds R_1 in its forwarding cache. Node R_1 adds R_2 into its forwarding cache after receiving the acknowledgment from R_2, and marks R_2 as its parent.

Spanning Tree (ST):

Consider the following problem: let $G = (V, E)$ be a graph where each edge $e \in E$ is associated with an edge cost $d(e)$. There is a subset of vertices $C(t)$, which is the set of DVE clients. Find a subtree G' of G that includes all the vertices in $C(t)$ such that the sum of all edge costs in the G' is less than or equal to L.

The motivation for finding the above subgraph is to guarantee that the total edge cost is less than or equal to L, in order to provide an upper bound on the network bandwidth consumption. This problem is the Steiner tree problem, which is known to be NP-complete[4]. Therefore, we propose to find an ST communication subgraph based on the following procedure:

- First, find a minimum spanning tree of the given graph G by some well known algorithm (e.g., Kruskal's algorithm or Prim's algorithm), the resulting tree being denoted by $G' = (V, E')$.
- For every node $v \in V$ with $v \notin C(t)$, check whether removing v and its associated edges will partition G' such that clients in $C(t)$ are in different partitions. If this does not occur, then remove v and its associated edges and consider the subgraph that contains $C(t)$. Otherwise, the node v should not be removed.
- The resulting subgraph is the communication subgraph G'.

Note that the subgraph G' generated by the above algorithm is also a tree. However, it does not guarantee the total edge cost to be minimum, since only a subset of nodes (e.g., the participating clients) are included in the subgraph.

In summary, the advantage of the MDS subgraph is that it can guarantee a low delay bound between any client in a DVE system. The CBT subgraph can be deployed in a wide area network such as the Internet, because it uses the existing Internet routing protocol. The ST subgraph can guarantee a low bandwidth consumption when sending synchronization messages between clients. The disadvantage of the MDS subgraph is that whenever a client joins or leaves the virtual world session, the DVE system may incur the overhead of constructing another MDS subgraph (more generally, the MDS construction procedure may be invoked after some fixed number of clients join or leave the system). The CBT, on the other hand, may not have a good transmission delay bound between two clients. Lastly, the ST subgraph may have a high client-to-client delay, and it will have some computational overhead to accommodate clients with dynamic membership characteristics.

4 Optimal Synchronization Interval

In this section we describe synchronization issues and present the derivation of the optimal synchronization interval. Without loss of generality, this is derived for one object, but the concept can be easily extended to multiple objects in a virtual environment. Consider a particular *dynamic* object o_i in a virtual environment, and let $\mathcal{S}_{o_i}$ be the set representing all possible states of object o_i. This dynamic object is associated with a master process P_{o_i} in a DVE system. For example, P_{o_i} may be a process running on one of the DVE servers or on one of the client's machines. To instantiate or enable the animation of object o_i, the master process P_{o_i} needs to perform the following:

1. Send the dead-reckoning information about object o_i to all participating clients who want to view o_i.

2. Send the "start-rendering" message to all participating clients.

3. Assume that we use the DRPS synchronization technique with τ the basic period of state update. For every period $n\tau$, the process P_{o_i} should send a "synchronization-message" to

all clients who are viewing o_i. The synchronization message has to carry the object ID of o_i, the current state of o_i, and if necessary, the new dead-reckoning information of object o_i (e.g., the new directional velocity so that object o_i can have a new trajectory).

For those clients who want to visualize the dynamics of object o_i, the rendering of o_i begins when they receive the "start-rendering" message. Since the trajectory of o_i can be computed based on the received dead-reckoning information in step 1, each client can *independently* begin the rendering process. That is, for every time step τ, each machine will update the state of the object o_i. When a client receives a "synchronization-message", if the state of the object o_i in the client's machine is equivalent to the state of o_i encoded in the synchronization message, then no adjustment is necessary and the client can continue to render the object o_i based on its current state. On the other hand, if the state of the object o_i in the client's machine is different from the state of o_i encoded in the synchronization message, then the current state of o_i is changed to the new state according to the information encoded in the synchronization message. Again, based on the dead-reckoning information embedded in the synchronization-message, each viewing client can compute the new trajectory of object o_i independently.

In Section 3, we presented several algorithms to construct a communication subgraph. Assume that we use one of the proposed methods and the resulting communication subgraph G'. We can use Equation (1) to find the maximum communication delay $d_{max}(G')$ between any two clients. Assume that there is a state update for every time step τ and τ is equal to $d_{max}(G')$. Therefore, if an object o_i has the sequence of state changes

$$s_{o_i}^1 \rightarrow s_{o_i}^2 \rightarrow s_{o_i}^3 \rightarrow s_{o_i}^4 \rightarrow \ldots$$

for every time step τ, then a client who is viewing object o_i should render the state sequence of object o_i such that:

$$c_{o_i}(d_{max}) = s_{o_i}^1; c_{o_i}(2d_{max}) = s_{o_i}^2; c_{o_i}(3d_{max}) = s_{o_i}^3; c_{o_i}(4d_{max}) = s_{o_i}^4; \ldots$$

where $c_{o_i}(t)$ denotes the state of object o_i at time t in the client's machine. Of course, if every client can render the state of object o_i exactly for every time step τ, then the phase difference (see Definition 1) of object o_i between any clients is equal to zero. However, the phase difference of object o_i between any two clients usually has a non-zero value. This can be explained by the fact that (1) the start-rendering operation may not arrive to different clients at the same time due to

the network delay, or (2) the workloads at different clients' machines are not the same, causing some machines to miss the rendering operation at certain time steps. In this paper, we consider the following *synchronization problem*:

Synchronization problem: Given the communication subgraph $G^{'}$ with maximum delay $d_{max}(G')$, how often (e.g., in terms of number of time steps) should the master process P_{o_i} of object o_i send the synchronization-message such that the condition:

$$E[|\Phi(C_A, C_B)|] \leq \Phi \tag{4}$$

can always be guaranteed ? In Equation (4), C_A and C_B are the two clients that are "furthest apart" in the communication subgraph $G^{'}$ (e.g., the communication delay between C_A and C_B is $d_{max}(G^{'})$), $E[|\Phi(C_A, C_B)|]$ is the expected absolute phase difference between the two clients C_A and C_B, and Φ is a pre-defined system parameter of the maximum allowable phase difference between any clients in a DVE system. Note that if Equation (4) holds, then the absolute phase difference of object o_i between any two clients C_i and C_j in a DVE system will be less than Φ, since the communication delay between C_i and C_j is guaranteed to be less than or equal to $d_{max}(G^{'})$.

To answer the question of how often the master process P_{o_i} needs to send a synchronization-message, we use a homogeneous discrete time Markov chain (DTMC) to represent the phase difference of o_i between clients C_A and C_B. To completely describe the DTMC, we need to specify the state space and the one-step transition probability matrix of the DTMC. In our model, a state of this DTMC specifies the phase difference of object o_i between clients C_A and C_B. For example, if object o_i in C_A is in state a and the object o_i in C_B is in state b, then the state of the DTMC (i.e., the phase difference between client C_A and C_B) is equal to $(a - b)$. We also assume that all participating clients need to update the state of object o_i every Δt where $\Delta t \leq \tau$. Since the maximum propagation delay of the subgraph is $d_{max}(G')$, we assume this DTMC will make a state transition every time step of length $d_{max}(G')$. Within a time step, each participating client can make at most $L = \lceil \frac{d_{max}(G^{'})}{\Delta t} \rceil$ state updates.

Formally, let $\mathcal{M} = \{\Phi(C_A, C_B)(n) | n \geq 0\}$ be this DTMC where $\Phi(C_A, C_B)(n)$ represents the phase difference between C_A and C_B at time step n. The state space of $\mathcal{M}$ is $\{\cdots, -2, -1, 0, 1, 2, \cdots\}$. For example, if $\Phi(C_A, C_B)(n) = 5$, then client C_A is ahead of client C_B by 5 at time step n. Let

$\boldsymbol{P}$ be the one-step transition probability matrix of $\mathcal{M}$, where $\mathsf{p}_{i,j}$ (i.e., the (i,j) entry of $\boldsymbol{P}$) represents the transition probability from state i to state j in $\mathcal{M}$ or $\mathsf{p}_{i,j} = \text{Prob}[\Phi(C_A, C_B)(n+1) = j | \Phi(C_A, C_B)(n) = i]$. Because both computing nodes of C_A and C_B may update the state of the object o_i for every Δt, at the end of every state transition of this DTMC, each computing node can make at most $L = \lceil \frac{d_{max}(G')}{\Delta t} \rceil$ state updates where $d_{max}(G')$ is the maximum network delay of the communication subgraph G'.

The next issue is to derive expressions for the transition probabilities $\mathsf{p}_{i,j}$ in $\boldsymbol{P}$. Let p_A (p_B) be the probability that client C_A (C_B) misses a rendering operation of object o_i in one time step[1] of Δt. In general, p_A and p_B are functions of the workload in C_A and C_B. Also, let p_{+k} and p_{-k} be the probability that the state of this DTMC increases or decreases by $k \geq 0$ for each state transition. We first derive the expression for p_0. The intuitive idea is to sum up probabilities of all possible cases with L state updates which result in no change in $\Phi(C_A, C_B)$. For example, we may have l state updates that cause $\Phi(C_A, C_B)$ to increase by l and, correspondingly, l other state updates that cause $\Phi(C_A, C_B)$ to decrease by l. The remaining $L - 2l$ state updates will not cause $\Phi(C_A, C_B)$ to change. Based on this observation, we have:

$$p_0 = \sum_{l=0}^{\lfloor \frac{L}{2} \rfloor} \left[\frac{L!}{l!l!(L-2l)!} (\alpha_F^l)(\alpha_B^l)(\alpha_S^{L-2l}) \right] \tag{5}$$

where $\alpha_F = (1 - p_A)p_B$ is the probability that $\Phi(C_A, C_b)$ increased by one (e.g, C_A updates the state change of o_i while C_B misses the state update of o_i), $\alpha_B = p_A(1 - p_B)$ is the probability that $\Phi(C_A, C_B)$ decreased by one and $\alpha_S = p_A p_B + (1-p_A)(1-p_B)$ is the probability that $\Phi(C_A, C_B)$ remains unchanged (e.g, C_A and C_B are both able or both unable to update the state of o_i).

