Performance Analysis of Distributed Virtual Environments

by

Michael Kwok

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Computer Science

Waterloo, Ontario, Canada, 2006

Abstract

A distributed virtual environment (DVE) is a shared virtual environment where multiple users at their workstations interact with each other. Some of these systems may support a large number of users, e.g., massive multi-player online games, and these users may be geographically distributed. An important performance measure in a DVE system is the delay for an update of a user's state (e.g., his position in the virtual environment) to arrive at the workstations of those users who are affected by the update. This update delay often has a stringent requirement (e.g., less than 100 ms) in order to ensure interactivity among users.

In designing a DVE system, an important issue is how well the system scales as the number of users increases. In terms of scalability, a promising system architecture is a two-level hierarchical architecture. At the lower level, multiple service facilities (or basic systems) are deployed; each basic system interacts with its assigned users. At the higher level, the various basic systems ensure that their copies of the virtual environment are as consistent as possible. Although this architecture is believed to have good properties with respect to scalability, not much is known about its performance characteristics.

This thesis is concerned with the performance characteristics of the two-level hierarchical architecture. We first investigate the issue of scalability. We obtain analytic results on the workload experienced by the various basic systems as a function of the number of users. Our results provide valuable insights into the scalability of the architecture. We also propose a novel technique to achieve weak consistency among copies of the virtual environment at the various basic systems.

Simulation results on the consistency/scalability tradeoff are presented.

We next study the update delay in the two-level hierarchical architecture. The update delay has two main components, namely the delay at the basic system (or server delay) and the network delay. For the server delay, we use a network of queues model where each basic system may have one or more processors. We develop an approximation method to obtain results for the distribution of server delay. Comparisons with simulation show that our approximation method yields accurate results. We also measure the time to process an update on an existing online game server. Our approximate results are then used to characterize the 95th-percentile of the server delay, using the measurement data as input.

As to the network delay, we develop a general network model and obtain analytic results for the network delay distribution. Numerical examples are presented to show the conditions under which geographical distribution of basic systems will lead to an improvement in the network delay. We also develop an efficient heuristic algorithm that can be used to determine the best locations for the basic systems in a network.

Acknowledgements

First and foremost, I want to thank my supervisor, Professor Johnny Wong. This work would not have been made possible without his advice and encouragement. He has provided me with enormous freedom to pursue my own interests and at the same time guided my thinking on basic problems. I am also grateful for his generous financial support in these years.

I want to express my deep gratitude to Professor Jay Black, Professor Tamer Ozsü, Professor Gordon Agnew, and Professor Oliver Yang, for serving on my doctoral committee and giving me invaluable comments on my thesis work. Special thanks go to Professor Oliver Yang for coming all the way from the University of Ottawa to serve as my external examiner.

No thesis is written without the influences of others. I want to thank all the members of the Network and Distributed Systems Group. They have taught me what it means to belong to a community of great scholars. I owe much to Professor Jay Black, Professor Tim Brecht, Professor Martin Karsten, Professor S. Keshav, Professor David Taylor, and Professor Paul Ward for all the valuable and enlightening discussions on various research subjects. Especially, I want to thank Tim and Martin. During our collaboration in the groupcast project, they have shown me the indispensable qualities of young and enthusiastic researchers. I also want to thank them for letting me access their research machines to run my simulation program.

I want to thank Nicole Keshav, who has greatly helped me in the writing of this thesis. Over the past year, she has taught me so much about writing well and being a cheerful person.

I believe a key to getting through a doctoral program is to have good friends to have fun with and complain to. I have been fortunate to have great officemates and friends: David Evans, Goretti Fung, Kinson Ho, Ambles Kock, Ellen Liu, Paul Lo, James She, Jialin Song, Alex Sung, Joshua Tang, Jennifer To, William Wong, and Gary Yeung. Not only they are the people whom I can discuss my research with and goof off with, but they also stand by my side through good times and bad times. I offer my special thanks to Michael Cheung and Wendy Rush for their moral support and help to pass all my self-doubting.

I express my sincere thanks to my father, Robert Kwok, and my mother, Alice Kwok, for their endless support and love since my childhood. I also thank my sister Maise. I never would have made it through this doctoral program without her. I also want to thank my father-in-law, K.K.Lau, Pamela Ng, and Aunt Winnie for their moral support.

Last, but not least, I would like to dedicate this thesis to my wife, Tess, and my son, Raphael, for their love, patience, and understanding. They are always there for me in my joyful and sometimes heavy moments. Tess, thank you very much for your unselfish love and support. I love you.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

In recent years, distributed virtual environments (DVEs) have gained popularity among Internet users. This is substantiated by the increased interest in DVE systems such as multi-player online games (MOGs) [1–6] and computer-supported collaborative workplaces (CSCWs) [7]. Some of these systems may support a large number of simultaneous users, and these users may be geographically distributed. For example, *World of Warcraft* (WoW) is a MOG that currently has nearly 4 millions subscribers [1]. Each of its game servers can support up to a few thousands of players simultaneously. The game's virtual environment is generally composed of dungeons, cities, and open areas. WoW offers its players an appealing entertainment experience; it also generates significant revenue from its subscribers. Hence, over the past few years, MOGs have become a major trend in the entertainment industry.

In general, a DVE is a shared virtual environment where multiple users at their workstations interact with each other. Such an environment is often considered as interactive, immersive, multi-sensory, and synthetic [8]. Each user is represented by an entity, called an avatar.[1] Users may move around, perform various actions, or interact with each other within the virtual environment. Each user has a "vision domain" which is defined as the area in the virtual environment where interactions between this user and other users may take place [9]. Any changes in an avatar's state (e.g., position, orientation, and velocity) must be distributed to all users within this avatar's vision domain in real time. These users are referred to as "affected users" in our investigation.

An important performance measure of a DVE system is the "update delay," which is defined as the elapsed time from when a user submits a state update to when this state update arrives at the workstation of an affected user. The update delay should be small, e.g., less than 100 ms [10] because excessive delay would annoy users and consequently ruin the sense of realistic interaction.

A popular architecture for DVE systems is "client-server" [1, 2]. In this architecture, the virtual environment is maintained by a central server; all users interact with this server. When a user makes a move, a state update packet is sent to the central server. When this packet is processed, the corresponding avatar is moved to its new location. The server also forwards the new state information to all affected users. Each affected user, upon receiving this information, renders the changes on his workstation. The client-server architecture is popular because of its ease of implementation. The server can also perform operations such as update verification

[1]Each user is often associated with one avatar. In this thesis, "avatar" and "user" will be used interchangeably.

and user authentication.

For the client-server architecture, the update delay is affected by the response time at the server and the round trip network delay between the users and the server. As the user population of a DVE grows, the rate of state update packets generated by the users increases, leading to an increased load at the server. This may have a negative impact on the response time performance. For instance, in *World of Warcraft*, if the number of players logged on to a server exceeds some maximum, the server's performance degrades dramatically, and the game play experienced by the players becomes "laggy" (or unresponsive). The architecture of the DVE system should therefore be scalable. By scalable, we mean that a system's capacity can be improved in a straightforward manner to support more users without suffering noticeable degradation in response time performance [11].

In terms of scalability, a promising architecture proposed for DVE systems is a two-level hierarchical architecture [12–15]. At the lower level, multiple service facilities (referred to as basic systems) are deployed and users are assigned to these basic systems. Each basic system maintains its own copy of the virtual environment, and interacts with its assigned users using the client-server model. At the higher level, the various basic systems communicate among themselves to ensure that updates are sent to affected users and that their copies of the virtual environment are as consistent as possible. With the two-level architecture, basic systems can be added when there is a need to support more users. The basic systems may also be placed at locations close to their users in order to reduce network delay. However, if a user makes a move and an affected user is at a different basic system, then the state update packet must be sent to the remote basic system, which may yield a longer update delay.

Although the two-level hierarchical architecture is believed to have good properties in terms of scalability, not much is known about its performance characteristics, especially when one considers scenarios with large user populations. In this thesis, we first study the performance characteristics of the two-level architecture, focusing on the issue of scalability. At each basic system, the workload can be measured by the rate at which state update packets arrive at the basic system. When there are more basic systems, the number of users assigned to each basic system is smaller, leading to reduced arrival rate of state update packets from these assigned users. However, the rate of state update packets sent between basic systems may be increased. The total arrival rate seen by each basic system is therefore of interest. An analysis of this total rate as a function of the number of users would provide insights into the scalability of the two-level architecture. Our approach is to first develop models for the overall system, virtual environment and vision domain, and then derive analytic results for the total arrival rate at each basic system. Numerical results showing the impact on scalability of factors such as the number of users and vision domain size will be presented.

For the two-level hierarchical architecture, an important consideration is the consistency among copies of the virtual environment at the various basic systems. It has been suggested that global synchronization should be performed periodically to ensure consistency [14]. Such periodic synchronization, however, would consume processing capacity at the basic systems. Furthermore, one can only achieve weak consistency using this approach because inconsistency may occur between successive global synchronizations. We propose a new technique called “virtual vision domain” which would also result in weak consistency. Our technique incurs overhead as well. Simulation results showing the consistency/scalability tradeoff are presented. These results show that the virtual vision domain technique is a viable alternative to global synchronization for achieving weak consistency.

As mentioned earlier, the update delay is an important performance measure for a DVE system, and it is desirable for the update delay to be below some maximum value. The update delay has two main components, namely the delay at the basic system (or server delay) and the network delay. For the server delay, we develop a network of queues model where each basic system may have one or more processors. The distribution of server delay is of interest because it would provide insights into the percentiles of server delay, e.g., the 95th-percentile. We develop an approximation method to obtain analytic results for the server delay distribution. This analysis takes into consideration the two scenarios where an affected user may be at the local basic system or at some other basic system. The accuracy of our approximate analysis is evaluated by comparison with simulation results. In order to get realistic data about server delay, we measure the time to process an update for an existing online game server. Using the measurement data as input, we are able to gain an understanding of the magnitude of the server delay.

As to the network delay, we develop a general network model where users and basic systems are connected to some network end points. For the two-level hierarchical architecture, basic systems may be co-located or geographically distributed.

For each of these two scenarios, we develop a performance model and derive analytic results for the network delay distribution. Numerical examples are presented to show the conditions under which geographical distribution of basic systems would lead to an improvement in the network delay. For the geographically distributed scenario, the network delay is affected by the locations of the basic systems. We develop an efficient heuristic algorithm that can be used to determine the best locations for the basic systems in a network.

This thesis represents a significant step in understanding the performance characteristics of the two-level hierarchical architecture for DVE systems. Our contributions are summarized in the next section.

1.2 Contributions

Our thesis is the first known attempt to carry out an in-depth analysis of the performance and scalability of the two-level hierarchical architecture for DVE systems. The main contributions are as follows.

- We conduct a scalability analysis of the two-level architecture. In particular, our results provide new insights into the impact of various factors on the architecture's scalability and confirm that the architecture possesses good properties with respect to scalability.

- We perform an in-depth investigation of the issue of consistency in the two-level architecture, and discover that consistency can be restored as a result of user movement in the virtual environment, without employing any explicit synchronization measures.

- We propose a novel technique, called the virtual vision domain, for achieving weak consistency. Our simulation results show that this technique is effective in avoiding inconsistencies. Also, it may lead to a reduction in the time spent by a user in the inconsistent state.

- We develop an approximation method to obtain accurate analytic results for the server delay distribution. These results provide improved understanding of the delay in processing an update at the basic systems.

- We obtain new analytic results for the network delay distribution. We also present numerical examples to show the conditions under which geographical distribution of the basic systems will lead to an improvement in the network delay.

- We propose a heuristic algorithm that can be used to determine the best locations for the basic systems in a network. Such an algorithm is shown to yield good results; it is also computationally efficient.

1.3 Thesis Organization

The remainder of this thesis is organized as follows.

Chapter 2 reviews work in related research areas.

Chapter 3 presents a scalability analysis of the two-level hierarchical architecture. Analytic results for the total arrival rate of packets to each basic system are derived, and scalability of the architecture is investigated.

Chapter 4 examines the issue of consistency. Our proposed virtual vision domain technique is discussed, and its performance is evaluated by simulation.

Chapter 5 presents our approximation method to obtain analytic results for the server delay distribution. The accuracy of our approximation method is evaluated. We also present numerical examples to characterize the 95th-percentile of the server delay.

Chapter 6 derives the network delay distribution. An evaluation of the performance difference between the co-located and geographically distributed basic systems scenarios is presented. Our heuristic algorithm for finding the best locations for the basic systems is also described.

Finally, Chapter 7 contains a summary of our findings and suggestions for future research.

Chapter 2

Related Work

In this chapter, we review the research relevant to our study in performance and scalability of DVE systems.

2.1 Classic Architectures

There are two classic architectures proposed for DVE systems, namely "client-server" and "peer-to-peer."

The client-server architecture is shown in Figure 2.1. In this architecture, a central server maintains a copy of the virtual environment; all users interact with this server. When a user makes a move, a state update packet is sent to the central server. This packet joins a queue awaiting processing by the server. When the server processes a state update packet, the corresponding avatar is moved to its new location, and the server forwards the new state information to all affected users. Each affected user, upon receiving this information, renders the changes on his workstation. Basically, the server mediates interaction among users, and the user workstations are responsible for sending/receiving state update packets and graphical rendering. Examples of DVE systems using the client-server architecture are *Everquest* [2], *Lineage* [4], and *World of Warcraft* [1].

The client-server architecture is popular because of its ease of implementation. The central server maintains an authoritative copy of the virtual environment, which makes it easy to guarantee a consistent view of the environment among all users. The server can also perform operations such as update verification and user authentication. In some DVE systems like multi-player online games, the presence of a central server allows game operators to collect usage information conveniently, and to ensure the "well-being" of the virtual environment (e.g., the prevention of cheating). However, the central server could be a performance bottleneck. Scalability of this architecture is therefore limited by the server's capacity.

The peer-to-peer architecture is depicted in Figure 2.2. For this architecture, state update packets are sent directly between user workstations. Each workstation updates its own copy of the virtual environment using the state updates received. The peer-to-peer architecture originates with *SIMNET* (Simulation Network), which is an interactive network system for real-time battle simulation and military training [16]. A later version of *SIMNET*, called *DIS* (Distributed Interactive Simulation), was proposed as an ANSI/IEEE standard [17]. Both *SIMNET* and *DIS* are autonomous simulators in the form of software running on geographically distributed, networked host computers [18]. Subsequent DVE systems based on the same architecture include *DIVE* [19], *MiMaze* [10, 20], and *NPSNET* [21].

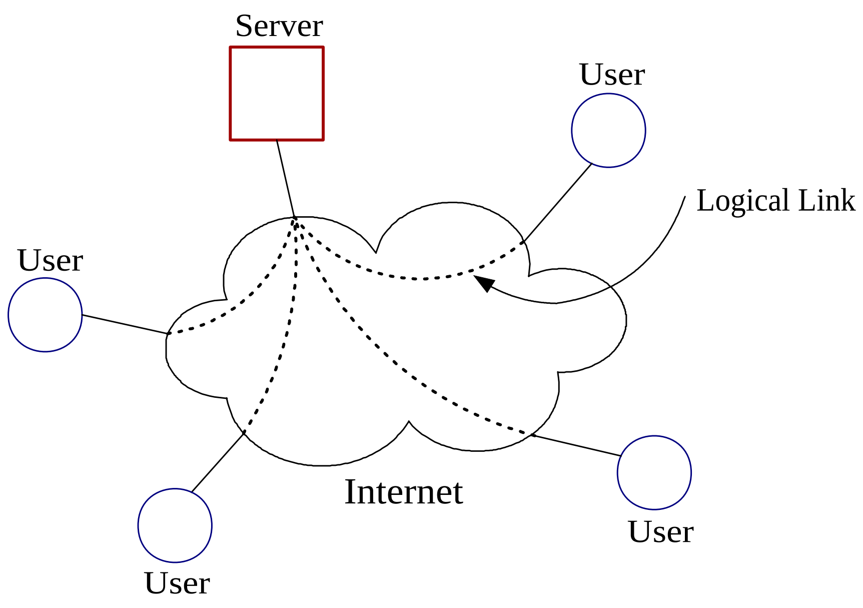


Figure 2.1: Client-server Architecture

The peer-to-peer architecture has several advantages over the client-server architecture [10]. First, the update delay is generally shorter because all state update packets are sent directly between user workstations without the involvement of a central server. Second, the peer-to-peer architecture does not suffer from a single point of failure or attack. Third, its "serverless" architecture avoids potential performance bottlenecks at the central server; such a bottleneck may limit scalability of the architecture. However, compared to client-server, the complexity and processing requirement at the user workstations may be significantly higher because each workstation is required to process state update packets (like performing update verification and user authentication on every state update packet), as well as render the changes caused by these updates.

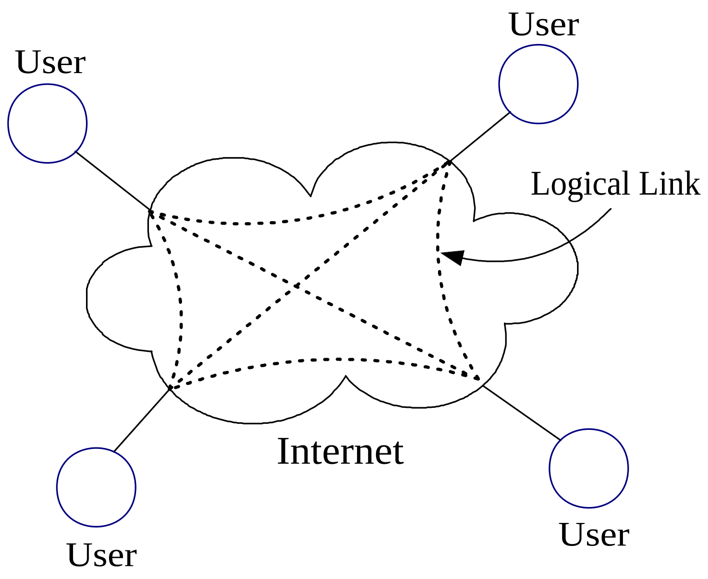


Figure 2.2: Peer-to-peer Architecture

2.2 Techniques to Improve Scalability

Several techniques have been proposed to improve scalability of the client-server and peer-to-peer architectures.

2.2.1 Dead Reckoning

Dead reckoning is a technique which reduces the number of state update packets that need to be transmitted by each user [22]. The principle is to model the state of an avatar, such as position, velocity, and orientation, by predictive extrapolation. This model is often referred to as the *dead-reckoning* model. For a given avatar, when the deviation between its exact state and the predicted state exceeds a pre-defined threshold, a state update packet is sent to those affected users. Each user uses the same dead-reckoning model to determine the state of other avatars between updates. With dead reckoning, state update packets may be sent at a lower rate, resulting in a reduction in both processing and bandwidth demands. Different extrapolation functions have been proposed to improve the accuracy of the prediction, while maintaining the computational complexity of the functions and the arrival rate of state update packets at a reasonable level [22–24].

2.2.2 Relevant Filtering

Another technique to improve scalability is relevant filtering, which is related to the notion of vision domain mentioned in Chapter 1. The basic idea is to reduce the packet traffic to individual workstations by sending only state updates that are relevant to them. This would result in a significant reduction in network traffic when compared to the case of broadcasting the updates to all other users [14]. In Figure 2.3, a "vision domain" is defined for every user to describe an area in the virtual environment within which interactions between this user and other users may take place. If user B is within user A's vision domain (see Figure 2.3), state updates made by user A will "affect" user B. With relevant filtering, each state update is delivered only to users who are affected by the update. It has been shown that this technique can reduce the number of state update packets transmitted and processed at the workstations significantly; therefore, it allows the DVE system to scale to a larger number of users [25].

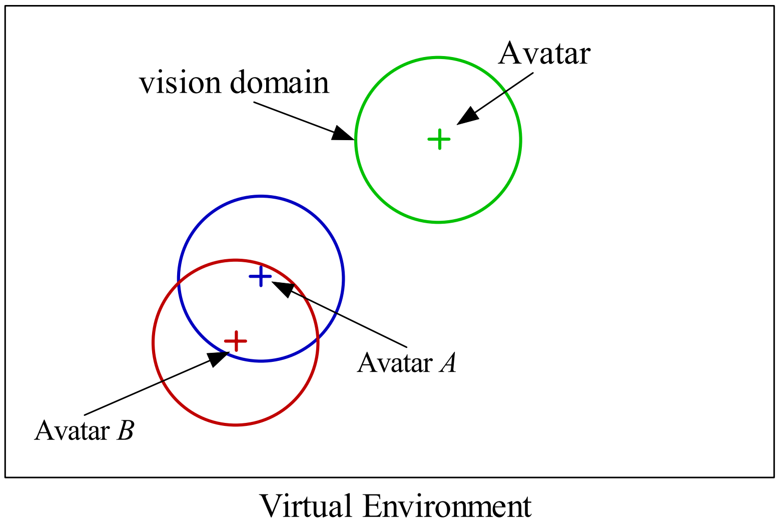


Figure 2.3: Vision Domains

2.2.3 Multicast

For the peer-to-peer architecture, relevant filtering has been implemented in DVE systems that make use of multicast [10, 26]. In these systems, users are mapped into different multicast groups, and state update packets are distributed within each group. A user subscribes to multicast groups of users whose vision domains contain this user; then this user will receive only those state updates that are relevant to him. Different techniques for assigning users to multicast groups have been proposed and studied [9, 27, 28]. In general, the use of multicast results in a smaller number of state update packets being transmitted, when compared to sending a separate unicast packet to each affected user. Note that multicast is applicable to both the peer-to-peer and client-server architectures. This technique, however, introduces complexities such as join/leave overhead and multicast group management.

2.2.4 Aggregation of Updates

For the client-server architecture, aggregation of updates at the central server is another technique that can lead to improved scalability. Instead of sending updates to an affected user immediately, the server may aggregate several updates (addressed to the same user) into a single message [29]. With one single message, the amount of traffic is reduced, and processing time for packet headers can be saved. Depending on the application protocol, it has been shown that the bandwidth requirement can be reduced by as much as 50% [29]. The drawback of this technique, however, is increased update delay.