Using a similar argument, the probabilities that the DTMC increases or decreases by $k > 0$ for each state transition are :

$$p_{+k} = \sum_{l=0}^{\lfloor \frac{L-k}{2} \rfloor} \left[\frac{L!}{(l+k)!l!(L-2l-k)!} (\alpha_F^{l+k})(\alpha_B^l)(\alpha_S^{L-2l-k}) \right] \quad \text{for } 0 < k \leq L \tag{6}$$

$$p_{-k} = \sum_{l=0}^{\lfloor \frac{L-k}{2} \rfloor} \left[\frac{L!}{(l+k)!l!(L-2l-k)!} (\alpha_B^{l+k})(\alpha_F^l)(\alpha_S^{L-2l-k}) \right] \quad \text{for } 0 < k \leq L. \tag{7}$$

[1] For example, if client C_k is viewing the object o_i at time t and if the client missed the rendering operation at time step t, then $s_{o_i}^k(t+1) = s_{o_i}^k(t)$.

The complete specification of the one-step transition probability matrix $\boldsymbol{P}$ is:

$$\mathsf{p}_{i,j} = \begin{cases} p_0 & \text{for } i = j \\ p_{+k} & \text{for } j = i + k, \quad \text{for } 0 < k \leq L \\ p_{-k} & \text{for } j = i - k, \quad \text{for } 0 < k \leq L \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

To illustrate, consider the example in Figure 6 which illustrates a DTMC for $L = 2$ state updates for each state transition in the Markov chain. Each state i can make a transition back to state i with probability p_0, or make transitions to state $i \pm 1$ and state $i \pm 2$ with probability $p_{\pm 1}$ and $p_{\pm 2}$ respectively.

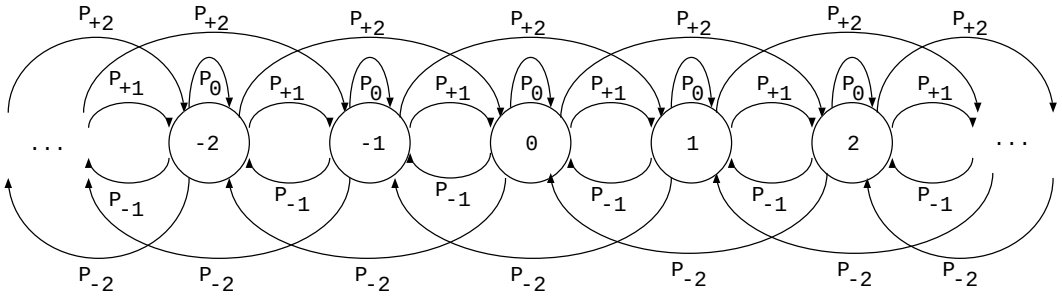


Figure 6: The Markov Chain with $L = 2$

To answer how we can maintain the condition of Equation (4) (e.g., how often a "synchronization-message" should be sent so that the expected value of the absolute phase difference between clients C_A and C_B is less than or equal to Φ), we use the theory of the fundamental matrix [2, 8] and analyze the underlying Markov chain $\mathcal{M}$. Let state s_i in $\mathcal{M}$ represent the value of the phase difference i. We aggregate (see [2]) all states $s_i, |i| > \Phi$, into one absorbing state (i.e., once the Markov chain reaches that state, it stays in that state forever), and this yields a new Markov chain $\mathcal{M}'$ with $2\Phi + 2$ states. The state space of $\mathcal{M}'$ is

$$\{s_{-\Phi}, s_{-\Phi+1}, \ldots, s_0, \ldots, s_{\Phi-1}, s_\Phi\} \cup \{s_a\}$$

where s_a is the absorbing state. The one-step transition probability matrix of $\mathcal{M}'$ is denoted by $\boldsymbol{P'}$ and has the form:

$$\boldsymbol{P}' = \left(\begin{array}{c|c} \boldsymbol{Q} & \boldsymbol{C} \\ \hline \mathbf{0} & 1 \end{array}\right)$$

where $\mathbf{Q}$ is a sub-stochastic matrix describing the transition probabilities between the $2\Phi + 1$ states correspond to phase difference values of $0, \pm 1, \pm 2, \ldots, \pm\Phi$, $\mathbf{C}$ is a column vector representing

the transition probabilities between the $2\Phi + 1$ transient states and the absorbing state, and $\mathbf{0}$ is a row vector of $2\Phi + 1$ zeros. Note that all entries in $\mathbf{Q}$ and $\mathbf{C}$ can be derived from $\boldsymbol{P}$, the one-step transition probability matrix of $\mathcal{M}$. Given $\mathbf{Q}$, the fundamental matrix $\boldsymbol{M}$ is:

$$\boldsymbol{M} \;=\; (\boldsymbol{I} - \boldsymbol{Q})^{-1} = \boldsymbol{I} + \boldsymbol{Q} + \boldsymbol{Q}^2 + \boldsymbol{Q}^3 + \cdots = \sum_{k=0}^{\infty} \boldsymbol{Q}^k. \tag{9}$$

The fundamental matrix $\boldsymbol{M}$ contains much useful information about the underlying DTMC. For example, we can use $\boldsymbol{M}$ to compute the average number of time steps (or the average number of transitions in the DTMC) so that the absolute value of the phase difference in $\mathcal{M}$ is greater than Φ.