2.2.5 Server Cluster

Also, for the client-server architecture, the processing capacity of the central server can be improved by the use of a server cluster [29]. A state update packet can be processed by any of the servers in the cluster. By deploying additional servers, a DVE system may support more users. However, as the number of servers increases, these servers may have to contend for resources, e.g., data access to the copy of the virtual environment. When this happens, an increase in the number of servers may not always lead to an improvement in the effective capacity.

2.2.6 Partitioning of the Virtual Environment

Another technique to improve scalability of a DVE system is to partition the virtual environment [30]. The main idea is to divide the virtual environment into two or more partitions and assign each partition to a different server. Then, each server is responsible for only those users located within its assigned partition. This technique not only reduces the potential resource contention among servers in a cluster, but also allows them to process packets in parallel. Different approaches used to partition the virtual environment can be found in [31–33].

Lui *et al.* proposed a partitioning algorithm for DVEs [31, 32]. Their goal is to obtain a partitioning where the workload is shared equally among the servers and the amount of server-to-server communication is kept to a minimum. As users move around, join, or leave a DVE, the workload among partitions may become imbalanced. Re-partitioning algorithms to restore the balance can be found in [31] and [33].

2.2.7 Hierarchical Architecture

A promising approach to supporting a large user population is a two-level hierarchical architecture, depicted in Figure 2.4. DVE systems using this architecture include *BrickNet* [34], *CyberWalk* [35], *Mirrored-Server* [12], *NetEffect* [13], *RING* [14], *Rokkatan* [15], and *Spline* [36].

In the two-level hierarchical architecture, multiple servers[1] are deployed, and

[1]Note that a server in the two-level hierarchical architecture could be replaced by a server cluster described in Section 2.2.5.

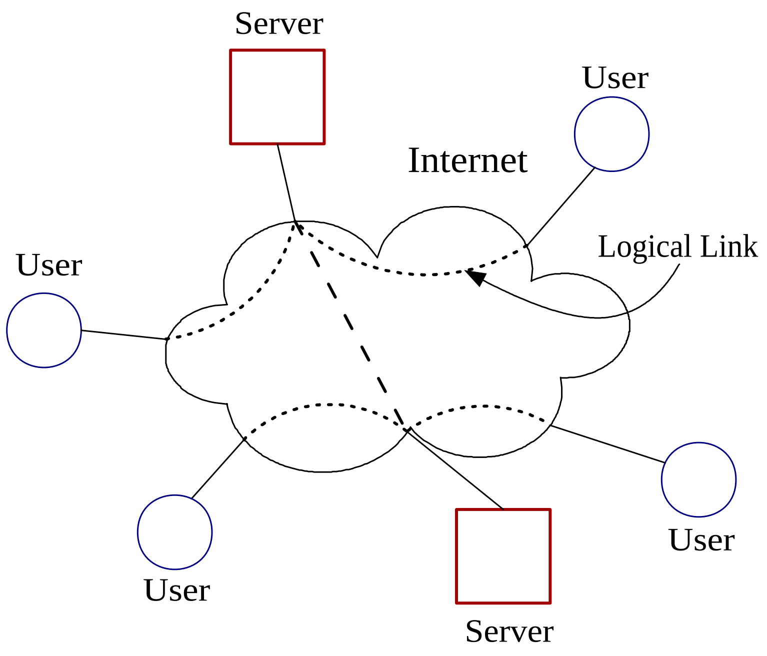


Figure 2.4: Two-level Hierarchical Architecture

they may be distributed geographically. Each server maintains its own copy of the virtual environment. Users are assigned to servers, and the assignment of users to servers can be based on load balancing and/or geographical considerations. Specifically, at the lower level, each user interacts with his assigned server as in the client-server architecture. When a user makes a move, a state update packet is sent to this assigned server. The server processes the packet, applies change(s) to its copy of the virtual environment, and distributes the update to all affected users. Note that the affected users may be at the local server or at some remote servers. For users at the local server, the update is sent directly to these users. Contrarily, for users at a remote server, the update is transmitted to the remote server via the higher level. The remote server then processes the packet, updates its copy of the virtual environment, and delivers the corresponding update to the affected users. Note that at the higher level, the servers communicate among themselves and operate like the peer-to-peer architecture.

With the two-level architecture, servers can be added when there is a need to support more users. The servers may also be placed at locations close to their users to reduce network delay. These are good properties with respect to performance and scalability. The two-level architecture also enjoys other advantages. For example, in case of server failure, users connected to the failed server may be temporarily redirected to the other servers. Another example is that techniques such as relevant filtering and aggregation of updates may be applied at both levels of the architecture. At the lower level (between users and their assigned servers), the details of these techniques have been described earlier in Sections 2.2.2 and 2.2.4.

At the higher level, a state update packet from a server can be sent only to those servers which have users affected by the update (relevant filtering). State update packets can also be aggregated into one single message before delivery to a specific remote server (aggregation of updates).

Nonetheless, the two-level hierarchical architecture introduces extra delay due to the exchange of state update packets among the servers.

2.3 Consistency

In addition to the extra delay, in the two-level hierarchical architecture, copies of the virtual environment at the various servers may become inconsistent.

It has been suggested that global synchronization should be performed periodically to ensure consistency [14]. Such a process, however, may consume processing capacity and may affect the architecture's scalability. Furthermore, one can only achieve weak consistency using this approach because inconsistency may occur between successive global synchronizations.

Note that the issue of consistency has also been investigated in distributed database. For example, in optimistic replication, data is replicated into multiple copies, called replicas, on separate computers, and any update to the data on a replica is distributed to the other replicas asynchronously. A detailed survey of optimistic replication can be found in [37].

2.4 Server Placement Problem

As servers in the hierarchical architecture are distributed across the network, a relevant issue is how to place these servers with a view to reducing network latency. The problem of placing the servers at strategic points is often referred to as the "server placement problem" (SPP).

SPP is closely related to a well-known class of problems, called "facility location problems." Two basic problems in this class are the p-center [38] and p-median problems [39]. The p-center problem finds the locations of p facilities (e.g., servers) so as to minimize the maximum distance from any demand node (e.g., user) to its closest facility. The p-median problem, on the other hand, finds the locations of p facilities, which will result in the minimum total distance between the demand nodes and the facility to which they are assigned. It is known that most of the facility location problems are computationally intractable [40], though exceptions occur in some special cases [41]. Thus, heuristics and approximation algorithms have been proposed. Surveys on this class of problems and solution techniques can be found in [40, 42]. More recent works regarding SPP, in the context of network applications, can be grouped into four areas: caching proxies [43, 44], web server replicas [45], reliable multicast [46], and Internet instrumentation [47].

With caching proxies, users' web requests may be satisfied by one of the proxies instead of the web server. This would tend to shorten the response time. In [43] and [44], mathematical models were developed and were used in the formulation of optimization problems to place these proxies in such a way that the average response time was minimized. The authors proved that the problems were *NP*-

Hard, and proposed algorithms for special types of network topologies such as line, ring, and tree.

Web server replicas, where web content is replicated to a number of servers called replicas, is another method to reduce the response time perceived by users. The problem is to place these servers in the network so that the average response time is minimized. In [45], several placement algorithms were evaluated by simulating their behavior on both synthetic and real network topologies. These algorithms included tree-based, greedy, random, and hot spot algorithms. The greedy algorithm gave the best performance and stability, yielding an average response time that was a factor of between 1.1 and 1.5 of the optimal.

SPP has also been investigated in the context of reliable multicast, where multicast servers (acting as intermediate servers) are introduced to alleviate the "acknowledgment implosion" problem. In case of packet loss, the multicast servers can provide users with fast retransmission/recovery. In [46], heuristic algorithms were presented for placing the multicast servers such that the bandwidth cost was minimized. The problem formulation was very similar to those for caching proxies and web server replicas.

Finally, Jamin *et al.* aimed at building a map of the Internet, which shows the distance between any two hosts on it [47]. The basic architecture consists of a network of instrumentation boxes, called *Tracers*, which measure the distances among themselves, and the distances among themselves and different regions of the Internet. One interesting problem is the placement of these Tracers such that the accuracy of the map is acceptable, while keeping the number of Tracers deployed to a minimum. This is similar to the p-center problem, which is known to be *NP-Hard*. Hence, the authors proposed an approximation solution.

2.5 Performance Analysis

Performance analysis of DVE systems is crucial to our understanding of system behavior. It can also help identify key parameters and their impact on system performance. Approaches that have been used to study the performance of DVE systems include analytic modeling, prototyping, simulation, and measurement.

2.5.1 Analytic Modeling

In [15], Muller *et al.* presented an online real-time strategy (RTS) game called *Rokkatan*. *Rokkatan* is based on a hierarchical architecture similar to that described in Section 2.2.7. The authors developed an analytic model, the "game scalability model" (GSM), to predict the number of players that their game system can support. This model divides the processing at the game server into five tasks: 1) receiving, validating, and processing state updates received from the server's local users; 2) receiving state updates from remote servers; 3) updating the database; 4) sending state updates to the server's local users; and 5) sending state updates to remote servers. For each of these tasks, GSM characterizes the average processing time required, the average amount of data received, and the average amount of data sent. Summing each measure over the five tasks and comparing the sums to pre-defined maximum values, GSM predicts the maximum number of players that can be supported. GSM accurately approximated the actual measurement made on *Rokkatan* with up to five servers.

Other work on analytic modeling of a DVE server can be found in [48]. In this work, the server performance is analyzed in terms of utilization and arrival rate of user requests at the server.

2.5.2 Prototyping and Simulation

In [49], a virtual battlefield was simulated using *SIMNET*. Users were allowed to control multiple objects in the virtual environment. The largest experiment, conducted in March 1990, showed that *SIMNET* was able to support up to 850 objects [50]. It was also shown that dead reckoning led to a reduction in the communication and processing load, and that predictive modeling resulted in a reduction of the perceived delay.

A number of simulators based on an *NPSNET* prototype were developed in order to investigate specific aspects of the system. One example is the use of multicast [27]. A 3D vehicle simulator was built using an IP-multicast version of *NPSNET* and was tested over the Internet with several North American sites. Input to the system was data obtained from a real-world military scenario. It was found that multicast reduced the bandwidth requirement significantly, when compared to the original broadcast approach. The reduction was on average over 70%. Another example is the exploration of a new technique for relevant filtering [28]. This technique is based on subdividing a virtual environment into octants. Octants may be coalesced on demand, depending on the density of avatars within them. It was found that this technique could eliminate up to 90% of network traffic.

Funkhouser implemented an experimental system of *RING* and tested it for a virtual environment with 800 “rooms” connected by “hallways” [14]. In his experiment, simulated avatars moved around randomly in the virtual environment. As the number of servers increased, the amount of server-to-user traffic decreased, but that of server-to-server traffic increased. The total amount of communication per server was in fact reduced. He also demonstrated that the use of relevant filtering could reduce the number of packets by up to 97.5%.

MiMaze is a multi-player online game which employs multicast and peer-to-peer architecture [10, 20]. Its virtual environment is a 3D maze. *MiMaze* was implemented and evaluated on the Mbone [51] with 25 players at different locations in France. It was observed that although the network delay was on average less than 100 ms, the delay distribution was widely dispersed (more specifically, the mean and the standard deviation were 55.47 ms and 50.44 ms, respectively). To reduce the standard deviation, Diot *et al.* proposed a mechanism called “bucket synchronization” [10]. The idea is to divide time into fixed length periods. State update packets received within a particular period are not processed until the end of the period. This mechanism helps reduce the standard deviation of the delay, and also allows the updates to be ordered properly (e.g., according to the time of occurrence) before they are processed.

2.5.3 Measurement

More recently, work has been done in measuring and characterizing the performance of multi-player online game (MOG) systems (one of the many DVE applications).

It can be categorized into three levels: network, server, and user.

Network-level characterization examines the network traffic between the user workstations and the server. In [52], Bangun *et al.* investigated the distributions of packet interarrival time and packet payload size for two MOG systems, *QuakeWorld* and *StarCraft*. It was found that in certain cases the distributions were independent of the number of users logged on to the game. Borella also attempted to characterize packet size and packet interarrival time for *Quake* [53]. A variety of distributions, including deterministic, exponential, and extreme value distributions, were considered. In a study similar to Borella's, a network traffic model was presented for *CounterStrike* [54]. Further studies on MOG network traffic can be found in [55–59].

Attempts to analyze MOG performance at the server level have, so far, been limited. Abdelkhalek *et al.* measured the behavior of a *Quake* game server in terms of throughput, network bandwidth requirement, and the processing time of the various tasks performed by the server [60]. In a follow-up to this work, the authors studied a parallel version of the game server [61] and the corresponding improvement in server performance.

Finally, at the user level, the focus has been on the users' reactions towards the network- and system-level behaviors of a MOG system. Several studies have evaluated the impact of packet loss and update delay on user performance in different types of MOGs [55–57, 62]. In particular, methodologies for evaluating user level performance were proposed in [55–57], and different measures such as the time for a user to complete a designed task and the probability that a user hits a target were developed. Participating players were also interviewed for their subjective experience. It was found that packet loss had little influence on user perceived performance, but update delay had a noticeable effect, especially in first-person shooting games.

2.6 Concluding Remarks

In this chapter, we presented a survey of the previous research related to the performance and scalability of DVE systems. From our investigation, we conclude that not much is known about the performance characteristics of the two-level hierarchical architecture. For example, how this architecture scales to a large user population, how the update delay may be analyzed, and the impact of server location on the update delay have not been well-studied. These topics will be investigated in this thesis.

Note that a hierarchical architecture may have more than two levels. However, we are not able to find any publications in the open literature about a DVE system with more than two levels.

Chapter 3

Scalability of the Two-level Architecture

This chapter investigates the scalability of the two-level hierarchical architecture. This involves the development of performance models for the overall system, virtual environment and vision domain, and the use of analytic results to illustrate the scalability of the architecture.

3.1 Performance Model

In this section, we develop a performance model for the two-level hierarchical architecture. This architecture consists of multiple servers (or server clusters) interconnected by a network. We refer to each of these servers as a "basic system." The various basic systems may be co-located or geographically distributed. Each basic system has a number of assigned users. Figure 3.1 shows a two-level architecture with two basic systems. Suppose user u is assigned to basic system i (denoted by BS_i). BS_i is referred to as the "local basic system" of user u, and user u is said to be a "local user" of BS_i.

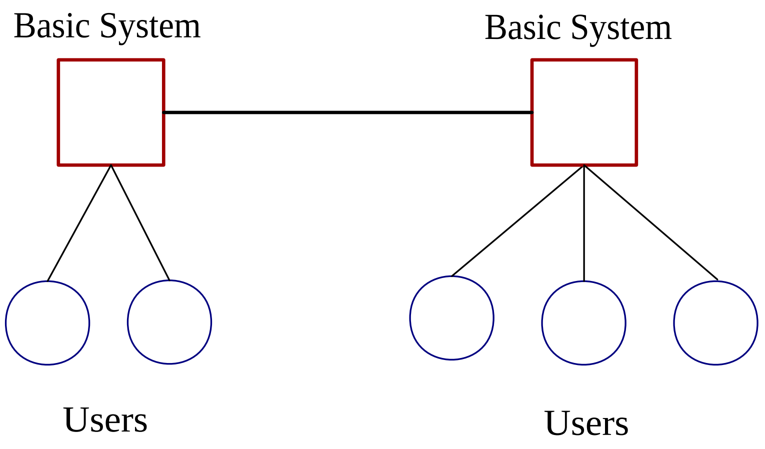


Figure 3.1: A Two-level Hierarchical Architecture with Two Basic Systems

At the user side, there is typically a DVE client program running on the workstation. This program accepts input commands from the user, which may trigger a change in the state of his avatar. For example, suppose a user decides to move his avatar to a new location. This will change his avatar's state, namely its position. Any such change is transmitted, in the form of an update packet, to the user's local basic system for processing, and rendered on the user workstation. When this update packet is received, the local basic system executes the DVE-specific logic (e.g., verifying the eligibility of the update), applies changes to its copy of the virtual environment, and distributes the update to the client programs of those users who are affected by the changes. Each client program, upon receiving the update, renders the changes in the virtual environment on its corresponding workstation. An affected user may be located at the local basic system or at a remote basic system. In the former case, the update is transmitted directly to the user. In the latter case, an update packet at the higher level of the hierarchy, referred to as a “syn packet”, is transmitted to this remote basic system. When this syn packet is processed, the remote basic system updates its own copy of the virtual environment, and distributes the update to its affected local users.

Based on the above description, two types of packets arrive at each basic system: update packets from its local users, and syn packets from some other basic systems. These packets join a queue at the basic system, awaiting processing (see Figure 3.2). When a syn packet is processed, the update in it is distributed to affected local users. On the other hand, when an update packet is processed, a syn packet may be generated for transmission to some other basic system(s), in addition to distribution of the update to affected local users, if any. Note that whether a syn packet is transmitted to some other basic system(s) is dependent on whether there are affected users there. We develop a model of the virtual environment and vision domain that can be used to determine the probability that such a syn packet will be generated. This model will be described in the next section.

3.2 Model of a Virtual Environment

At each basic system, a copy of the virtual environment (VE) is maintained. Such an environment is often very complex [8]. Its behavior is difficult to analyze and costly to simulate. In our study, we consider a model of the VE which is simple and

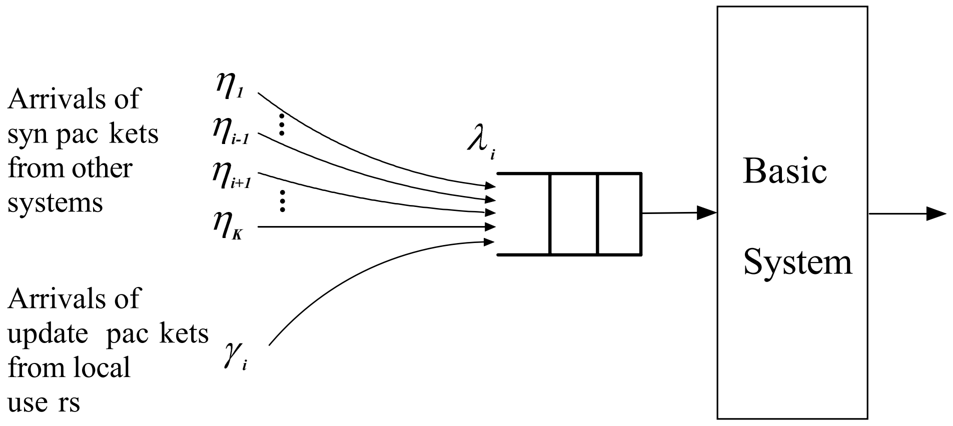

Arrivals of syn pac kets from other systems
η_1
η_{i-1}
η_{i+1}
η_K
Arrivals of update pac kets from local use rs
γ_i
λ_i
Basic System

Figure 3.2: Arrivals of Update and Syn Packets to a Basic System

yet realistic enough to gain insights into the performance characteristics of DVE systems.

For a DVE, it is common that the VE is organized using a xy-coordinate system. We therefore model our VE as a two-dimensional area organized as a unit square grid. Let A and B (in number of units) be the width and height of the VE, respectively. We index the width of the VE from left to right by $0, 1, \ldots, A-1, A$ and the height from top to bottom by $0, 1, \ldots, B-1, B$ (see Figure 3.3). Avatars can only be located at a grid intersection; one or more avatars can be located at any given intersection. Note that in our VE model the granularity of a unit may be varied. Our model is therefore general enough to represent different VE organizations, e.g., rooms & hallways [14] or a maze [10].

A user can perform various actions within the VE, e.g., moving from one place to another and interacting with the other users. In our investigation, we focus on

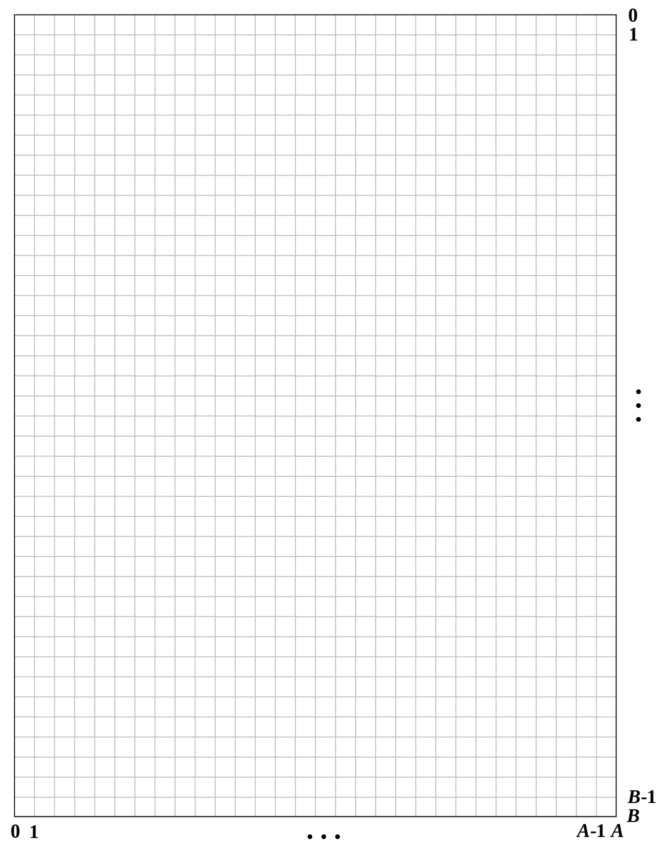


Figure 3.3: An Example VE Model

user movement because it is the most common action performed by a user [10, 25]. We assume for simplicity that the movement of any given user is independent of the behavior of the other users. We also assume that the movement of each user is modeled by a Markov Chain, which is consistent with the measurement data that we have obtained for an existing DVE [63]. When a user makes a move, he chooses one of the four possible directions (up, down, left, and right) according to a probability distribution and moves his avatar in the chosen direction by one unit (or step). We assume that this distribution is the same for all users. Let $q_{a,b;c,d}$ be the probability that a user moves from location (a, b) to (c, d) in one step. Note that

$$q_{a,b;c,d} = 0 \qquad \text{if} \quad |c-a| > 1 \quad \text{or} \quad |d-b| > 1.$$

Since a user cannot move out of the VE, we also have $0 \le a, c \le A$ and $0 \le b, d \le B$. Finally, for simplicity, the time until the user makes the next move is assumed to be exponentially distributed.