Theorem 3 *Let $X_{i,j}$ ($-\Phi \leq i, j \leq \Phi$) be the random variable representing the number of time steps at which the DTMC $\mathcal{M}'$ visits state s_j before entering the absorbing state, given that the DTMC $\mathcal{M}'$ started from state s_i. We have*

$$E[X_{i,j}] = m_{i,j} \qquad \text{for } -\Phi \leq i, j \leq \Phi$$

where $m_{i,j}$ is the (i, j) element of the fundamental matrix $\boldsymbol{M}$.

Proof: Please refer to [2]. ∎

Remark: The above theorem implies that the entry $m_{i,j}$ of $\boldsymbol{M}$ represents the average number of transitions (or average number of time steps) for the DTMC $\mathcal{M}'$ to move to state s_j, given that the DTMC $\mathcal{M}'$ starts from state s_i, before entering the absorbing state s_a. Since states s_i and s_j represent the phase difference between C_A and C_B for object o_i, by calculating the value of $m_{i,j}$ we know the mean number of transitions (or average time) from state s_i to state s_j before the phase difference exceeds the maximum allowable phase difference of Φ.

Corollary 1 *Assume that the phase difference between client C_A and C_B is initially zero. Let t^* be the number of time steps such that the expected value of the absolute phase difference between C_A and C_B is greater than Φ. Then t^* is given by the expression:*

$$t^* = \sum_{k=-\Phi}^{\Phi} m_{0k} \tag{10}$$

Proof: We can show this by directly applying Theorem 3. Since the phase difference between clients C_A and C_B is initially zero, the DTMC starts from state s_0. Given the fundamental matrix $\boldsymbol{M}$, the entry m_{0j} is the average number of time steps for the DTMC to move to state s_j, given that the DTMC starts from state s_0, before entering the absorbing state s_a. Therefore, by summing m_{0k} for all possible states s_j different from s_a, we obtain t^*. ∎

Corollary 2 *To ensure that the expected value of the absolute phase difference between client C_A and C_B is less than or equal to Φ for all time steps, the master process P_{o_i} of object o_i needs to send the* "synchronization-message" *every $t^* = \sum_{k=-\Phi}^{\Phi} m_{0k}$ time steps.*

Proof: This can be directly observed by applying Corollary 1. ∎

Remark: Note that once the "synchronization-message" is received by all clients, the phase difference of object o_i between any clients is reset to zero. Therefore, in order to maintain the phase difference between any two clients to stay within Φ, the master process P_{o_i} needs to periodically broadcast the synchronization message with period $t^* = \sum_{k=-\Phi}^{\Phi} m_{0k}$.

5 Experiments

In this section, we carry out experiments to illustrate the maximum delay, the bandwidth consumption and the synchronization interval for different communication subgraphs which are generated by the MDS, CBT and the ST subgraph algorithms.

In our experimental study, we generate two graphs, namely, G_1 and G_2. Three parameters n, l and m are used for graph generation, which represent the number of nodes, the number of edges in the graph and the number of participating clients, respectively. For each edge e in the graph, there is an associated delay $d(e)$, which is generated uniformly between 1 and D_{max} (the maximum allowable delay in the graph). We use different values for these parameters to generate two graphs G_1 and G_2, namely,

	n (total # of vertices)	m (total # of clients)	l (total # of edges)	D_{max} (max. delay)
G_1	100	50	200	10
G_2	200	100	400	10

Given the graph G_1 or G_2, we construct the corresponding communication subgraph G' using the MDS, the CBT and the ST algorithms. Figure 7 illustrates the corresponding subgraph diameter as well as the total edge cost in the corresponding communication subgraphs. It is interesting to

	diameter	total edge cost
MDS	30	415
ST	73	354
CBT	58	261

communication subgraph for G_1

	diameter	total edge cost
MDS	37	751
ST	70	612
CBT	67	561

communication subgraph for G_2

Figure 7: Communication subgraph diameter and total edge cost for (a) G_1 and (b) G_2

point out that MDS guarantees a minimum diameter communication subgraph, while ST cannot provide a subgraph with a minimum edge cost (as compared to CBT).

Given the communication subgraph $G^{'}$, the next experiment illustrates the tradeoff between bandwidth consumption and the synchronization interval. For this experiment, each client broadcasts a synchronization message at every fixed interval to other clients in a DVE system. Each client uses these synchronization control messages to maintain a consistent view of the virtual world. When the synchronization message passes through a communication channel (e.g., represented by an edge in the communication subgraph), it spends a fixed time equal to the delay of the channel. The network bandwidth consumption is computed based on Equation (3). Figure 8 and Figure 9 illustrate the bandwidth consumption for the graph G_1 and G_2, respectively. For each network topology, we construct various communication subgraphs $G_1^{'}$ and $G_2^{'}$ using the MDS, CBT and ST algorithms. We can observe that the shorter the synchronization interval, the higher the value of the network bandwidth consumption. Another important observation is that the bandwidth consumption of the MDS subgraph is less than the bandwidth consumption of the CBT and ST subgraphs. The reason is that the MDS subgraph is constructed based on an all pairs shortest path algorithm. Therefore, the communication paths are much shorter and the time for the synchronization control message to reside in the network is much smaller than the other two communication subgraphs. In general, the CBT and the ST subgraphs do not guarantee the shortest path between all pairs of senders and receivers. Therefore synchronization control messages stay in the network for a longer period of time.

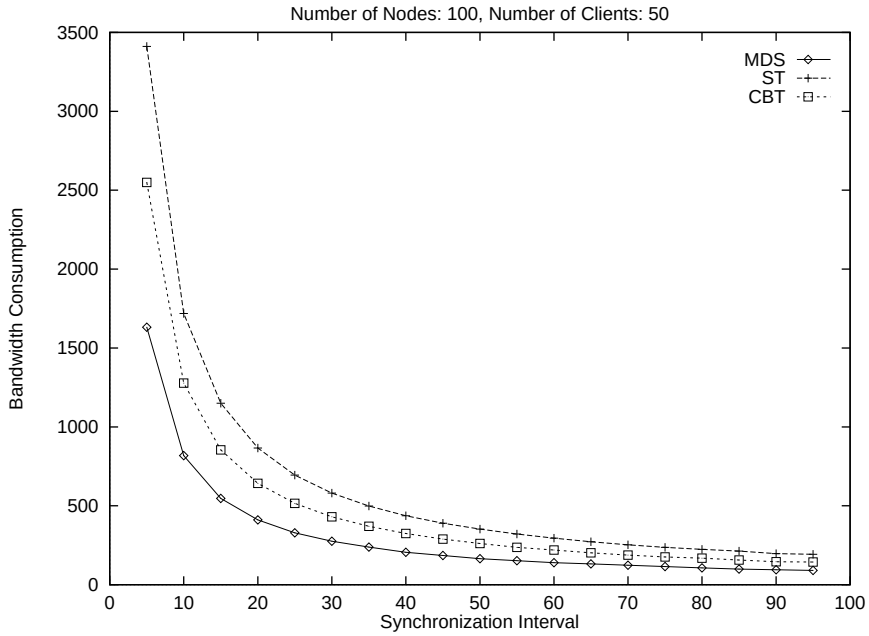


Figure 8: Bandwidth Consumption vs Synchronization Interval for G_1

The next experiment illustrates the effectiveness of finding the optimal synchronization interval under various system parameters. We consider the graph topology G_1, and we use a homogeneous system such that every machine will have the same probability p of missing a rendering computation. Figure 10 illustrates the optimal synchronization interval (the derivation is based on Corollary 2) versus Φ, the maximum allowable phase difference requirement. In general, the lower the value of p, the more the system can afford a longer synchronization interval to maintain view consistency. This implies that the lower the value of p, the lower the bandwidth consumption for sending the synchronization control message. Another important point is that given a particular value of the optimal synchronization interval, the MDS has less bandwidth consumption than other subgraphs generated via the CBT and the ST methods. For example, if we consider Figure 10 with $p = 0.2$ and the maximum allowable phase difference Φ is equal to 3, then the optimal synchronization interval is equal to 28. Using this synchronization interval, we can calculate the network bandwidth consumption for various communication subgraphs. If we consider the network G_1 in Figure 8 as an example, the bandwidth consumption for various communication subgraphs under the optimal synchronization interval of 28 is given in the following table:

	MDS	ST	CBT
BW consumption	295.826	624.247	461.127

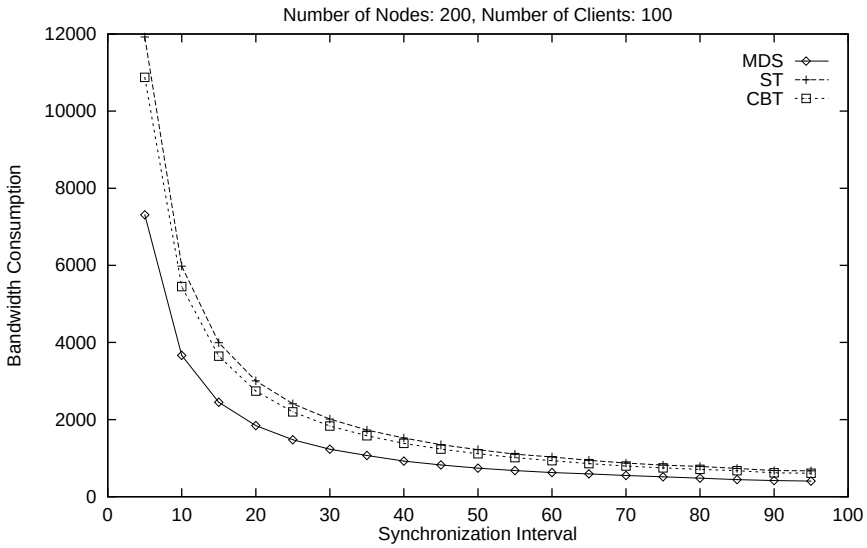


Figure 9: Bandwidth Consumption vs Synchronization Interval for G_2

Therefore, given this synchronization interval, we see that the communication bandwidth consumption for the MDS subgraph has a factor of 2.11 and 1.35 reduction over the bandwidth consumption for subgraphs generated by the ST and the CBT methods, respectively.