It follows from the above assumptions that at steady state, the probability that a user is at location (a, b) (denoted by $p_{a,b}$) can be obtained by solving the following set of equations:

$$\begin{aligned} p_{c,d} &= \sum_{a=0}^{A}\sum_{b=0}^{B} p_{a,b}q_{a,b;c,d} \qquad \text{for } c = 0,1,\ldots,A, \quad \text{and} \quad d = 0,1,\ldots,B \\ \sum_{a=0}^{A}\sum_{b=0}^{B} p_{a,b} &= 1. \end{aligned} \tag{3.1}$$

We next define our model of a vision domain. As discussed in Section 2.2.2, a state update from a user is sent only to those who are within this user's vision domain (in other words, who are affected by the update). For any user, we assume that his vision domain is a rectangle with width U and height V, and that the user's avatar is centered at this rectangle (see Figure 3.4). This implies that U and V are even numbers. An example of a vision domain of size 6×4 ($U = 6, V = 4$) is shown in Figure 3.4. The above assumption is consistent with the common practice that the vision domain of a user in a DVE is the area surrounding the user [9].

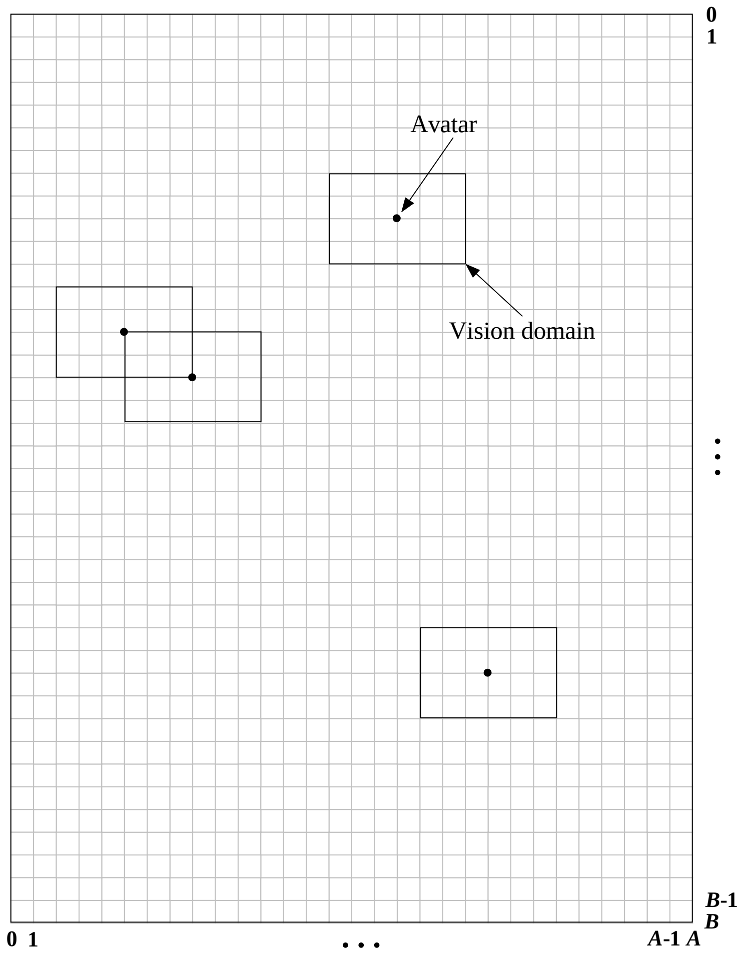


Figure 3.4: Vision Domain Model

3.3 Analysis of Arrival Rates

In this section, we present analytic results for the arrival rates of update and syn packets to a basic system, based on our models described in Sections 3.1 and 3.2.

Our analysis is for the case that the number of logged-on users at each basic system is a constant. We use N_i to denote the number of logged-on users at BS$_i$, $i = 1, 2, \ldots, K$, where K is the total number of basic systems. We recognize that N_i typically varies over time. Including such variations would introduce complexity in the analysis. N_i can be viewed as the maximum number of logged-on users. Our analysis is then for the worst case scenario where the number of logged-on users is always at the maximum. Such a scenario is of interest when one considers the issue of performance and scalability.

Arrival Rate of Update Packets

Consider first the arrival rate of update packets to BS$_i$, $i = 1, 2, \ldots, K$. Recall that when a user makes a move, a state update, in the form of an update packet, is generated and sent to his local basic system. The arrival rate of such update packets is determined by how frequently users make their moves (or submit their updates). Let ϕ be the rate at which update packets are submitted by a user, which is assumed to be the same for all users. The arrival rate of update packets to BS$_i$, denoted by γ_i, is given by:

$$\gamma_i = N_i \phi. \tag{3.2}$$

Arrival Rate of Syn Packets

We next determine the arrival rate of syn packets to BS$_i$. These packets are sent from the other basic systems. Consider the transmission of syn packets from BS$_k$ to BS$_i$ ($i \neq k$). Upon processing an update packet from a local user, BS$_k$ checks to see if there are users at BS$_i$ who are within the local user's vision domain. If so, a syn packet is sent to BS$_i$, as discussed in Section 3.1.

To determine the rate of syn packets transmitted from BS$_k$ to BS$_i$, we assume that the system is in steady state. Consider a "tagged" user at BS$_k$. Suppose this user is currently at location (a, b) in the VE. For any other user, the probability that he is within the tagged user's vision domain is:

$$h(a,b) = \sum_{x=x'}^{x^*} \sum_{y=y'}^{y^*} p_{x,y} \tag{3.3}$$

where $p_{x,y}$, the probability that a user is at location (x, y), is given by Equation 3.1, $x' = max\{0, a-\frac{U}{2}\}$, $x^* = min\{A, a+\frac{U}{2}\}$, $y' = max\{0, b-\frac{V}{2}\}$, and $y^* = min\{B, b+\frac{V}{2}\}$. The variables x', x^*, y' and y^* are defined so that the vision domain is inside the VE boundaries.

Our interest is the probability that after this tagged user has made a move, there are one or more users logged on to BS$_i$, who are within the tagged user's vision domain. This is also the probability that a syn packet is sent from BS$_k$ to BS$_i$. We now derive an analytic expression for this probability, denoted by $g_{k,i}$.

Note that at BS$_i$, each of the N_i logged-on users could be within the tagged user's vision domain. Let $\xi_{k,i}(n)$ be the probability that n users at BS$_i$ are within the tagged user's vision domain. $\xi_{k,i}$ is given by:

$$\xi_{k,i}(n) = \sum_{a=0}^{A} \sum_{b=0}^{B} \left[\binom{N_i}{n} (h(a,b))^n (1 - h(a,b))^{N_i - n} \right] p_{a,b} \tag{3.4}$$

where $h(a,b)$ is given by Equation 3.3. It follows that

$$g_{k,i} = 1 - \xi_{k,i}(0). \tag{3.5}$$

Finally, considering the updates submitted by the N_k users at BS$_k$, the arrival rate of syn packets from BS$_k$ to BS$_i$ is given by:

$$\eta_{k,i} = g_{k,i} N_k \phi \tag{3.6}$$

Summing over all the other basic systems, the arrival rate of syn packets to BS$_i$, denoted by η_i, is:

$$\eta_i = \sum_{k=1, k \neq i}^{K} \eta_{k,i} \tag{3.7}$$

Total Arrival Rate

The total arrival rate of update and syn packets to BS$_i$, denoted by λ_i, is simply the sum of γ_i and η_i. We thus have:

$$\lambda_i = \gamma_i + \eta_i \tag{3.8}$$

where γ_i and η_i are given by Equations 3.2 and 3.7, respectively.

3.4 Results and Discussions

In this section, we present numerical results that illustrate the scalability of the two-level architecture. Suppose there are K basic systems. Let μ_i be the capacity of BS$_i$, measured in number of packets (update or syn packets) processed per unit time. For BS$_i$ to be stable, the total arrival rate to BS$_i$ must be less than μ_i, or the traffic intensity, as given by $\rho_i = \lambda_i/\mu_i$, must be less than 1. In practice, it is desirable to keep λ_i below a certain level (e.g., by admission control) such that $\rho_i \leq y$, for some given value y (e.g., $y = 0.8$ or 0.9); otherwise, the response time (or delay in processing the update or syn packets) may be excessive. Analysis of response time will be considered in Chapter 5.

Suppose there are N logged-on users and these users are distributed equally to the K basic systems. The arrival rate of update packets to any basic system, say BS$_i$, is $\gamma_i = N\phi/K$ (see Equation 3.2). As N increases, γ_i can be reduced by deploying more basic systems (i.e., increasing K). However, syn packets are present when $K > 1$, and the arrival rate of syn packets η_i is affected by factors such as user movement and vision domain size. An important question is whether it is possible to configure a DVE system, such that for each BS$_i$, $\lambda_i = \gamma_i + \eta_i$ is less than the desired level for that BS$_i$. The system is scalable if this is possible. The numerical results in the rest of Chapter 3 will provide insights into this question.

3.4.1 Total Arrival Rate

We first present numerical results for the total arrival rate of packets to each basic system as a function of the number of basic systems K. We note that the arrival rate of syn packets is affected by the size of the vision domain because a larger vision domain means more users are affected by an update. The arrival rate of syn packets is also affected by the density of avatars, which is defined as the number of avatars per grid intersection. This density is determined by the number of logged-on users N and the size of the VE.

In our numerical examples, we assume for convenience that ϕ, the rate at which state updates are sent from a user to his local basic system, is the same for all users. Without loss of generality, we assume that ϕ is equal to 1. We also assume that the VE and vision domain are square in shape. This allows us to use a single parameter, namely E, for both the width and height of the VE (i.e., $A = B = E$), and a single parameter, D, to denote both the width and height of the vision domain (i.e., $U = V = D$). The above assumptions are expected to have a minimal impact on our observations because the scalability of the two-level architecture is affected by the sizes of the VE and vision domain, rather than by their shapes.

Four combinations of N and E are considered in our numerical examples ($N = 500, 1000$ and $E = 100, 150$). This would allow the investigation of different densities of avatars in the VE. Several sizes of the vision domain D are also selected; they are $2, 4, \ldots, 10$ (note that D must be even).

As to user movement, we consider the special case in which an avatar moves to each of the four directions with equal probability, except that when the avatar is on an edge (or at a corner) of the VE, the number of possible directions is reduced to 3 (or 2). This results in $p_{a,b}$ being a uniform distribution, i.e., an avatar is equally likely to be at any grid intersection. One may argue that in practice there may be regions with a high concentration of avatars due to user movement. In our examples, the effect of different concentrations (or densities) is studied by considering different combinations of N and E.

Due to the symmetric and uniform nature of our assumptions, it is sufficient to present results for one of the basic systems, say BS_i.

Consider first the arrival rate of update packets γ_i. As indicated by Equation 3.2, γ_i is a decreasing function of the number of basic systems K. In particular, suppose we already have K basic systems. Deploying an additional basic system would result in $\frac{1}{K+1}$ reduction in γ_i, which implies a diminishing rate of return. Equation 3.2 also indicates that γ_i is directly proportional to the number of users N, and that γ_i is not affected by the sizes of the VE and vision domain (E and D). This behavior is illustrated by Figure 3.5 where we plot γ_i against K for $N = 500$ and 1000.

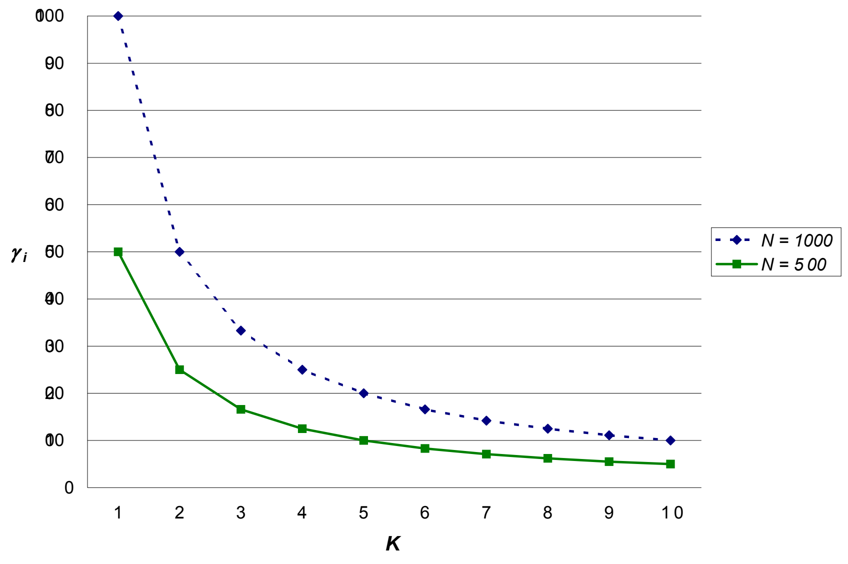
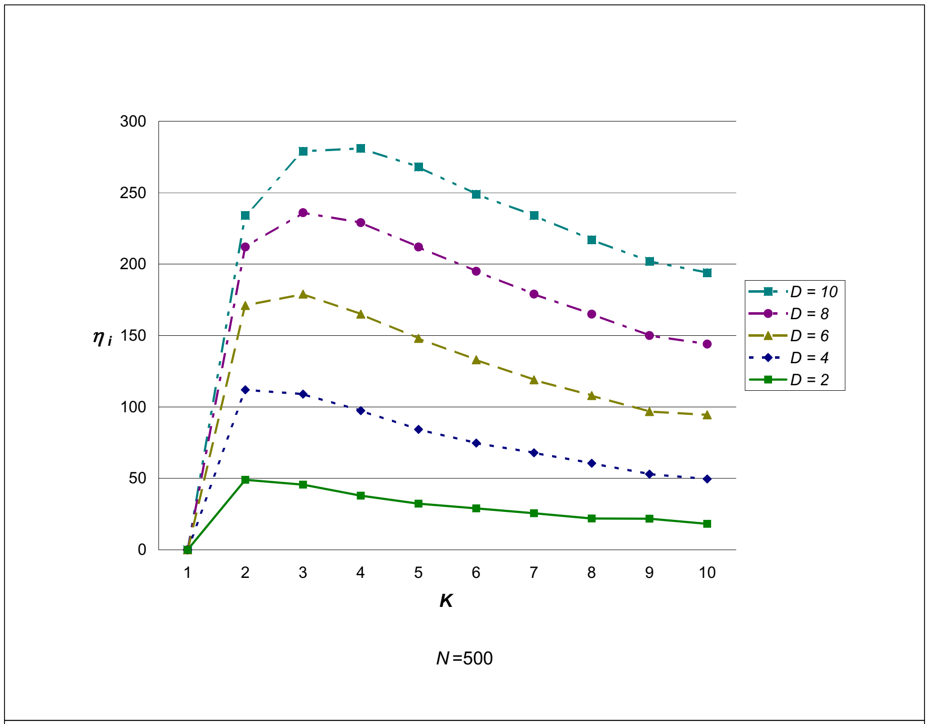


Figure 3.5: Arrival Rate of Update Packets to BS_i γ_i

We next present results for the arrival rate of syn packets η_i, which are computed using Equation 3.7. In Figures 3.6 and 3.7, η_i is plotted against the number of basic systems K for $D = 2, 4, \ldots, 10$, and for the four combinations of N and E. As expected, $\eta_i = 0$ when $K = 1$. This is because no syn packet is sent when there is only one basic system. As K increases, we observe that η_i increases at first, but then decreases quickly as more basic systems are added. This is a result of two opposing effects. For a given N, an increase in K means that fewer users are at BS$_i$ and more users are at the other basic systems. This would tend to increase the rate of syn packets sent from the other basic systems to BS$_i$. Contrarily, with fewer users at BS$_i$, the chance that BS$_i$ has a local user within the vision domain of users at the other basic systems is smaller, thus reducing the number of syn packets from these basic systems. The results in Figures 3.6 and 3.7 show that the latter effect dominates the former as K increases, leading to a reduction in η_i.

Comparing the results for different sizes of the vision domain D in Figures 3.6 and 3.7, we observe that the larger the vision domain, the higher is the arrival rate of syn packets η_i. This is as expected because a larger vision domain means a higher probability of user interaction, leading to a higher rate of syn packets among the basic systems.

We now combine the results for γ_i and η_i to obtain λ_i, the total arrival rate at BS$_i$. The results are shown in Figures 3.8 and 3.9. We observe that λ_i is a decreasing function of K, indicating that the two-level architecture has good properties with respect to scalability. The issue of scalability will be discussed in the next section.



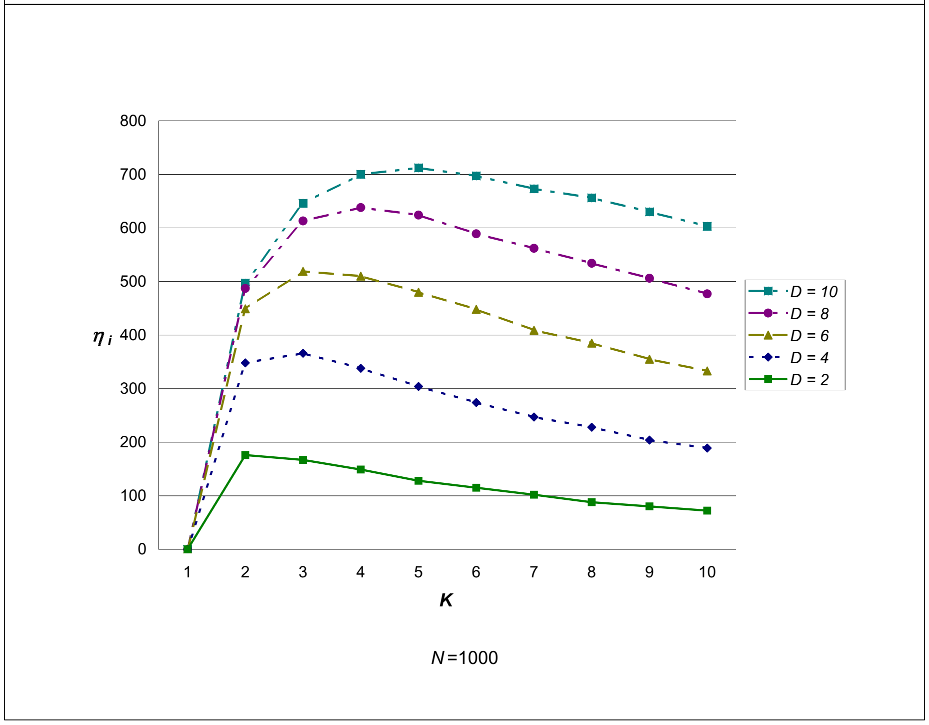
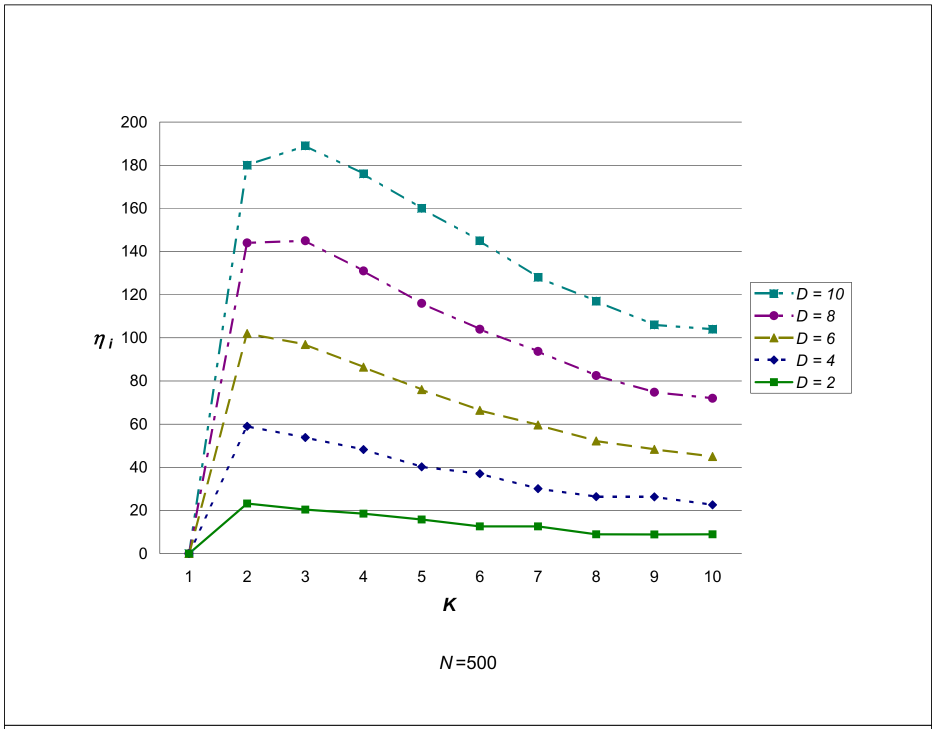


Figure 3.6: Arrival Rate of Syn Packets to BS$_i$ η_i for $E = 100$



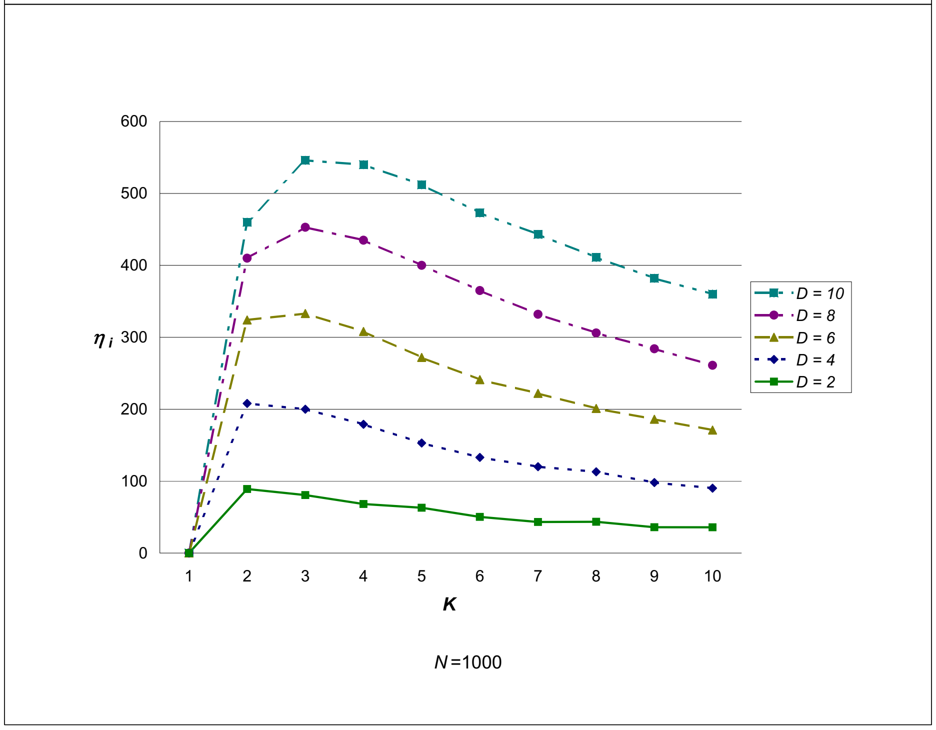


Figure 3.7: Arrival Rate of Syn Packets to BS$_i$ η_i for $E = 150$

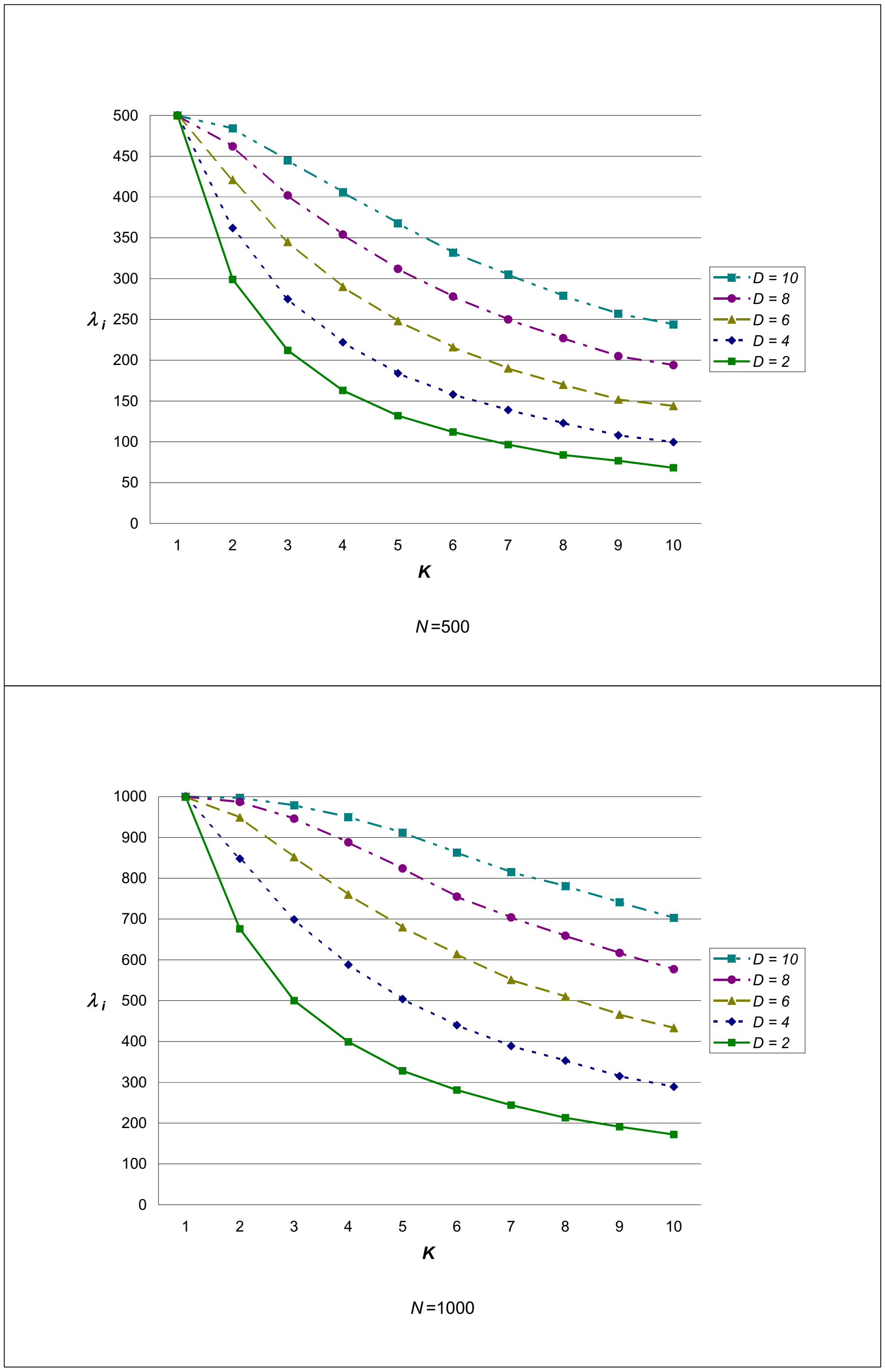


Figure 3.8: Total Arrival Rate to BS$_i$ λ_i for $E = 100$

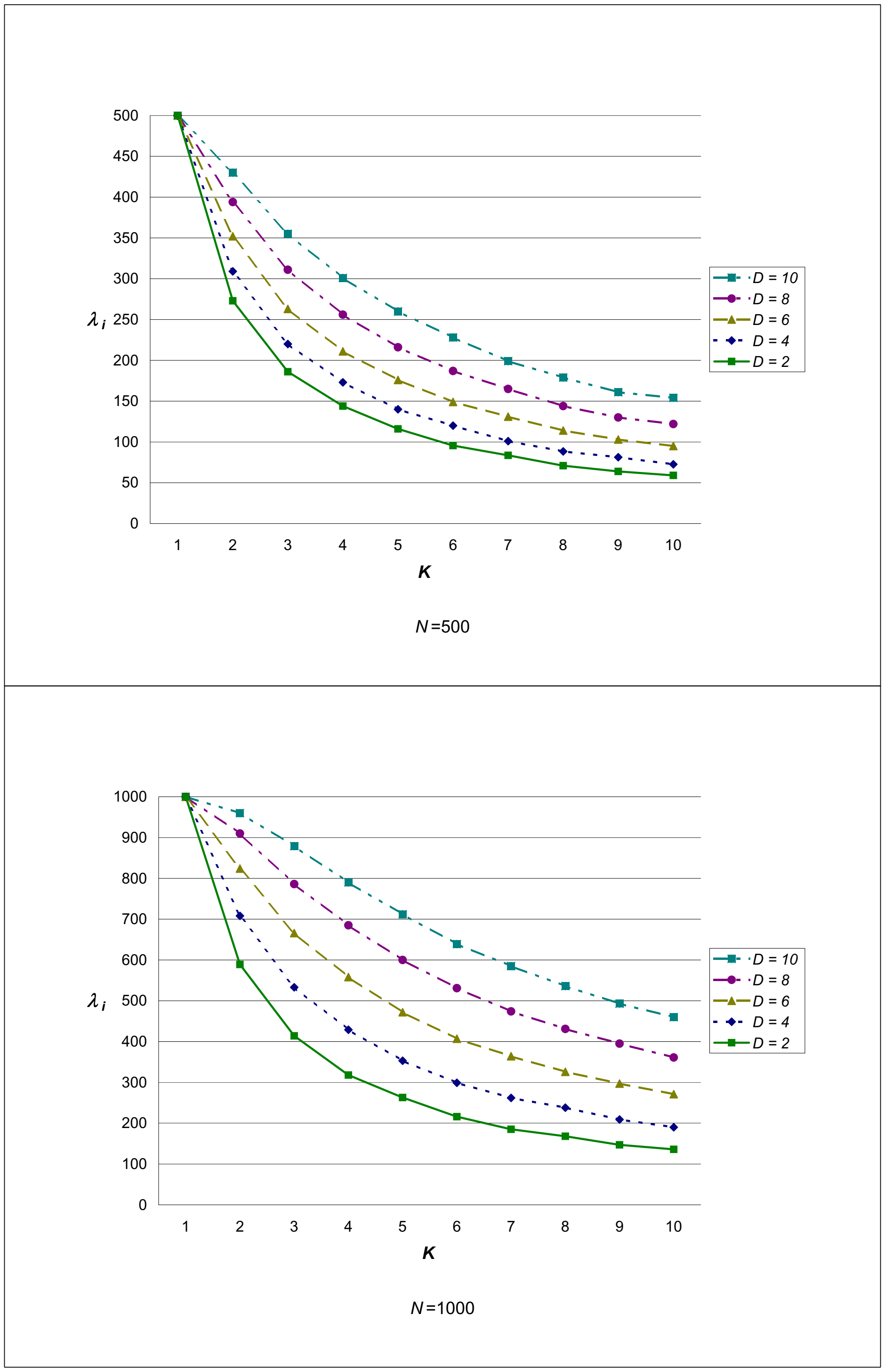


Figure 3.9: Total Arrival Rate to BS_i λ_i for $E = 150$

3.4.2 Scalability

As mentioned in the last section, it is desirable in practice to keep λ_i, the total arrival rate to BS_i, below a certain level so that the traffic intensity

$$\rho_i = \lambda_i / \mu_i \leq y. \tag{3.9}$$

We refer to y as the traffic intensity parameter. From the results in Figures 3.8 and 3.9, we observed a larger λ_i when N is increased from 500 to 1000. This may result in $\rho_i > y$. When this happens, more basic systems may be deployed so that the resulting ρ_i for each basic system is less than y. The system architecture is scalable if this can be done. In our investigation of scalability, we use, as our metric, the minimum number of basic systems required to support N users, while maintaining $\rho_i \leq y$ at each basic system. We denote this number by K_{min}. For illustrative purposes, we consider values of N in the range of 500 to 1000. Two values of y are considered: 0.8 and 0.9. We select the capacity μ_i of each basic system so that ρ_i is less than y when $N = 500$ and significantly larger than y when $N = 1000$, but K_{min} is no more than 10. Using Equations 3.8 and 3.9, we numerically solve for the value of μ_i and find that $\mu_i = 850$.

We first consider the case of $E = 100$. This corresponds to a smaller VE than $E = 150$, and thus results in a higher density of avatars. In Figure 3.10, the minimum number of basic systems required K_{min} is plotted against the number of users N for $y = 0.8$ and 0.9. At $y = 0.8$, $K_{min} = 1$ for $N = 500$ and 600. This means that a single basic system has sufficient capacity to support as many as 600 users. When $N \geq 700$, more than one basic system is required, and K_{min} increases almost linearly with N. This is a good property with respect to scalability. However, the rate of increase in K_{min} is dependent on the size of the vision domain D. For a small D ($D = 2$), 2 basic systems would be sufficient to support $N = 1000$ users (twice the number of users when compared to $N = 500$). Contrarily, when $D = 10$, the minimum number of basic systems required is about 10. This is a significant increase in resource requirement. In general, a larger D means more avatars are potentially within the vision domain of a given user; this has a negative impact on scalability.

The results for $y = 0.9$ are shown in Figure 3.10. We again observe that K_{min}, the minimum number of basic systems required, increases linearly with N, the number of users. This represents additional evidence that the two-level architecture has good scalability. Besides, the results for K_{min} at $y = 0.9$ are generally smaller than those at $y = 0.8$. This is because at $y = 0.9$, a higher utilization is possible at each basic system. Therefore, each basic system is able to support more users, and fewer basic systems are needed. For example, at $y = 0.9$, one basic system is sufficient to support up to $N = 700$ users. When N is increased to 1000, we need only 8 basic systems for a large vision domain $D = 10$. As seen in Figure 3.10, at $y = 0.9$, the size of the vision domain D still has a significant impact on the architecture's scalability.

Consider next the case of a larger VE, $E = 150$. The corresponding results for K_{min} are plotted in Figure 3.11. Similar to our previous observations (for the case of $E = 100$), the results demonstrate that the two-level architecture scales well, and resource requirement is affected by the traffic intensity parameter y and the size of the vision domain D.

We further observe that for a given D, the rate of increase in the number of basic systems required K_{min} (as the number of users N increases) is lower with a larger VE (e.g., $E = 150$). This is because a larger VE means that the density of avatars is smaller and fewer avatars are potentially within the vision domain of a given user. The number of syn packets is reduced, leading to a decrease in the total arrival rate of packets to each basic system. This in turn leads to a smaller number of basic systems required. Consider, for example, the number of users in the DVE N is doubled from 500 to 1000. For $E = 100$ and $D = 10$, K_{min} is increased from 1 to 10 at $y = 0.8$ and from 1 to 8 at $y = 0.9$. For $E = 150$ and $D = 10$, on the other hand, K_{min} is increased from 1 to 6 at $y = 0.8$ and from 1 to 5 at $y = 0.9$. Hence, the scalability of the architecture is also affected the density of avatars, namely that a lower density of avatars yields better scalability.

Note that our analysis can be extended to a VE with regions that have different densities of avatars. For example, one can define the transition probabilities in our user movement model (Section 3.2) such that users are more likely to be in one region than another.

3.5 Concluding Remarks

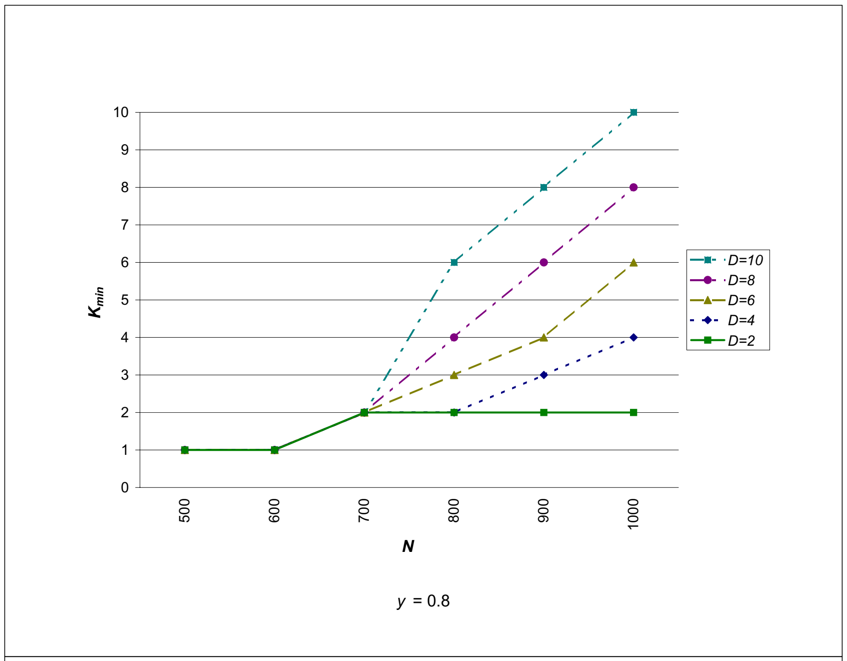
In this chapter, we investigated the scalability of the two-level hierarchical architecture. Our investigation began with analyzing the arrival rate of packets to a basic system in the architecture. We derived the analytic results for the total arrival rate, and the numerical examples presented gave us valuable insights into the impact of the various factors on the scalability. More importantly, they confirmed that the two-level architecture possesses good properties with respect to scalability.

Our analysis can be extended to user actions other than user movement. As an example, for the case of shooting, a state update resulting from the shooting is sent to all affected users, similar to a state update resulting from user movement. However, including these other user actions introduces complexity to our model. Also, since these other actions tend to occur infrequently, we believe that they have a minimal impact on the scalability of the two-level architecture. Therefore, these other actions are not included in our investigation.

There are other factors related to the scalability of the two-level hierarchical architecture. These include the following.

- Consistency among copies of the VE at the various basic systems is an important issue. Processing capacity at the basic systems is required to achieve consistency. This would have a negative impact on the scalability. In Chapter 4, the issue of consistency will be discussed.

- For the two-level architecture, an update from a user may have to be processed first at the local basic system, and then at a remote basic system. This would incur additional delay when compared to the case of an one-level architecture. The server delay will be analyzed in Chapter 5.



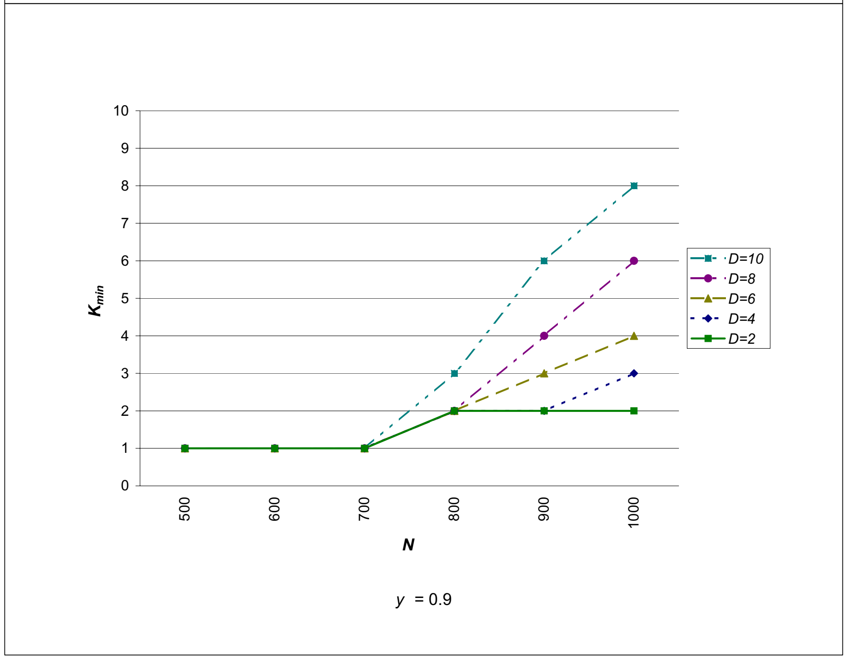


Figure 3.10: Minimum Number K_{min} of Basic Systems Required to Support N Users while $\rho_i \leq y$ for $E = 100$

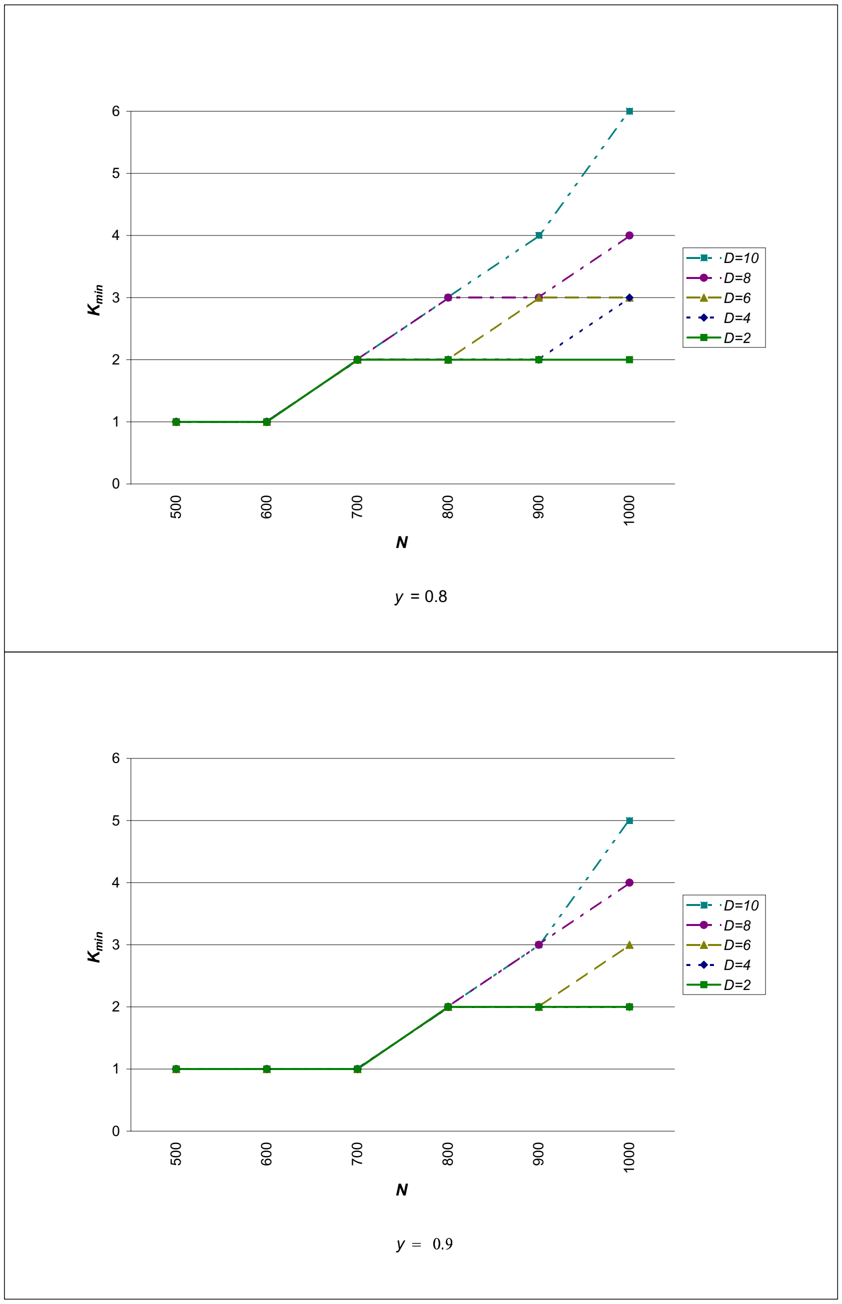


Figure 3.11: Minimum Number K_{min} of Basic Systems Required to Support N Users while $\rho_i \leq y$ for $E = 150$

Chapter 4

Consistency of the Virtual Environment

In the two-level hierarchical architecture, consistency among copies of the VE at the various basic systems is an important issue. Such consistency is necessary for the basic systems to determine accurately if two users are within each other's vision domain. It has been suggested that "periodic-update" packets be exchanged among the basic systems at regular intervals in order to synchronize the current locations of all users in their copies of the VE [14]. We refer to such a technique as "periodic global synchronization." Periodic global synchronization consumes resources and may have a negative impact on scalability. Also, this technique can only achieve weak consistency because inconsistency may occur between successive global synchronizations.