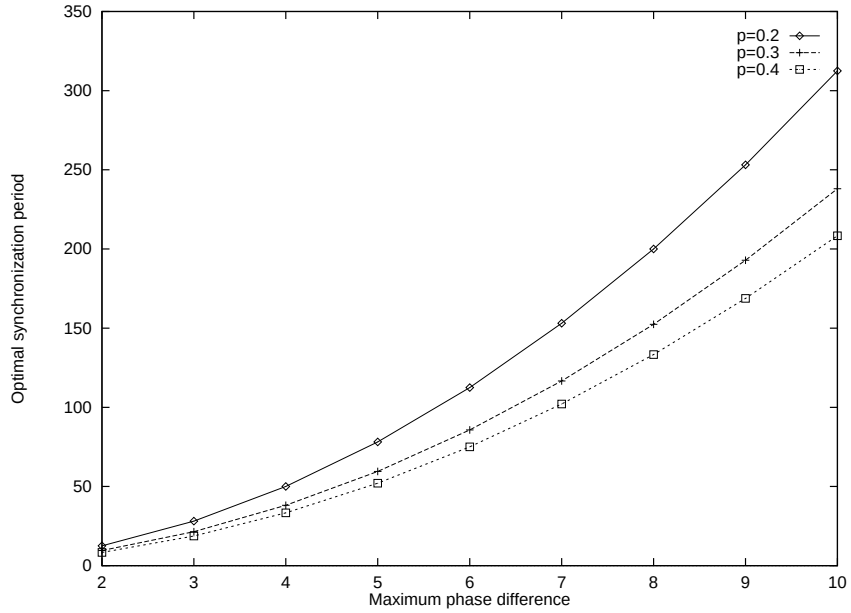


Figure 10: Optimal synchronization interval t^* vs. maximum allowable phase difference Φ

Figure 11 illustrates the relationship between p, the probability of missing a rendering operation (x-axis), Φ, the maximum allowable phase difference (y-axis) and the network bandwidth consumption (z-axis). In general, the system will have a high bandwidth consumption if the synchronization requirement is high (e.g., Φ is small). Maximum bandwidth consumption occurs

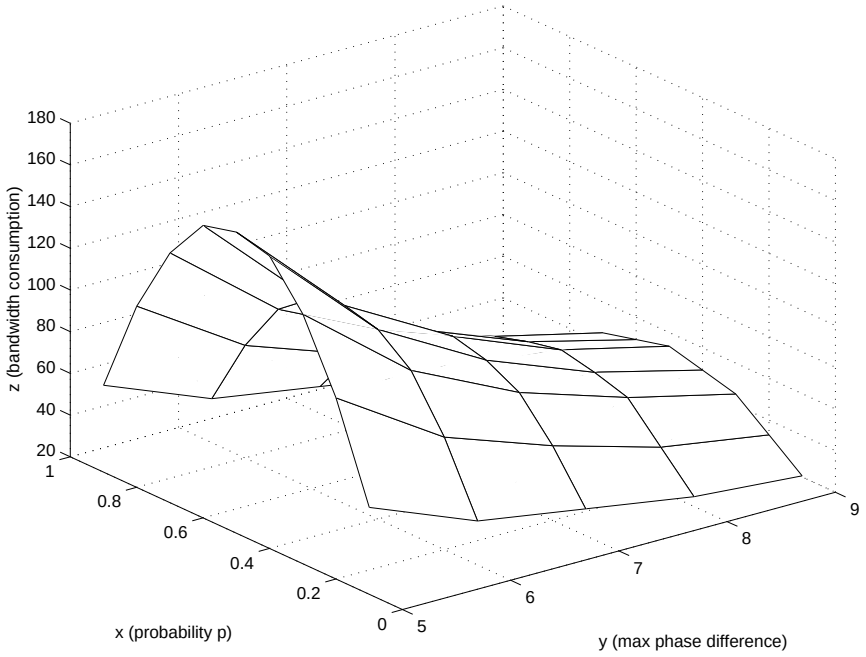


Figure 11: Relation between p (x-axis), Φ (y-axis) and bandwidth consumption (z-axis)

when $p = 0.5$. In this case, the expected value of the absolute phase difference between the two viewing clients is very high, and the system needs to broadcast the synchronization message more often so as to maintain a given level of view consistency.

In the next experiment, we investigate the relationship between the optimal synchronization interval t^* under different values of synchronization threshold Φ and different values of the probability of missing a rendering operation for clients C_A and C_B. We set $p_A = 0.1$ and we vary p_B from 0.2 to 0.8. We also vary the synchronization threshold Φ from 5 to 10. Figure 12 illustrates the result of the synchronization interval under different settings. In general, we observe that the larger the difference between p_A and p_B, the smaller the synchronizing interval required to support a given value of threshold Φ. This is because the skewness of p_A and p_B specifies the difference in the behavior of these two computing nodes, which reflects the difference in the rate of state updates. We also observe that the smaller the value of the threshold Φ, the smaller the synchronizing interval required so as to maintain the view consistency of a DVE system.