This chapter investigates the issue of consistency in the two-level hierarchical architecture. We first examine how inconsistency may occur and show that consistency may be restored as a result of user movement. Then, we propose a technique called "virtual vision domain," as an alternative to periodic global synchronization, to achieve weak consistency. We also present a simulation study to evaluate its effectiveness in improving consistency and its impact on scalability of the architecture.

4.1 Inconsistency

A basic system receives updates from its local users whenever these users make a move. As a result, the basic system always knows the exact location of its local users. However, this may not be the case for users at the other basic systems. For example, consider two basic systems, BS_i and BS_j. Suppose users u and v are logged on to BS_i and BS_j, respectively. When user v makes a move, in order for BS_i to have up-to-date information on this user's location, BS_j needs to send BS_i a syn packet containing the state update. This happens only if one or more users logged on to BS_i are within user v's vision domain, according to BS_j's copy of the VE. Otherwise, BS_i's copy of the VE will not reflect the new location of user v.

Generally, a global view of the VE contains the up-to-date locations of all the users in the VE. In our investigation, consistency is defined with respect to this global view. Let VE_i be the copy of the VE at BS_i. For any given user (say user u) at BS_i, this user is in the "consistent" state if the content of his vision domain, as shown by VE_i, is the same as that shown in the global view; otherwise, user u is in the "inconsistent" state. During user movement, there are two scenarios that may result in user u entering the inconsistent state.

Scenario I. User u's vision domain contains users who should not be there according to the global view.

Scenario II. User u's vision domain does not show users who are supposed to be there according to the global view.

Scenario I can be illustrated by the example depicted in Figure 4.1. There are two basic systems. At each basic system, a solid circle represents a local user, while a hollow circle represents a user at the other basic system (or remote basic system).

- At time t_0, users u and v are in each other's vision domain. Both of them are in the consistent state (i.e., the contents of their respective vision domains are consistent with the global view).

- At time t_1, user v at BS_j moves to the left by one step. Since there are no users within user v's vision domain after the move, BS_j will not send any syn packet to BS_i. As a result, user u's vision domain still contains user v. User u is now in the inconsistent state. Note that user v remains in the consistent state because his vision domain is empty, which is the same as that shown in the global view.

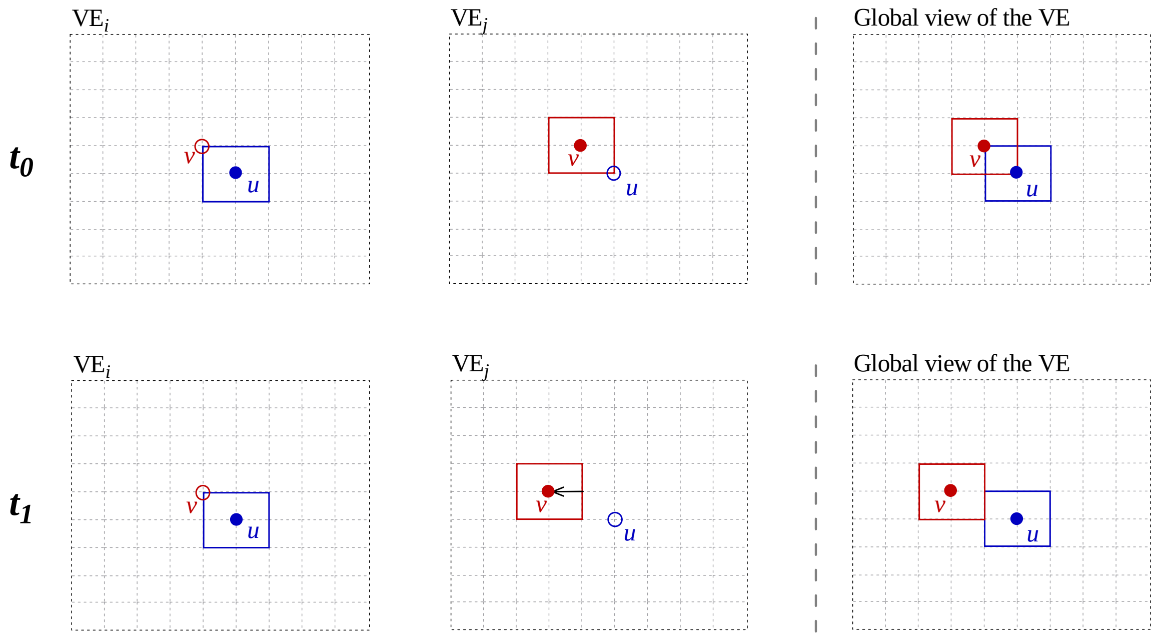


Figure 4.1: Inconsistency: Scenario I

An example that illustrates Scenario II is as follows (see Figure 4.2).

- At time t_0, both users u and v are in the consistent state. They are not in each other's vision domain.
- At time t_1, user v at BS$_j$ moves down by one step. No syn packet is sent from BS$_j$ to BS$_i$ because according to VE$_j$, user u is still not within user v's vision domain after the move. Nevertheless, both users remain in the consistent state because their vision domains are both the same as those shown in the global view.
- At time t_2, user u at BS$_i$ moves to the left by one step. No syn packet is sent from BS$_i$ to BS$_j$ because VE$_i$ shows that there are no users within user u's vision domain after the move. However, with respect to the global view, the two users, u and v, are now in each other's vision domain. Both users have thus entered the inconsistent state.

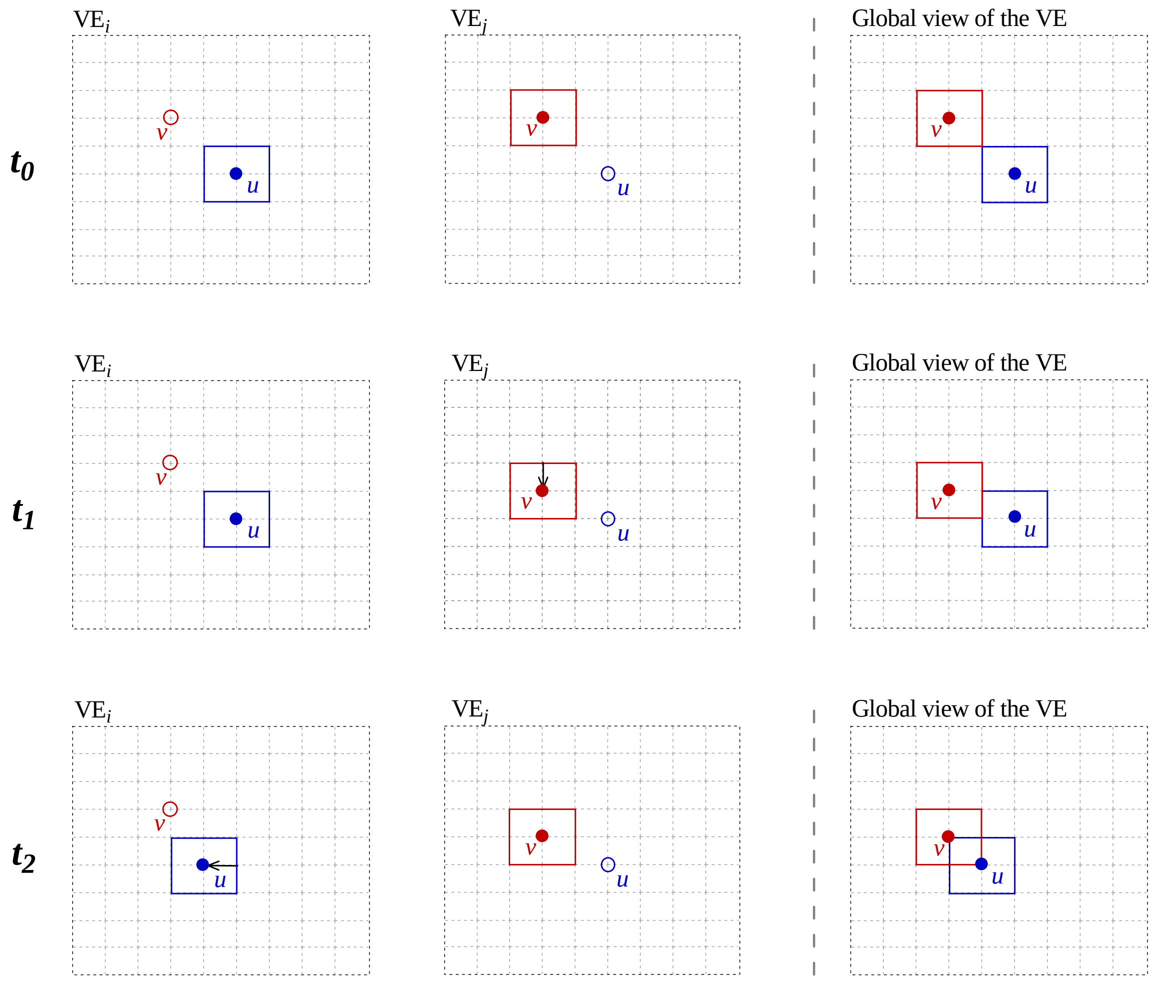


Figure 4.2: Inconsistency: Scenario II

Further investigation of consistency reveals that a user in the inconsistent state may return to the consistent state as a result of user movements in the future. For instance, consider again the example for Scenario I shown in Figure 4.1. The locations of users u and v at time t_1 are reproduced in Figure 4.3. Recall that user u is in the inconsistent state at time t_1. Suppose the future user movements are as follows.

- At time t_2, user u moves to the left by one step. Since according to VE_i, user v is within user u's vision domain (even though his location is incorrectly shown), a syn packet is sent from BS_i to BS_j. At BS_j, the processing of this syn packet will update the new location of user u at BS_j. User v remains in the consistent state. User u is still in the inconsistent state because his vision domain is different from that of the global view.

- At time t_3, user v at BS_j moves down by one step. BS_j sends BS_i a syn packet containing user v's new location. At BS_i, after processing this syn packet, user u returns to the consistent state.

As another example, we consider the example for Scenario II shown in Figure 4.2. The locations of users u and v at time t_2 are reproduced in Figure 4.4. Recall that both users are in the inconsistent state at time t_2. Suppose the future user movement is as follows.

- At time t_3, user v at BS_j moves to the right by one step. According to VE_j, user u is within user v's vision domain; a syn packet is sent from BS_j to BS_i. Upon processing this packet, BS_i has the up-to-date location of user v,

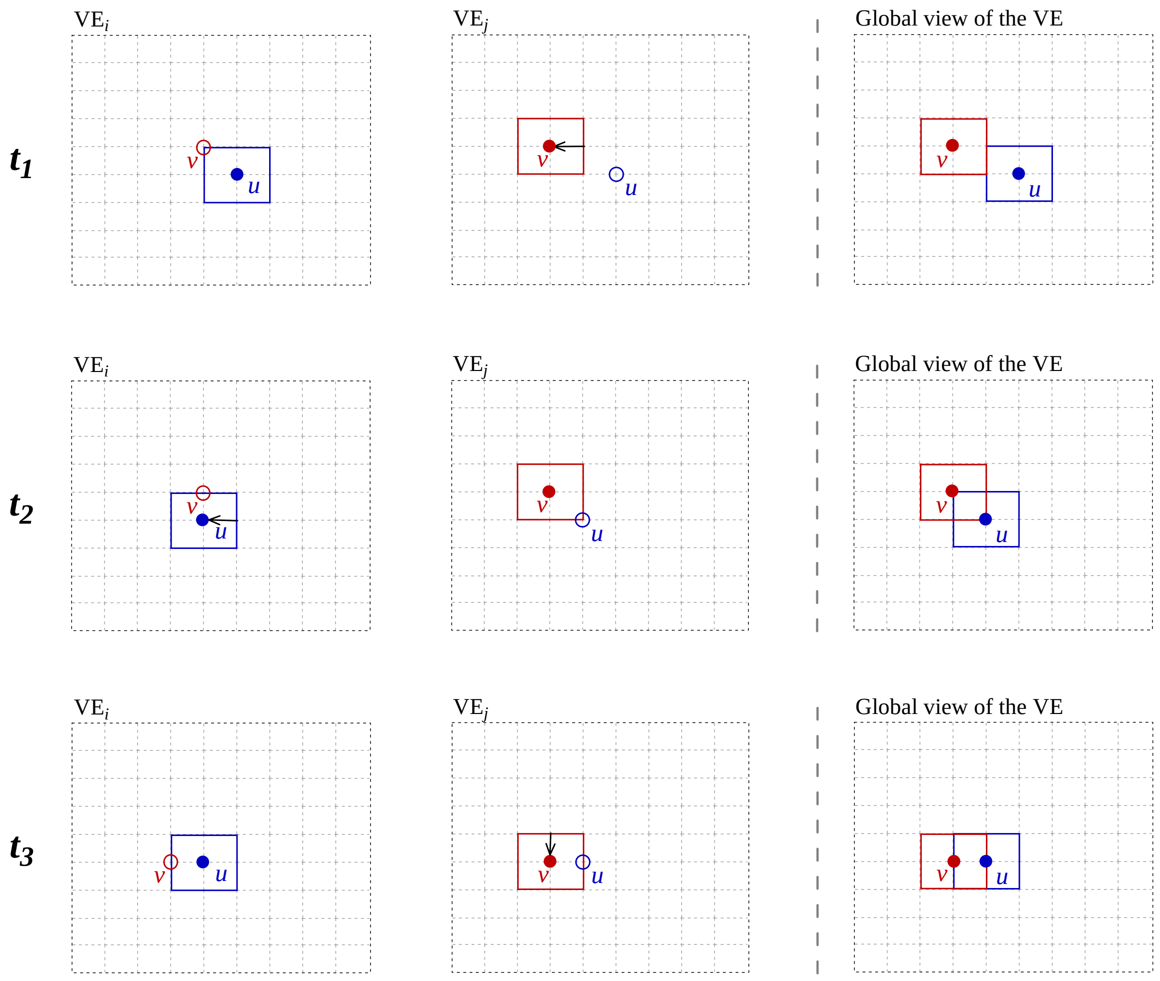


Figure 4.3: Scenario I Inconsistency: How Consistency is Restored

and user u returns to the consistent state. User v, however, is still in the inconsistent state; further user movements may result in his state becoming consistent again.

The above examples show how users u and v may enter the inconsistent state and then return to the consistent state because of user movement. In some instances, inconsistency can actually be avoided. Consider again the example for Scenario II shown in Figure 4.2. Suppose there is a third user, w, logged on to BS_i. Figure 4.5 depicts the location of this user. With the presence of user w, the same sequence of movements by users u and v will not result in inconsistency. This can be explained as follows.

- At time t_0, users u, v, and w are in the consistent state.
- At time t_1, user v at BS_j moves down by one step. Since user w is within the vision domain of user v according to VE_j, a syn packet is sent to BS_i. BS_i has the up-to-date location of user v, and both users u and w remain in the consistent state.
- At time t_2, user u moves to the left by one step. User v is now in the vision domain of user u according to VE_i. A syn packet is sent from BS_i to BS_j. BS_j has the up-to-date location of user u, and user v remains in the consistent state.

For the remainder of this chapter, we will investigate the issue of consistency among copies of the VE by focusing on the inconsistency that may arise. We will

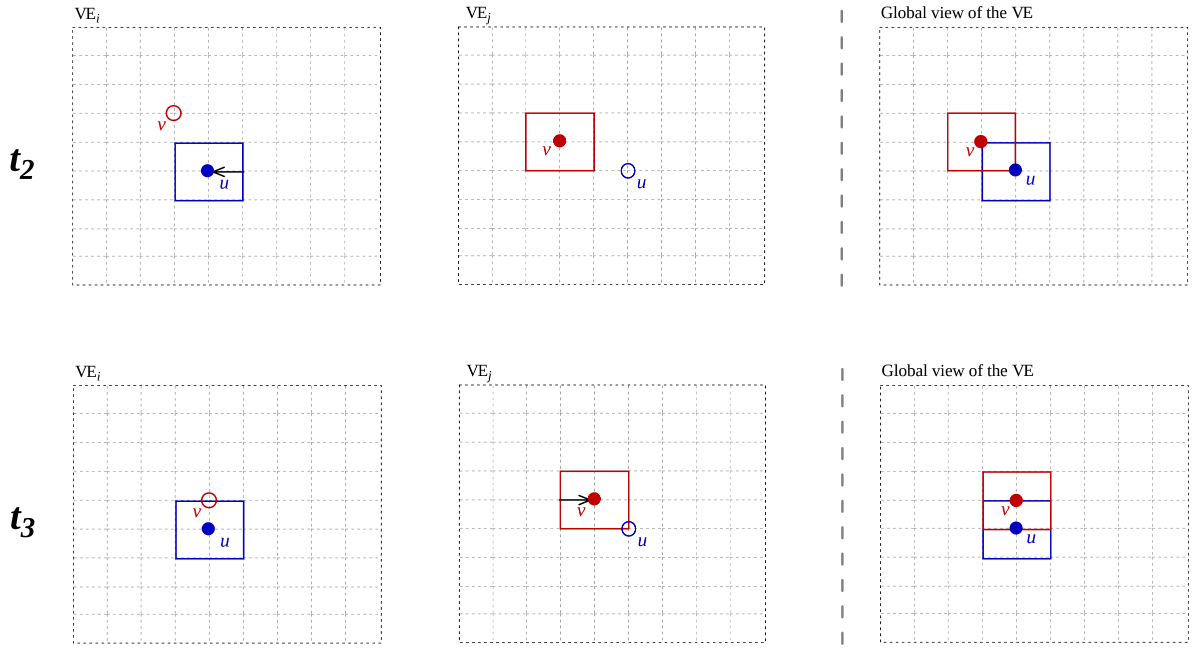


Figure 4.4: Scenario II Inconsistency: How Consistency is Restored

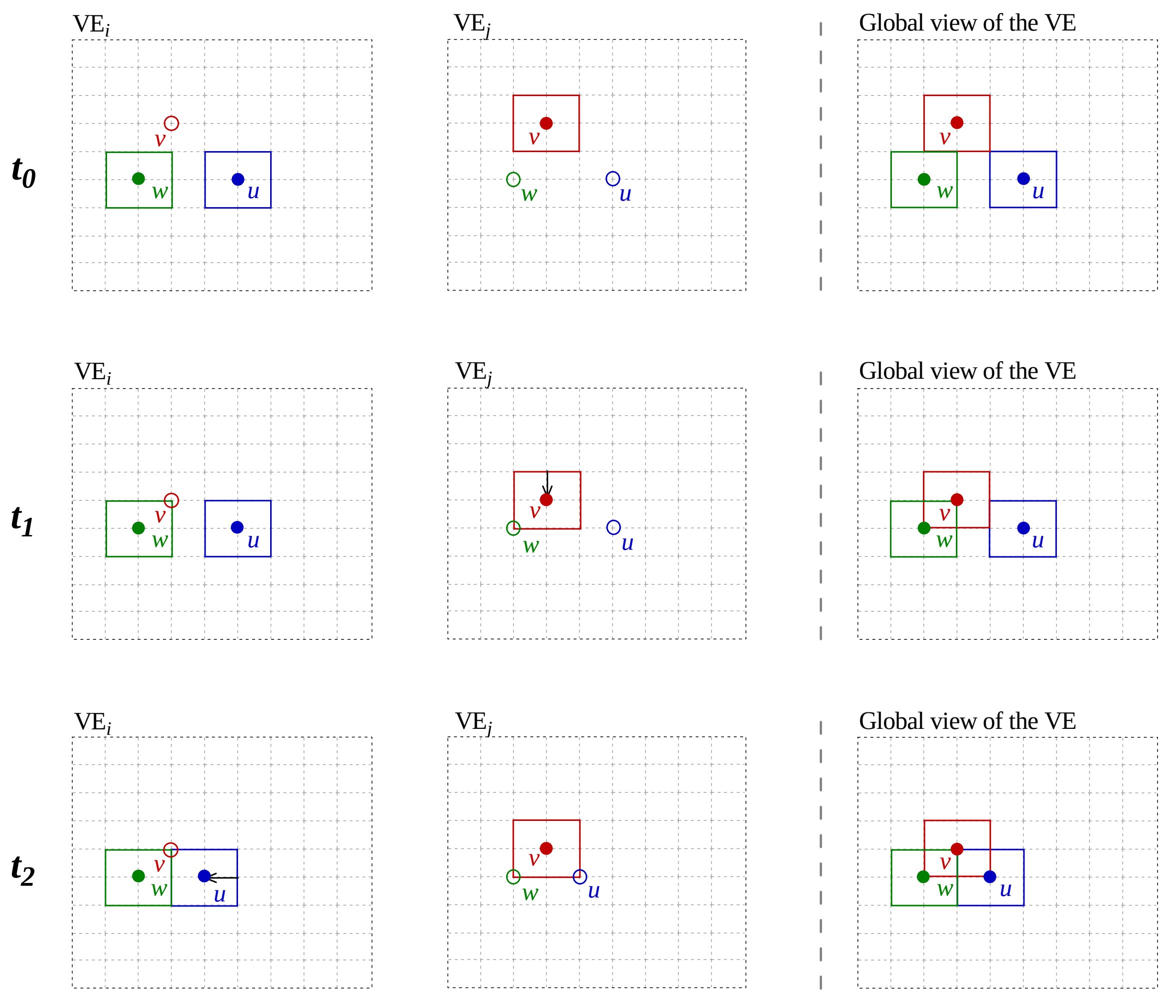


Figure 4.5: How Inconsistency is Avoided with the Presence of a Third User

look at how the inconsistency is affected by the various factors, and how the level of inconsistency may be reduced.

4.2 Preliminary Observations

We conduct simulation experiments to evaluate the extent of inconsistency that may occur among copies of the VE at the basic systems. Our simulation program implements our models of the two-level hierarchical architecture and VE, as described in Sections 3.1 and 3.2, respectively. To verify the correctness of our simulation program, we obtained simulation results for the arrival rates of update and syn packets using our program, and confirmed that they are consistent with the corresponding analytic results in Section 3.3.

Of interest is the number of users who are in the inconsistent state at a given time t, and we denote it by $I(t)$. Recall that a user is said to be in the inconsistent state if the content of his vision domain, as shown in his local basic system's copy of the VE, is not the same as that shown in the global view of the VE.

Our simulation is based on the performance model described in Chapter 3. The definition of consistency is extended to more than two basic systems. This extension is straightforward because consistency is defined with respect to the global view. For each experiment, we run a time-driven simulation for 50 time units and collect data for $I(t)$, for $t = 0, 1, \ldots, 50$. A total of 100 replications are made. Using the data collected, we construct distributions of $I(t)$. For all experiments that we conducted, the distributions of $I(t)$ converge to their respective steady state distributions as t increases.

In our first experiment, the input parameters are shown in Table 4.1. Using the parameter values in Table 4.1, the minimum number of basic systems required K_{min} is 4 (see Figure 3.10).

N	Number of users	1000
E	Size of the VE	100
D	Size of the vision domain	4
y	Traffic intensity parameter	0.8
μ_i	Processing capacity of BS$_i$, same for all i	850
$q_{a,b;c,d}$	User movement probability	uniformly distributed

Table 4.1: Parameter Values Used in the First Experiment

Let $\overline{I}(t)$ be the mean number of users who are in the inconsistent state at time t. In other words, $\overline{I}(t)$ is the mean of the distribution of $I(t)$. The behavior of $\overline{I}(t)$ as a function of t is shown in Figure 4.6. At $t = 0$, every copy of the VE is assumed to be synchronized, so we have $\overline{I}(0) = 0$. Starting from $t = 0$, $\overline{I}(t)$ increases with t, and when $t \approx 30$, steady state is reached. We denote the steady state results for $\overline{I}(t)$ by $\overline{I}$. In Figure 4.6, we observe that $\overline{I}$ is approximately equal to 397 (of 1000). The existence of steady state behavior illustrates the facts that user movement may lead to inconsistency and that consistency can be restored without employing any explicit measures, as discussed in Section 4.1.

In our next experiment, we use the input parameters shown in Table 4.1 with the exception that $y = 0.9$ instead of 0.8. The corresponding K_{min}, as determined from Figure 3.10, is 3. Again, we observe that steady state exists although the mean number of users in the inconsistent state $\overline{I}$ is 285 (out of 1000). This is smaller than that for $y = 0.8$ and $K_{min} = 4$, which is 397. This can be explained

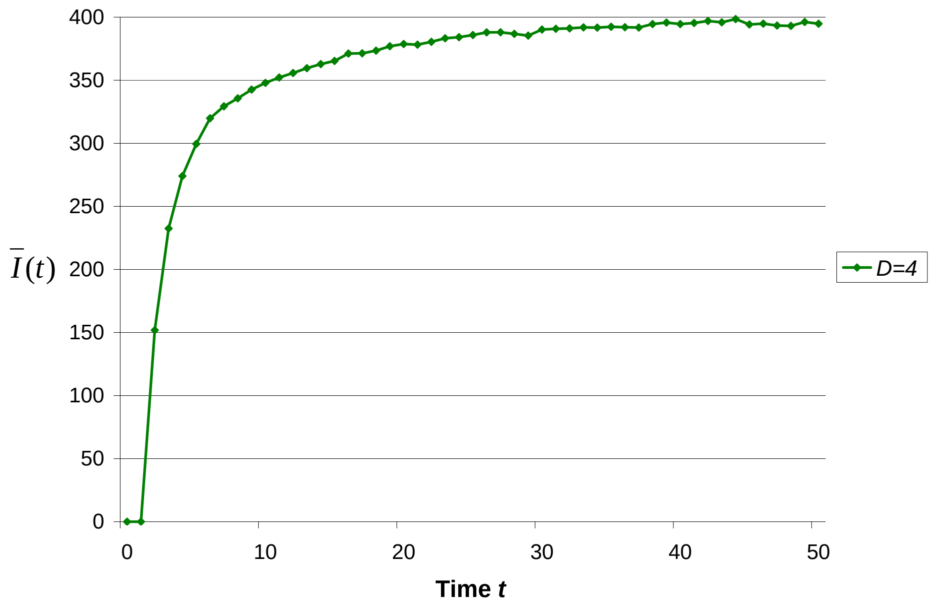


Figure 4.6: Mean Number of Users in the Inconsistent State $\overline{I}(t)$ for $E = 100$, $N = 1000$, $D = 4$ and $y = 0.8$

as follows. When the number of basic systems is larger (4 instead of 3 in this case), more users are located at some other basic systems. This would tend to increase the chance that a user enters the inconsistent state because remote users are those who can potentially cause inconsistency. Hence, $\overline{I}$ tends to be higher when more basic systems are deployed.

One technique for improving consistency is periodic global synchronization [14]. With this technique, all copies of the VE are synchronized periodically. Consider again the results in Figure 4.6. Suppose periodic global synchronization is performed every 10 time units. We would expect in Figure 4.6 that $\overline{I}(t)$ drops to almost zero[1] every 10 time units and increases again to its steady state behavior (i.e., approximately 397). This is depicted in Figure 4.8. In that figure, we see that inconsistency still exists between successive global synchronizations, but the use of periodic global synchronization leads to a reduction in the long-term average of $\overline{I}(t)$.[2] If the time interval between successive synchronizations (referred to as the synchronization interval) is shorter, the long-term average would be reduced further. For instance, Figure 4.9 shows the behavior of $\overline{I}(t)$ when the synchronization interval is 5 time units.

Global synchronization generally involves exchanging state update information among all the basic systems and processing these updates. This consumes processing resources at the basic systems. When global synchronization is performed

[1]Due to potential server delay and network delay experienced by the global synchronization messages, users in the inconsistent state may not return to the consistent state at the same time. Therefore, a certain level of inconsistency may still exist when global synchronization is performed.

[2]The long-term average is given by $\frac{1}{L}\int_0^L \overline{I}(t)dt$ where L denotes the length of observation period.

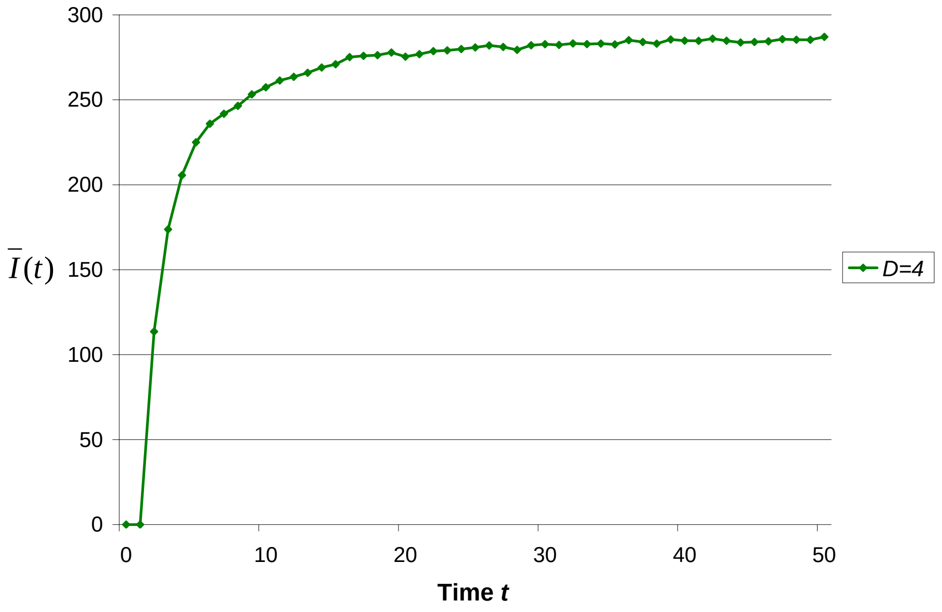


Figure 4.7: Mean Number of Users in the Inconsistent State $\overline{I}(t)$ for $E = 100$, $N = 1000$, $D = 4$ and $y = 0.9$

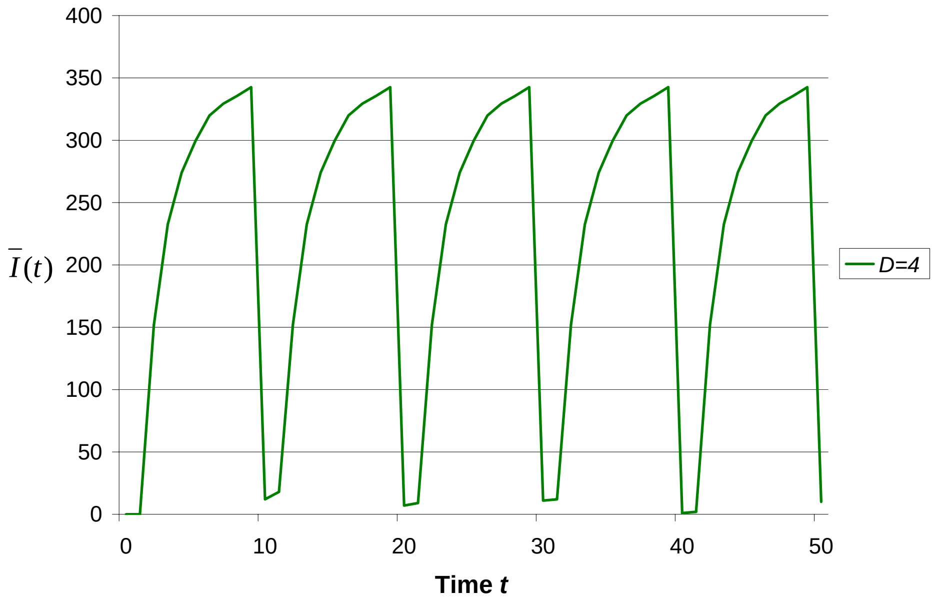


Figure 4.8: Mean Number of Users in the Inconsistent State $\overline{I}(t)$ with Periodic Global Synchronization (Every 10 Time Units) for $E = 100$, $N = 1000$, $D = 4$ and $y = 0.8$

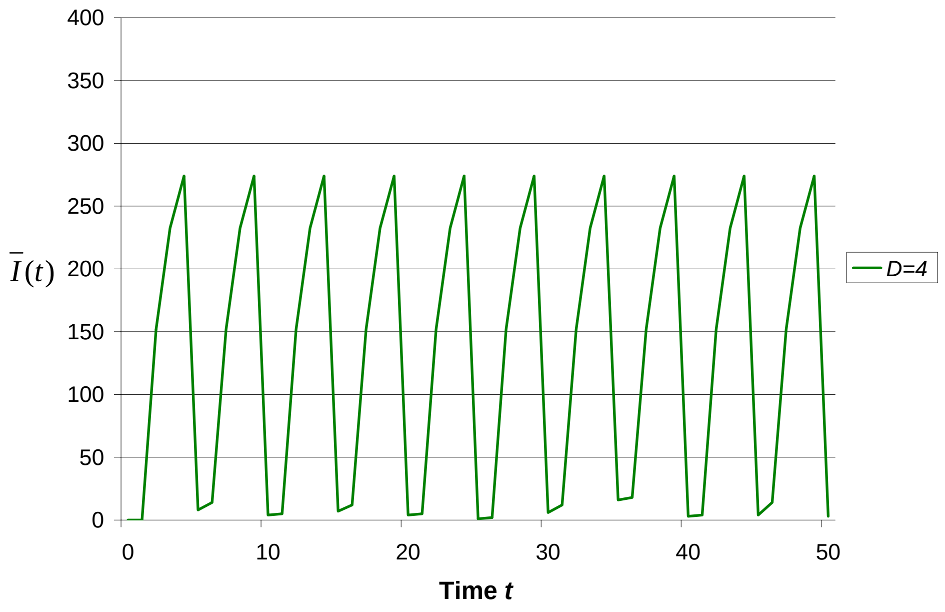


Figure 4.9: Mean Number of Users in the Inconsistent State $\overline{I}(t)$ with Periodic Global Synchronization (Every 5 Time Units) for $E = 100$, $N = 1000$, $D = 4$ and $y = 0.8$

more frequently (by choosing a shorter synchronization interval), more resources are required although better consistency is achieved. With periodic global synchronization, the capacity at each basic system available for processing update and syn packets is therefore reduced. This has a negative impact on scalability because more basic systems may be required to support the same number of users. A detailed evaluation of the effect of periodic global synchronization is out of the scope of this thesis.

As an alternative to periodic global synchronization, we propose a technique called the "virtual vision domain," which will be investigated in the next section.

4.3 Virtual Vision Domain

The basic idea is to extend a user's vision domain to a larger size. We refer to the extended vision domain as the *virtual* vision domain and the original one as the *real* vision domain. The virtual vision domain is used when a basic system wishes to determine whether a local user's update (in the form of a syn packet) is to be distributed to another basic system. The real vision domain is used when the state of a user (consistent or inconsistent) is to be determined. The relationship between the real and virtual vision domains is shown in Figure 4.10.

The use of the virtual vision domain avoids some inconsistencies which would occur if the technique were not used. Consider again the example of inconsistency for Scenario I shown in Figure 4.1. The contents of the VEs at time t_0 are reproduced in Figure 4.11. Suppose we use a virtual vision domain which is 2 units

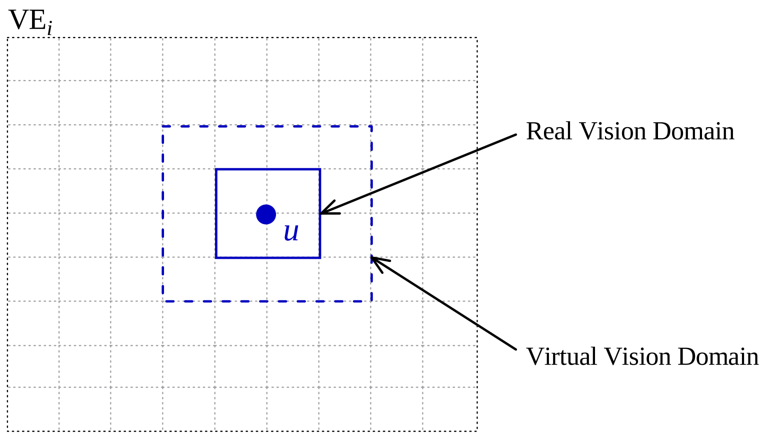


Figure 4.10: Real and Virtual Vision Domains

larger than the real vision domain.[3]

- At time t_1, user v moves to the left by one step. User u is still inside user v's virtual vision domain, causing BS$_j$ to send a syn packet to BS$_i$. As a result, inconsistency is avoided, and user u remains in the consistency state.

For Scenario II inconsistency, consider again the example shown in Figure 4.2. The contents of the VEs at time t_0 are reproduced in Figure 4.12. Again, suppose we use a virtual vision domain which is 2 units larger than the real vision domain. We note that at time t_0, both users u and v are within each other's virtual vision domain.

- At time t_1, user v at BS$_j$ moves down by one step. According to VE$_j$, user u is still inside user v's virtual vision domain. A syn packet is sent from BS$_j$ to

[3]2 units are the smallest possible increment in the size of the vision domain because the size is assumed be an even number.

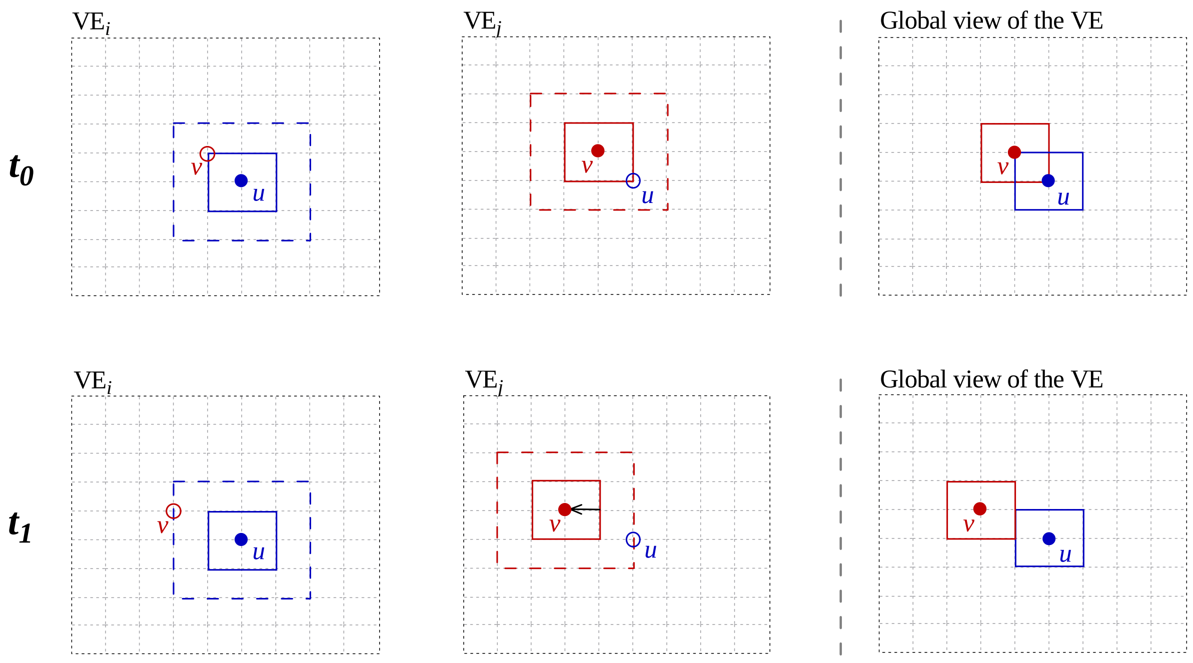


Figure 4.11: How the Virtual Vision Domain Technique Avoids Scenario I Inconsistency

BS$_i$. Upon processing this packet, BS$_i$ has the up-to-date location of user v, and both users remain in the consistent state.

- At time t_2, user u at BS$_i$ moves to the left by one step. User v is within the virtual vision domain of user u, causing BS$_i$ to send a syn packet to BS$_j$. As a result, both users are still in the consistent state; inconsistency is avoided.

The above examples confirm that the area inside the virtual but outside the real vision domain can be viewed as a buffer/margin that helps avoid potential inconsistencies.

Since the virtual vision domain is larger than the real vision domain, additional syn packet traffic is generated. The total rate of packets to each basic system is therefore increased, resulting in more basic systems being required to support the same number of users.

4.4 Evaluation of the Virtual Vision Domain Technique

In this section, we evaluate our proposed virtual vision domain technique by means of simulation. We first consider the time dependent behavior of the mean number of users in the inconsistent state and then study the steady state behavior, focusing on the effectiveness of our virtual vision domain technique and the issue of scalability.

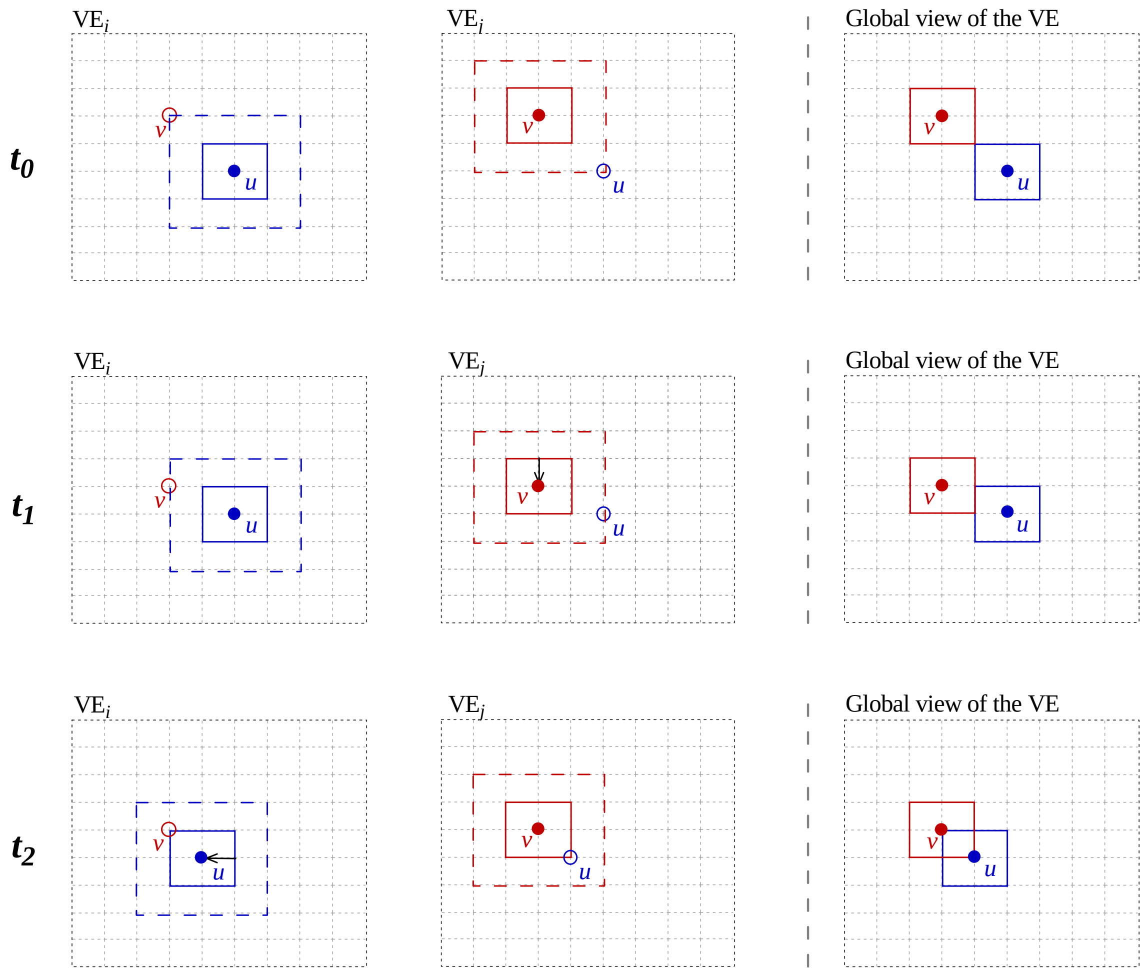


Figure 4.12: How the Virtual Vision Domain Technique Avoids Scenario II Inconsistency

4.4.1 Time Dependent Behavior

Consider again the input parameters shown in Table 4.1. Figure 4.6 has shown the results for $\overline{I}(t)$ as a function of t when the virtual vision domain is not used. These results are for a real vision domain of size 4 (or $D = 4$). Let D' be the size of the virtual vision domain. In our first simulation experiment, we consider two values for D': $D' = D + 2$ and $D + 4$. For each of these values, we obtain the minimum number of basic systems required K_{min} (see Figure 3.10). Note that K_{min} is determined based on D' rather than D because, as mentioned in Section 4.3, a basic system uses the virtual vision domain to determine if a local user's update (in the form of a syn packet) is to be transmitted to another basic system. Using the parameter values in Table 4.1, K_{min} is therefore equal to 6 and 8 for $D' = D + 2$ and $D + 4$, respectively.