Lastly, we investigate the effect of the magnitude of p_A and p_B on the optimal synchronization interval and the values of the threshold Φ. We vary p_A from 0.1 to 0.8, and for a given value of p_A we set $p_B = 0.1 + p_A$. We also vary the threshold value Φ from 5 to 10. Figure 13 illustrates the optimal synchronization interval t^* under different system configurations. Notice that when p_A

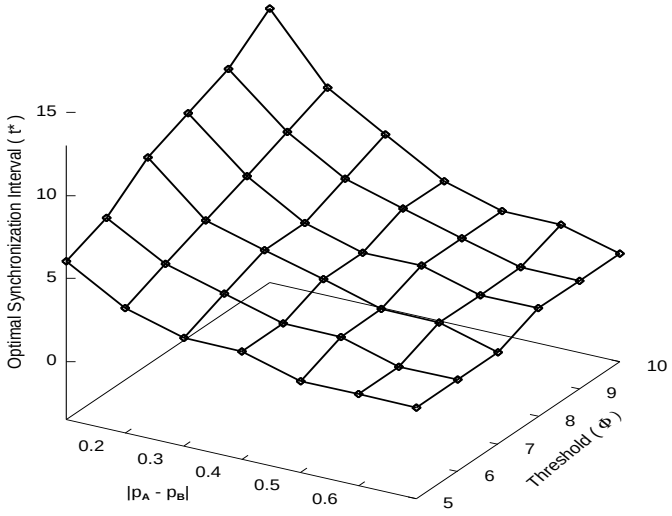


Figure 12: Optimal synchronization interval vs. synchronization threshold Φ with $|p_A - p_B| \leq 0.7$.

and p_B approach 0.5, the smallest synchronizing interval is required.

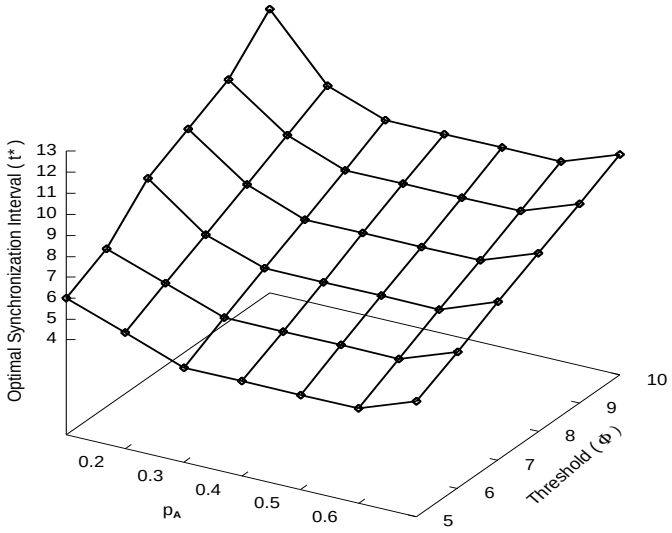


Figure 13: Optimal synchronization interval vs. synchronization threshold Φ with $p_B = p_A + 0.1$

6 Conclusion

We consider synchronization issues for a large scale DVE system. Our work is divided into two parts. In the first part, we propose several communication subgraph construction algorithms for connecting all participating clients in a DVE environment. Our experiments show that the MDS algorithm can guarantee the communication subgraph to have a minimum diameter, thereby minimizing the maximum delay between any two clients. From our experiments, the bandwidth consumption of the communication subgraph constructed by the MDS is the lowest, when compared with the communication subgraph generated by the CBT and ST algorithms. Given a communication subgraph G', we can easily find the corresponding maximum network delay $d_{max}(G')$. We then derive the optimal synchronization interval such that the expected phase difference between any two clients in a DVE system is less than or equal to Φ. The approach for deriving the optimal synchronization interval t^* is to represent the phase difference of the two furthest clients with a DTMC and use the theory of the fundamental matrix. Whenever there is a large deviation between p_A and p_B, a DVE system needs to send the synchronization message more often to guarantee that the phase difference between any two clients is less than or equal to Φ. Future work includes studying how to track the probability of missing a rendering operation under a real-time environment. One possibility is to use a statistical approach wherein a process measures the number of missing rendering operations for a given period. This estimate is then used as the probability p in our proposed mathematical model to determine the synchronization interval.

Acknowledgments

We would like to acknowledge various anonymous referees for their insightful and constructive comments.