The results for $\overline{I}(t)$ are plotted in Figure 4.13. We also include the plot in Figure 4.6 that corresponds to the case where the virtual vision domain is not used. We observe that steady state also exists with the virtual vision domain technique. Furthermore, the use of the virtual vision domain results in a significant reduction in $\overline{I}$, the mean number of users in the inconsistent state (at steady state). Specifically, when $D' = D + 2$ (or $D + 4$), $\overline{I}$ is reduced from 397 to 126 (or 67). In general, the larger the virtual vision domain, the greater is the reduction.

We also consider the case of $y = 0.9$ (instead of $y = 0.8$). Simulation results for $\overline{I}(t)$ for $D' = D + 2$ and $D + 4$ are obtained. For these two values of D', the corresponding values of K_{min}, as determined from Figure 3.10, are 4 and 6, respectively. The results for $\overline{I}(t)$ are shown in Figure 4.14. We again include the

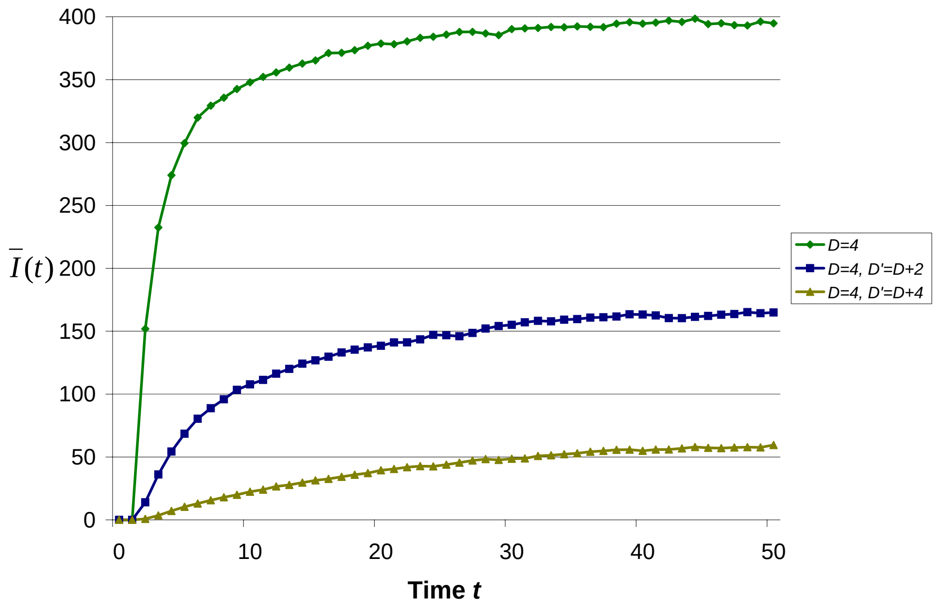


Figure 4.13: Mean Number of Users in the Inconsistent State $\overline{I}(t)$ with Virtual Vision Domain for $E = 100,\ N = 1000,\ D = 4$ and $y = 0.8$

plot in Figure 4.7 that corresponds to the case where the virtual vision domain is not used. We observe that the use of the virtual vision domain again leads to a reduction in $\overline{I}$; this time from 285 to 81 for $D' = D + 2$, and from 285 to 34 for $D' = D + 4$.

The results presented in Figures 4.13 and 4.14 confirm the ability of the virtual vision domain technique to improve consistency. The use of it, however, increases the number of basic systems required K_{min}. In the next section, we will present a detailed analysis of the steady state results to gain insights into the effectiveness of the virtual vision domain technique and the issue of scalability.

4.4.2 Steady State Results

Our evaluation is based on the fraction of users in the inconsistent state, denoted by I_{frac}, and the minimum number of basic systems required K_{min}. Note that $I_{frac} = \overline{I}/N$. The use of I_{frac} instead of $\overline{I}$ facilitates the discussion of our results for different values of N.

The input parameters to our evaluation are summarized in Table 4.2. The parameter values are consistent with those used in Section 3.4.2. We note from the results in Figures 3.10 and 3.11 that one basic system is sufficient to support $N \leq 700$ users, and no inconsistency will occur in the case of one basic system. We therefore select N to be 800 and 1000.

To obtain steady state results for I_{frac}, we choose the length of simulation runs and number of replications so that the width of the 95% confidence intervals is less than 5% of the sample means. We use, as our result, the sample means of I_{frac}

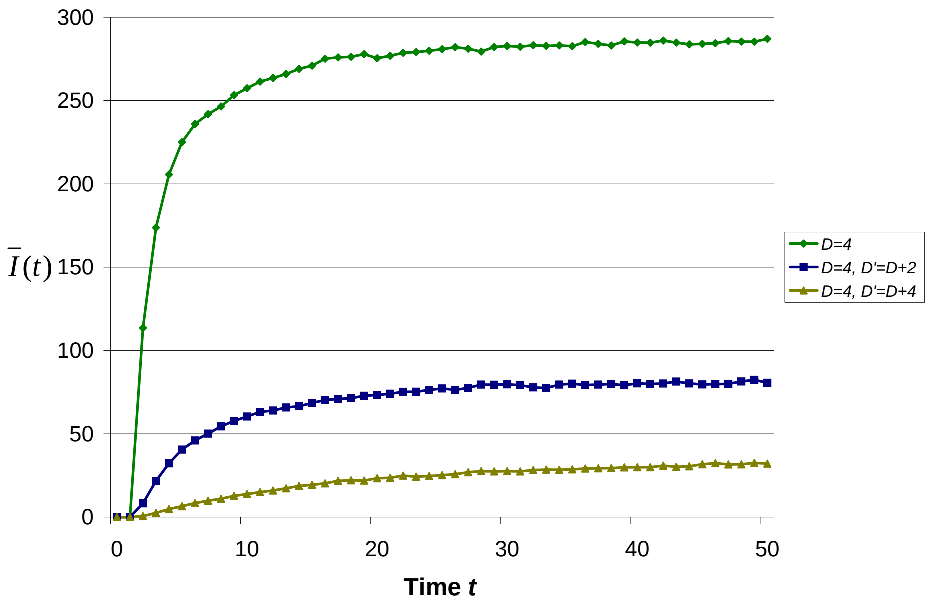


Figure 4.14: Mean Number of Users in the Inconsistent State $\overline{I}(t)$ with Virtual Vision Domain for $E = 100$, $N = 1000$, $D = 4$ and $y = 0.9$

N	Number of users	800, 1000
E	Size of the VE	100, 150
D	Size of the real vision domain	$2, 4, 6$
D'	Size of the virtual vision domain	$D + 2$, $D + 4$
y	Traffic intensity parameter	0.8, 0.9
μ_i	Processing capacity of BS$_i$, same for all i	850
$q_{a,b;c,d}$	User movement probability	uniformly distributed

Table 4.2: Parameter Values Used in Our Simulation Study

over all the replications. In addition, to avoid any potentially negative effect of the initial transient period (i.e., the time before steady state is reached), initial data deletion is performed.

As to K_{min}, its values are determined from Figures 3.10 and 3.11, using the size of the virtual vision domain D'.

The results for I_{frac} for different combinations of N and E are shown in Tables 4.3 – 4.6. We observe that the virtual vision domain technique significantly reduces the fraction of users who are in the inconsistent state. For instance, with $D' = D + 2$ (a virtual vision domain 2 units larger than the real vision domain), I_{frac} is reduced by as much as 92%. With $D' = D + 4$, the largest percentage of reduction in I_{frac} observed is 99%. These results are consistent with our comments in Section 4.3 that the use of the virtual vision domain provides a buffer/margin to avoid potential inconsistencies. Our results also show that a larger virtual vision domain (relative to the real vision domain) leads to a greater percentage of reduction in I_{frac}. When the size of the virtual vision domain D' approaches "infinity" (meaning that the virtual vision domain always covers the entire VE), I_{frac} would drop to zero. This further supports our observation that consistency improves with the size of the virtual vision domain.

The above results illustrate how the virtual vision domain technique can help improve consistency. However, the improved performance comes with a cost of increased resource requirements. As mentioned earlier, having a virtual vision domain larger than the real vision domain would generate additional syn packets, thus increasing the total arrival rate to each basic system. Additional processing capacity is required; this may mean more basic systems for the same number of users.

In Tables 4.3 – 4.6, we also show the results for the number of basic systems required K_{min}. We see that as the size of the virtual vision domain D' increases, K_{min} may increase. Consider, for example, the case of $D = 4$ in Table 4.4. At $y = 0.8$, when $D' = D + 2$, K_{min} is increased from 4 to 6; when $D' = D + 4$, K_{min} is further increased to 8. As another example, consider the case of $D = 4$ in Table 4.5. At $y = 0.8$, when $D' = D + 2$, K_{min} remains unchanged. This means that 2 basic systems are still sufficient to handle the increase in the amount of syn packet traffic. However, when $D' = D + 4$, an additional basic system is required. Comparing these two examples, we observe that the increase in K_{min} is dependent on the size of the virtual vision domain as well as the density of avatars. The results in Table 4.4 are for $N = 1000$ and $E = 100$, while those in Table 4.5 are for $N = 800$ and $E = 150$, which is less dense. In general, a lower density of avatars will result in fewer syn packets being generated and hence fewer basic systems required.

Another interesting performance measure is the mean length of time that a user

		$y = 0.8$			$y = 0.9$		
D	D'	K_{min}	I_{frac}	% reduction	K_{min}	I_{frac}	% reduction
2	D	2	0.1794	—	2	0.1794	—
2	$D+2$	2	0.0308	82.83%	2	0.0308	82.83%
2	$D+4$	3	0.0102	94.31%	2	0.0025	98.61%
4	D	2	0.1536	—	2	0.1536	—
4	$D+2$	3	0.0535	65.17%	2	0.0165	89.26%
4	$D+4$	4	0.0152	90.10%	2	0.0010	99.35%
6	D	3	0.2108	—	2	0.0809	—
6	$D+2$	4	0.0637	69.78%	2	0.0061	92.46%
6	$D+4$	5	0.0230	89.09%	3	0.0031	96.17%

Table 4.3: Fraction of Users in the Inconsistent State I_{frac} for $E = 100$ and $N = 800$

		$y = 0.8$			$y = 0.9$		
D	D'	K_{min}	I_{frac}	% reduction	K_{min}	I_{frac}	% reduction
2	D	2	0.1930	—	2	0.1930	—
2	$D+2$	4	0.1094	43.32%	3	0.0669	65.34%
2	$D+4$	6	0.0290	84.97%	4	0.0166	91.40%
4	D	4	0.3970	—	3	0.2852	—
4	$D+2$	6	0.1261	68.24%	4	0.0817	71.35%
4	$D+4$	8	0.0678	82.92%	6	0.0341	88.04%
6	D	6	0.3963	—	4	0.2964	—
6	$D+2$	8	0.2023	48.95%	6	0.1219	58.87%
6	$D+4$	10	0.1095	72.37%	8	0.0540	81.78%

Table 4.4: Fraction of Users in the Inconsistent State I_{frac} for $E = 100$ and $N = 1000$

		$y = 0.8$			$y = 0.9$		
D	D'	K_{min}	I_{frac}	% reduction	K_{min}	I_{frac}	% reduction
2	D	2	0.1592	—	2	0.1592	—
2	$D+2$	2	0.0420	73.62%	2	0.0420	73.62%
2	$D+4$	2	0.0087	94.54%	2	0.0087	94.54%
4	D	2	0.1390	—	2	0.1390	—
4	$D+2$	2	0.0384	72.37%	2	0.0384	72.37%
4	$D+4$	3	0.0224	83.88%	2	0.0062	95.54%
6	D	2	0.1207	—	2	0.1207	—
6	$D+2$	3	0.0744	38.36%	2	0.0264	78.13%
6	$D+4$	3	0.0151	87.49%	2	0.0037	96.93%

Table 4.5: Fraction of Users in the Inconsistent State I_{frac} for $E = 150$ and $N = 800$

		$y = 0.8$			$y = 0.9$		
D	D'	K_{min}	I_{frac}	% reduction	K_{min}	I_{frac}	% reduction
2	D	2	0.1346	—	2	0.1346	—
2	$D+2$	3	0.0824	38.78%	2	0.0404	69.99%
2	$D+4$	3	0.0199	85.22%	3	0.0199	85.22%
4	D	3	0.2814	—	2	0.1646	—
4	$D+2$	3	0.0819	70.90%	3	0.0819	50.24%
4	$D+4$	4	0.0343	87.81%	4	0.0343	79.16%
6	D	3	0.2573	—	3	0.2573	—
6	$D+2$	4	0.1106	57.02%	4	0.1106	57.02%
6	$D+4$	6	0.0686	73.34%	5	0.0443	82.78%

Table 4.6: Fraction of Users in the Inconsistent State I_{frac} for $E = 150$ and $N = 1000$

is in the inconsistent state (denoted by I_{time}). I_{time} indicates the average length of time that a user stays in the inconsistent state before he returns to the consistent state. Simulation results for I_{time}, together with the minimum number of basic systems required K_{min}, are shown in Tables 4.7 – 4.10. We observe improvements in I_{time} when the virtual vision domain is used. The percentage of improvement, however, is not as significant as that in I_{frac}. We also observe that an increase in the size of the virtual vision domain D' generally yields a good reduction in I_{time}. Finally, we note that the observed values of I_{time} range from 1.0 to 2.8 (approximately). Our time unit is the mean time between updates submitted by the same user. This means that a user in the inconsistent state will usually return to the consistent state in less than three moves by some other user, and less than two in many cases.

Note that a user may be in the inconsistent state for an extended period of time. Consider the scenario shown in Figure 4.15. Users u and v are assigned to BS$_i$ and BS$_j$, respectively. At time t_0, the two users are in the consistent state and located at the opposite corners of the VE. Then, at some arbitrary time t, they move to within each other's vision domain (according to the global view). However, with respect to VE$_i$ and VE$_j$, the two users are not within each other's vision domain; they thus have entered the inconsistent state. Now, suppose they move within each other's vision domain for a long time. They will then be in the inconsistent state for a long period of time.

For the above scenario, one approach to improve consistency is to have BS$_j$ send a state update of user v to BS$_i$ if BS$_j$ has not sent any update of user v to BS$_i$ for a

		$y = 0.8$		$y = 0.9$	
D	D'	K_{min}	I_{time}	K_{min}	I_{time}
2	D	2	2.0922	2	2.0922
2	$D+2$	2	1.2919	2	1.2919
2	$D+4$	3	1.2584	2	1.0091
4	D	2	1.8253	2	1.8253
4	$D+2$	3	1.5040	2	1.1939
4	$D+4$	4	1.4572	2	0.9901
6	D	3	2.0466	2	1.5396
6	$D+2$	4	1.6279	2	1.0945
6	$D+4$	5	1.6109	3	1.2049

Table 4.7: Mean Length of Time that a User is in the Inconsistent State I_{time} for $E = 100$ and $N = 800$

		$y = 0.8$		$y = 0.9$	
D	D'	K_{min}	I_{time}	K_{min}	I_{time}
2	D	2	1.9367	2	1.9367
2	$D+2$	4	1.7184	3	1.4726
2	$D+4$	6	1.5169	4	1.3143
4	D	4	2.1973	3	2.7986
4	$D+2$	6	1.8999	4	1.6136
4	$D+4$	8	2.1343	6	1.7132
6	D	6	2.5104	4	2.8236
6	$D+2$	8	2.3622	6	1.933
6	$D+4$	10	2.4637	8	1.9425

Table 4.8: Mean Length of Time that a User is in the Inconsistent State I_{time} for $E = 100$ and $N = 1000$

		$y = 0.8$		$y = 0.9$	
D	D'	K_{min}	I_{time}	K_{min}	I_{time}
2	D	2	2.5167	2	2.5167
2	$D+2$	2	1.7757	2	1.7757
2	$D+4$	2	1.4837	2	1.4837
4	D	2	2.4896	2	2.4896
4	$D+2$	2	1.7478	2	1.7478
4	$D+4$	3	1.9698	2	1.4887
6	D	2	2.2335	2	2.2335
6	$D+2$	3	2.1157	2	1.5959
6	$D+4$	3	1.7637	2	1.4302

Table 4.9: Mean Length of Time that a User is in the Inconsistent State I_{time} for $E = 150$ and $N = 800$

		$y = 0.8$		$y = 0.9$	
D	D'	K_{min}	I_{time}	K_{min}	I_{time}
2	D	2	2.3286	2	2.3286
2	$D+2$	3	2.0105	2	1.6246
2	$D+4$	3	1.6495	3	1.6495
4	D	3	3.0368	2	2.3217
4	$D+2$	3	2.0434	3	2.0434
4	$D+4$	4	2.0309	4	2.0924
6	D	3	2.6904	3	2.6904
6	$D+2$	4	2.2693	4	2.2693
6	$D+4$	6	2.6828	5	2.2736

Table 4.10: Mean Length of Time that a User is in the Inconsistent State I_{time} for $E = 150$ and $N = 1000$

certain period of time. This approach would keep the length of time that a user is in the inconsistent state under a given threshold. However, it may also result in the distribution of unnecessary state updates among the basic systems. Investigation of this approach is out of the scope of this thesis.

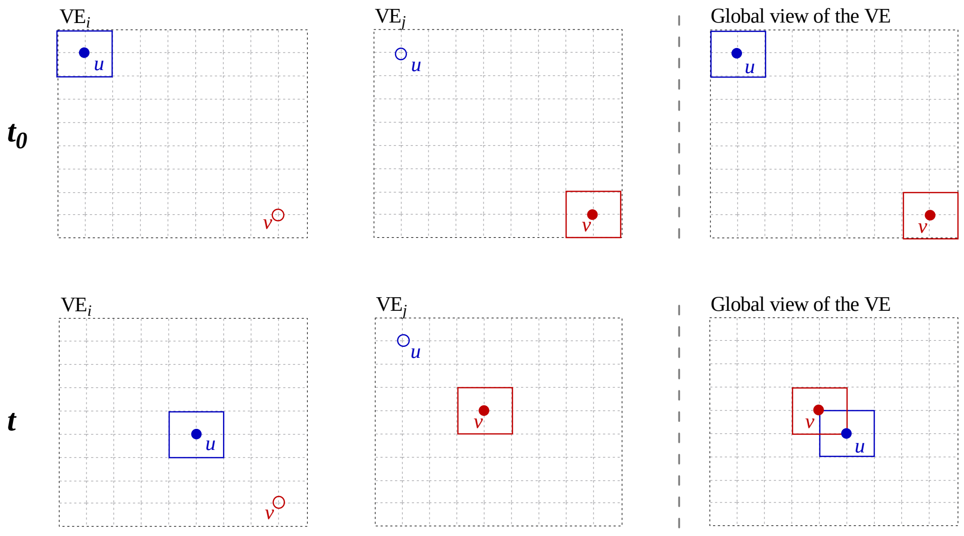


Figure 4.15: An Example

4.5 Concluding Remarks

In this chapter, we investigated the consistency issue in the two-level architecture. We showed that consistency can be restored as a result of user movement. This property is very useful. For example, in a typical DVE, users tend to gather together in a small area to interact with one another. Inconsistency may occur because of the two scenarios described in this chapter, e.g., inconsistency due to a user from outside of the small area entering the area. However, consistency can be restored as a result of user movement, without employing any explicit measures by the basic systems.

Furthermore, we proposed the virtual vision domain technique, as an alternative to periodic global synchronization, to provide weak consistency. Our simulation study showed that this technique is effective in reducing the fraction of users in the inconsistent state, but at the cost of a potential increase in the number of basic systems required.

Another technique that may help improve consistency is to distribute a state update resulting from a user's move to all other users within the vision domains based on the user's old and new locations. With this technique, Scenario I inconsistency is avoided completely because a user is always notified when someone moves out of his vision domain. Note that this technique has the effect of extending the vision domain as far as the distribution of updates is concerned. It can therefore be considered as a special case of our virtual vision domain technique.

Chapter 5

Server Delay

As mentioned in Chapter 1, an important performance measure for a DVE system is the update delay, which is defined as the elapsed time from when a user submits an update to when this update is received by an affected user. Generally, the update delay is composed of two main components: server delay and network delay. The server delay includes the time required to execute the DVE-specific logic on an update received, apply changes to the database, prepare and transmit the resulting update packets, and any queueing delay at a basic system. The network delay, on the other hand, is determined by the queueing delay and processing time at the routers, and packet transmission time and signal propagation delay along the path traversed by a packet. For a DVE system, it is important that the update delay does not exceed some given maximum (e.g., 100 ms); therefore, we are interested in the update delay distribution. Such a distribution will allow us to determine the probability that the update delay is larger than some given value.