APPENDIX I: notation

$G(V, E)$	=	A connected graph G which represents the underlying network. V is the set representing nodes in G and E is the set representing edges in G.
v_i	=	A node in V which can represent a router or a participating client.
$\mathcal{V}$	=	The set of computers (or nodes in V) that participates in a DVE session. We have $\mathcal{V} \subseteq V$.
$e_{i,j}$	=	A communication link between node v_i and v_j.
$d(e_{i,j})$	=	Upper bound of the transmission delay for the communication link $e_{i,j}$.
$C(t)$	=	A set of participating clients at time t.
$G'(t)$	=	A connected communication subgraph of the underlying graph $G = (V, E)$. In general, we have $G'(t) = (C(t), E')$ where $C(t) \subseteq \mathcal{V} \subseteq V$ and $E' \subseteq E$.
$p_{i,j}$	=	A loop-free shortest path between node v_i and v_j.
$d(p_{i,j})$	=	Transmission delay for the path $p_{i,j}$.
$d_{max}(G)$	=	The maximum delay between all pairs shortest paths in the connected graph G.
o_i	=	An object in the DVE system.
$\mathcal{S}_{o_i}$	=	A set representing all possible states of object o_i in a DVE system.
$s_{o_i}(t)$	=	The state of object o_i at time t.
$c^j_{o_i}(t)$	=	The state of object o_i at time t that client j is rendering.
Seq	=	Sequence of state changes for object o_i.
Φ	=	Maximal allowable phase difference for any object between any clients in a DVE system.
P_{o_i}	=	A process that is responsible for sending the start-rendering and synchronization-message for object o_i in a DVE system.
τ	=	Basic synchronization period for the DRS and DRPS synchronization techniques.
$N_{G'}(t)$	=	Total number of synchronization message in transit at time t for the communication subgraph $G'(t)$.
$\mathcal{M}$	=	A DTMC which represents the phase difference of two furthest participating clients in a DVE system.
$\Phi(C_A, C_B)$	=	The random variable denoting the phase difference between the two furthest clients C_A and C_B.
Δt	=	The length of the time step wherein each participating node will update the state of object o_i in its local display. In general, we have $\Delta t \leq \tau$.
L	=	Maximum number of state updates for a computing node for each state transition in the DTMC.
s_i	=	The state in the DTMC $\mathcal{M}$ which represents the phase difference of C_A and C_B is equal to i.
s_a	=	An absorbing state in the DTMC $\mathcal{M}'$.
M	=	The fundamental matrix according to Equation (9).
$X_{i,j}$	=	A random variable representing the number of time steps that the DTMC $\mathcal{M}'$ visits state s_j before entering the absorbing state s_a, given that the DTMC started from state s_i.
t^*	=	The optimal synchronization interval of maintaining view consistency.

References

[1] A. J. Ballardie, P.F. Francis, and J. Crowcroft, “Core based trees”, In *Proceedings of ACM SIGCOMM*, pages 85–95, San Francisco, 1993.

[2] U. N. Bhat, *Elements of Applied Stochastic Processes*, John Wiley & Sons, New York, 1984.

[3] F. Fabbrocino, J.R. Santos, and R. Muntz. “An Implicitly Scalable, Fully Interactive Multimedia Storage Server”, *International Workshop on Distributed Interactive Simulation and Real Time Applications (DIS-RT’98)*, 1998.

[4] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W.H. Freeman, San Francisco, CA, 1979.

[5] John C.S. Lui, M.F. Chan, T.F. Chan, W.S. Cheung, W.W. Kwong. “Virtual Exploration and Information Retrieval System: Design and Implementation”. 3^{rd} *International Workshop on Multimedia Information Systems*, 1997.

[6] John C.S. Lui, M.F. Chan, K.Y. So, T.S. Tam, “Balancing Workload and Communication Cost for a Distributed Virtual Environment”, *Fourth International Workshop on Multimedia Information Systems*, September 24-26, 1998.

[7] R. Muntz, J.R. Santos, and S. Berson. “A Parallel Disk Storage System for Realtime Multimedia Application”, *Special Issue on Multimedia Computing Systems of the International Journal of Intelligent Systems*, 13(12), pp. 1137-1174, December, 1998.

[8] E. Parzen, *Stochastic Processes*, Holden-Day, San Francisco, California.

[9] J. Plesník, “The complexity of designing a network with minimum diameter”, Networks, 11, pp. 77-85, 1981.

[10] B. Roehle, “Channeling the data flood”, *IEEE Spectrum*, pp 32-38, March, 1997.

[11] H. Schulzrinne, GMD Fokus, S. Casner, R. Frederick, V. Jacobson “RTP: A Transport Protocol for Real-Time Applications”, *RFC 1889*, January 1996.

[12] S. K. Singhal, D. R. Cheriton. *Using a Position History-Based Protocol for Distributed Object Visualization*, Technical Report, Stanford University, Department of Computer Science, CS-TR-94-1505, Feb 1994.

[13] UCLA Virtual World Data Server Project (`http://mml.cs.ucla.edu/`).

[14] R. C. Waters, J.W. Barrus. “The Rise of Shared Virtual Environments”, *IEEE Spectrum*, pp 20-25, March, 1997.

[15] Lixia Zhang, Stephen E. Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala, “RSVP: A New Resource Reservation Protocol”, *IEEE Network*, 7(5):8-18, September 1993.

BIOGRAPHY

John C.S. Lui received his Ph.D. in Computer Science from UCLA. During the summer of 1990, he participated in a parallel database project in the IBM Thomas J. Watson Research Center. After his graduation, he joined a team at the IBM Almaden Research Laboratory/San Jose Laboratory and participated in research and development of a parallel I/O architecture and file system project. He later joined the Department of Computer Science and Engineering of the Chinese University of Hong Kong. His current research interests are in communication networks, distributed multimedia systems, OS design issues, parallel I/O & storage architectures and performance evaluation theory. His personal interests include general reading and movies.