Let T, S, and L be the update delay, server delay, and network delay, respectively. We can write:

$$T = S + L.$$

Assuming that S and L are independent of each other, the distribution of T is given by considering the convolution of the probability density functions of S and L. Such a convolution tends to yield rather complex results because analytic results for the distributions of S and L are not simple functions of the input parameters. In this thesis, we analyze the distributions of S and L separately to reduce complexity. We also note that factors affecting the network delay, such as network topology, do not seem to have any major impact on the server delay. We therefore believe that valuable insights into the update delay can be obtained by considering S and L separately. Our analysis will provide answers for the probability that server delay will exceed some given value, and similar answers for the network delay.

In this chapter, we focus on the server delay of our two-level hierarchical architecture; the network delay will be investigated in the next chapter. We first develop a performance model for a basic system and obtain analytic results for its response time distribution. These results are then used to characterize the server delay. When an affected user is at the same basic system, the server delay is the same as the response time at the basic system. Contrarily, when the affected user is at a remote basic system, the server delay is given by the sum of the response times at the local and remote basic systems. Our approach is to obtain approximate results for the distribution of the sum of these response times.

5.1 Performance Model of a Basic System

We model a basic system by a single service facility with multiple processors, as depicted in Figure 5.1. There is a single queue; an arriving packet (an update or a syn packet) joins the queue, awaiting processing by an available processor. Update or syn packets can be processed by any of the processors. Such an architecture is commonly used because the rate at which packets are processed can often be increased by adding more processors. We use m_i to denote the number of processors at BS$_i$.

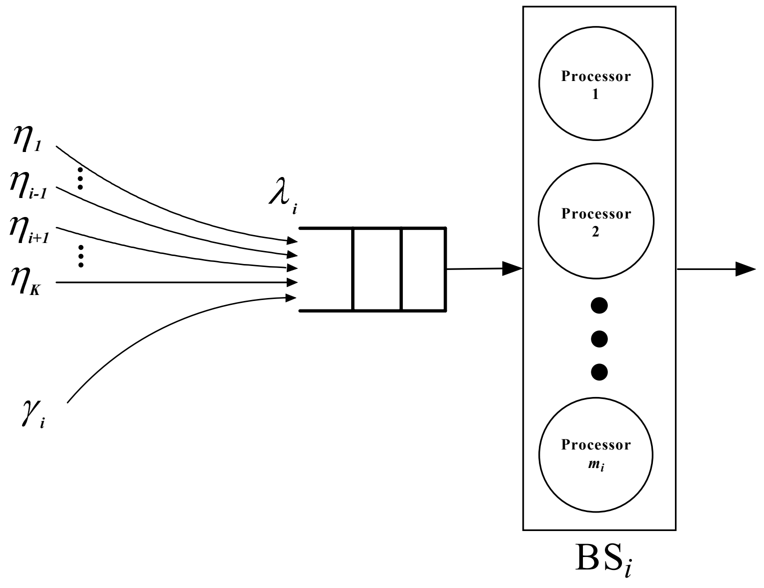


Figure 5.1: A Model of a Basic System

5.1.1 Arrival Process

As shown in Figure 5.1, there are two classes of packets arriving at each basic system, namely update packets from its local users and syn packets from other basic systems. In Chapter 3, we obtained analytic results for γ_i and η_i, the arrival rates of update and syn packets to BS$_i$, respectively. These results are found in Equations 3.2 and 3.7 which are repeated below:

$$\gamma_i = N_i \phi$$

and

$$\eta_i = \sum_{k=1,k\neq i}^{K} \eta_{k,i}$$

where $\eta_{k,i} = g_{k,i} N_k \phi = g_{k,i}\gamma_k$ (see Equation 3.6).

For our model of BS$_i$, we assume that the arrival process of update packets is Poisson with rate γ_i. This is consistent with our earlier assumptions in Section 3.2 that the time between successive moves made by a user is exponentially distributed, and the actions of the N_i users at BS$_i$ are independent of each other.

We assume that the arrival process of syn packets to BS$_i$ is also Poisson, but with rate η_i. This assumption can be justified as follows. Consider, for example, syn packets from BS$_k$. We note that the arrival process of update packets to BS$_k$ is Poisson with rate γ_k. For each of these packets, the probability that a syn packet is sent to BS$_i$ is given by $g_{k,i}$ in Equation 3.5. It follows that the departure process of syn packets from BS$_k$ (destined to BS$_i$) is also Poisson, but with rate $\eta_{k,i}$. These syn packets will experience some delay in the network before reaching BS$_i$. We have two scenarios for this delay.

Scenario 1. BS_i and BS_k are co-located—we assume that the network delay is minimal and can be neglected.

Scenario 2. BS_i and BS_k are not co-located—we assume that the delay in the network is constant and is given by the sum of signal propagation delay and packet transmission time from BS_k to BS_i. This assumption will be discussed further in Chapter 6.

Considering these two scenarios together, a syn packet leaving BS_k will either experience a negligible delay or a constant delay before arriving at BS_i. Both scenarios result in an arrival process of syn packets from BS_k to BS_i that is Poisson. Finally, the aggregation of the $K-1$ streams of syn packets from all the other basic systems yields a Poisson process with rate $\sum_{k=1,k\neq i}^{K} \eta_{k,i} = \eta_i$ (as shown in Equation 3.7).

5.1.2 Service Time Distribution

When an update packet is processed, the update may need to be sent to multiple recipients in addition to executing the DVE-specific logic and applying changes to the database. These recipients include affected users at the local system and/or at any other basic systems. Similarly, when a syn packet is processed, the update may be sent to one or more recipients that are affected by the update, but at the local system only. To simplify our analysis, we assume that the service times of update and syn packets have the same distribution which is exponential with mean $1/\mu$.

We further assume that the rate μ is independent of the number of recipients. This assumption can be justified as follows.

It has been suggested that multicast can be used to disseminate a packet to multiple recipients in a DVE system [10, 26]. With multicast, a basic system is only required to transmit one packet, regardless of the number of recipients. The underlying multicast network will look after the distribution of the packet to the recipients. In case multicast is not supported by the underlying network, unicast has to be used to transmit a separate packet to each recipient. In [64], we reported a kernel-based technique to efficiently unicast a packet to multiple recipients with only minimal additions to the sending operating system interface and implementation. Using our technique, we found that the delay in distributing a packet is largely insensitive to the number of users, when the packet size is small, e.g., less than 100 bytes. Small packet sizes are typically found in DVE systems such as multi-player online games [54, 65].

5.1.3 Other Assumptions

Finally, we assume that packets are serviced in a first-come-first-served (FCFS) manner. Our model of BS$_i$ thus becomes an $M/M/m_i$ model with two classes of packets (update and syn packets) and FCFS discipline. The rate at which packets are processed is $m_i\mu$, and the traffic intensity at BS$_i$ is given by $\rho_i = \lambda_i/m_i\mu$.

5.2 Response Time Distribution

Let X_i be the response time of a packet (an update or a syn packet) at BS$_i$. For the above model, the cumulative distribution function (c.d.f.) of X_i, denoted by $F_{X_i}(t)$, is given by:

$$\begin{aligned} F_{X_i}(t) &= Pr[X_i \leq t] \\ &= \begin{cases} 1 - e^{-\mu t} - \kappa \mu e^{-\mu t(m_i - \sigma)} \left[\frac{1-e^{-\mu t(1-m_i+\sigma)}}{(1-m_i+\sigma)}\right] & \text{for } \sigma \neq m_i - 1 \\ 1 - e^{-\mu t} - \kappa \mu e^{-\mu t} t & \text{for } \sigma = m_i - 1 \end{cases} \end{aligned} \qquad (5.1)$$

where

- $\sigma = \rho_i m_i$
- $\kappa = P(0) \frac{\sigma^{m_i}}{m_i!} \frac{m_i}{(m_i - \sigma)}$, and
- $P(0) = \left(\sum_{n=0}^{m_i - 1} \frac{\sigma^n}{n!} + \frac{m_i \sigma^{m_i}}{m_i!(m_i - \sigma)} \right)^{-1}$.

The response time distribution of each class of packets is the same and is given by Equation 5.1. The derivation of Equation 5.1 can be found in Appendix A.

5.3 Server Delay

In the two-level architecture, a user participating in the DVE submits a stream of update packets to his local basic system. After processing an update packet, the local basic system distributes the update to all affected local users. The local basic system also sends a syn packet to those remote basic systems that have affected users. Consequently, an update submitted by a user may result in several other users receiving the update (at the local or at remote basic systems). Each of these other users will experience a server delay.

In our analysis of overall server delay, the delay experienced by individual affected users will be taken into consideration. We use S_i to denote the overall server delay resulting from the submission of an update packet by a user at BS$_i$, and $F_{S_i}(t)$ to denote its cumulative distribution function (c.d.f.). Our goal is to obtain analytic results for $F_{S_i}(t)$. There are two cases, depending on whether the affected user is at the same local basic system or at a remote basic system.

5.3.1 Case 1 — Local Basic System

For an affected user at the same local basic system, S_i is the same as X_i, the response time at BS$_i$. We thus have:

$$F_{S_i}(t|\text{case } 1) = F_{X_i}(t) \tag{5.2}$$

where $F_{X_i}(t)$ is given by Equation 5.1.

5.3.2 Case 2 — Remote Basic System

Suppose as a result of processing an update submitted by a user at BS$_i$, a syn packet is sent to BS$_j$. This implies that one or more users at BS$_j$ are affected by the update. For each of these users, the server delay is the sum of X_i and X_j, the response times at BS$_i$ and BS$_j$, respectively. We use $X_{i,j}$ to denote this sum. As an approximation, we assume that the packet processing times at BS$_i$ and at BS$_j$ are mutually independent. This means that the server delay distribution can be written as a convolution of the two response times. We thus have:

$$F_{X_{i,j}}(t) = \int_0^t F_{X_i}(t-s) f_{X_j}(s) ds \quad \text{where} \quad f_{X_j}(t) = \frac{dF_{X_j}(t)}{dt} \tag{5.3}$$

and

$$F_{S_i}(t|\text{case } 2, j) = F_{X_{i,j}}(t). \tag{5.4}$$

5.3.3 Overall Server Delay

We now consider $F_{S_i}(t)$, the c.d.f of the overall server delay of an update submitted to BS$_i$. Recall that $\xi_{i,j}(n)$, as defined in Equation 3.4, is the probability that n users at BS$_j$, $j = 1, 2, \ldots, K$, are within the vision domain of any given user at BS$_i$. This is also the probability that when a user at BS$_i$ submits an update packet, n users at BS$_j$ are affected by the update and will receive the update. Note that the arrival rate of update packets submitted to BS$_i$ is given by γ_i. For case 1, let $C_{i,i}$ be the rate at which updates are received by affected users at the same local system. $C_{i,i}$ can be written as:

$$C_{i,i} = \gamma_i \sum_{n_i=0}^{N_i-1} \xi_{i,i}(n_i) n_i.$$

Similarly, for case 2, the rate at which updates are received by affected users at a remote basic system BS$_j$, $j \neq i$, is given by:

$$C_{i,j} = \gamma_i \sum_{n_j=0}^{N_j} \xi_{i,j}(n_j) n_j.$$

Combining the two cases, the c.d.f. of the overall server delay S_i is:

$$F_{S_i}(t) = \frac{1}{C_i} \left[C_{i,i} F_{S_i}(t|\text{case } 1) + \sum_{j=1, j\neq i}^{K} C_{i,j} F_{S_i}(t|\text{case } 2, j) \right] \tag{5.5}$$

where $C_i = \sum_{j=1}^{K} C_{i,j}$. Substituting Equations 5.2 and 5.4 into Equation 5.5, we get:

$$F_{S_i}(t) = \frac{1}{C_i} \left[C_{i,i} F_{X_i}(t) + \sum_{j=1, j\neq i}^{K} C_{i,j} F_{X_{i,j}}(t) \right]. \tag{5.6}$$

Finally, we obtain the distribution of server delay over all the basic systems. It is given by:

$$F_S(t) = \frac{1}{C} \sum_{i=1}^{K} C_i F_{S_i}(t) \tag{5.7}$$

where $C = \sum_{i=1}^{K} C_i$.

5.3.4 Evaluation of Accuracy

As mentioned previously, the results for $F_{X_{i,j}}(t)$ in Equation 5.3 are an approximation. This is due to the assumption that the packet processing times at BS$_i$ and at BS$_j$ are mutually independent. In this section, we evaluate the accuracy of our approximation method.

Our approach is to compare the results for $F_{X_{i,j}}(t)$, as computed from Equation 5.3, with those obtained from simulation. Our comparison is based on the percentile of $F_{X_{i,j}}(t)$. Specifically, we define x_p as the p-th percentile of $F_{X_{i,j}}(t)$. x_p is obtained by solving the following equation:

$$F_{X_{i,j}}(x_p) = p\%.$$

The parameter values used in our experiments are summarized in Table 5.1. The number of basic systems required K_{min} is determined using the results in Section 3.4.2. We thus obtain $K_{min} = 4$ when $y = 0.8$, and $K_{min} = 3$ when $y = 0.9$. Each basic system is then assigned N/K_{min} users. The service rate μ of each processor is chosen to be 850, 425, 170, and 85 for $m_i = 1, 2, 5$, and 10, respectively. These values are selected so that the traffic intensities for the various values of m_i remain unchanged. This would allow the evaluation of accuracy for a range of values for the number of processors while keeping the total processing capacity of the basic system the same. Since different service rates are used for different values of m_i, we present results for the p-th percentile relative to the mean service time, namely that

$$\hat{x}_p = \frac{x_p}{\text{mean service time}}$$

where the mean service time is given by $1/\mu$.

We first consider the scenario where $y = 0.8$. In Figure 5.2, we plot the approximate and simulation results for $\hat{x}_p$ for different values of m_i. We observe that

N	Total number of users	1000
E	Size of the VE	100
D	Size of the vision domain	4
μ	Service rate of each processor	850, 425, 170, 85
m_i	Number of processors at BS$_i$, same for all i	1, 2, 5, 10
y	Traffic intensity parameter	0.8, 0.9
$q_{a,b;c,d}$	User movement probability	uniformly distributed

Table 5.1: Parameter Values Used in the Simulation Experiments

the approximate and simulation results are consistent with each other. We further calculate the percentages of error as follows:

$$\frac{\mid \text{simulation result for } \hat{x}_p \text{ - approximate result for } \hat{x}_p \mid}{\text{simulation result for } \hat{x}_p} * 100\%.$$

Table 5.2 shows the percentages of error. The errors shown are generally less than 5%, confirming that our approximation method yields accurate results.

We next consider the scenario where $y = 0.9$. The approximate and simulation results for $\hat{x}_p$ for different values of m_i are shown in Figure 5.3, and the corresponding error percentages are shown in Table 5.3. We again observe a good agreement between these results.

The above experiments confirm the accuracy of our approximate results for $F_{X_{i,j}}(t)$, as given by Equation 5.3.

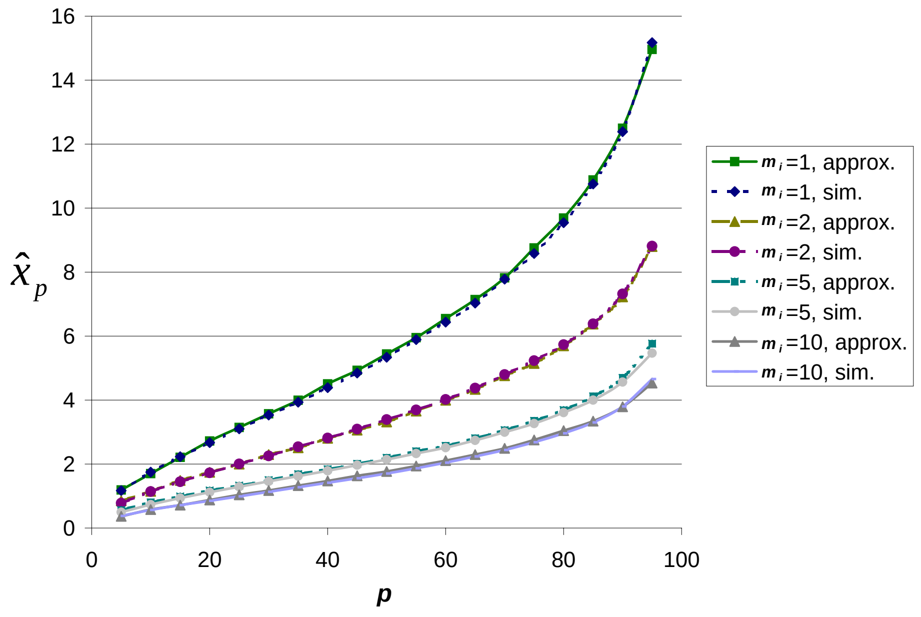


Figure 5.2: Approximate and Simulation Results for $\hat{x}_p$, $y = 0.8$

p	Error Percentages of $\hat{x}_p$			
	$m = 1$	$m = 2$	$m = 5$	$m = 10$
5	1.77	9.26	13.00	3.27
10	2.62	0.43	9.20	2.49
15	0.92	2.67	5.24	0.03
20	1.88	0.89	4.94	2.83
25	1.44	0.37	2.79	4.82
30	1.07	1.74	2.61	3.79
35	1.56	1.31	3.77	4.11
40	2.66	0.30	2.52	4.01
45	1.75	1.07	2.00	4.92
50	1.92	2.28	2.52	3.82
55	1.05	0.99	2.93	3.76
60	1.71	0.59	2.15	3.18
65	1.57	0.92	2.36	2.56
70	0.52	0.80	2.11	2.07
75	2.01	1.73	2.51	2.75
80	1.54	0.68	2.25	2.81
85	1.17	0.14	2.84	1.21
90	0.86	1.29	2.80	0.25
95	1.40	0.17	5.39	2.67

Table 5.2: Percentages of Error between Approximate and Simulation Results for $\hat{x}_p$, $y = 0.8$

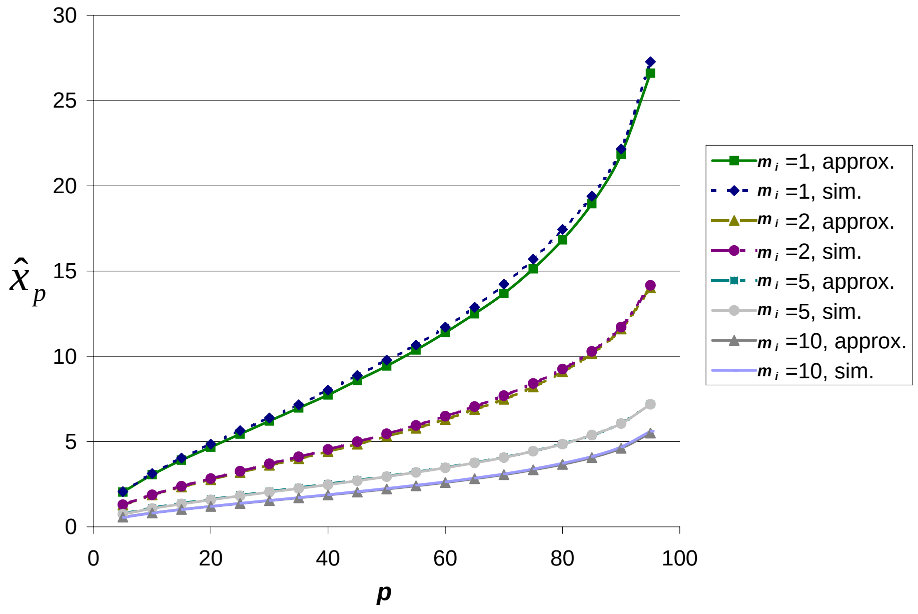


Figure 5.3: Approximate and Simulation Results for $\hat{x}_p$, $y = 0.9$

p	Error Percentages of $\hat{x}_p$			
	$m = 1$	$m = 2$	$m = 5$	$m = 10$
5	1.12	1.82	3.34	3.03
10	2.07	0.34	3.20	0.55
15	3.07	1.85	2.80	0.48
20	3.58	2.36	2.05	0.03
25	3.54	2.23	2.19	0.77
30	2.80	2.30	1.90	0.65
35	2.63	2.81	1.97	0.77
40	3.48	2.66	1.73	0.91
45	3.29	2.95	1.39	1.44
50	3.51	2.69	1.01	1.42
55	2.70	2.81	0.55	1.44
60	2.75	3.04	0.71	1.42
65	2.94	2.38	0.62	1.28
70	3.85	2.79	0.47	1.46
75	3.62	2.43	0.12	1.49
80	3.49	1.68	0.24	1.81
85	2.26	1.31	0.18	1.54
90	1.41	0.92	0.19	1.71
95	2.45	0.97	0.02	1.86

Table 5.3: Percentages of Error between Approximate and Simulation Results for $\hat{x}_p$, $y = 0.9$

5.4 Numerical Examples

In this section, we present numerical examples to gain insights into the server delay. The distribution of server delay over all the basic systems is given by Equation 5.7. The performance measure of interest is the 95th-percentile of the overall server delay, denoted by s_{95}. It is obtained by solving the following equation:

$$F_S(s_{95}) = 95\%.$$

In order to obtain realistic values of the service time for our numerical examples, we conducted an experiment on an existing DVE server, called RockyMud [66], and found that the mean service time to process an update packet is around 0.33 ms. (See Appendix B for details.) We therefore assume that the service rate of each processor μ is 3000 packets per second. This rate is the same for all basic systems. The input parameters used in our numerical examples are listed in Table 5.4.

N	Total number of users	3000, 6000
E	Size of the VE	150
D	Size of the vision domain	2, 4, 6, 8, 10
μ	Service rate of each processor	3000
m_i	Number of processors at BS$_i$, same for all i	1, 2
y	Traffic intensity parameter	0.8, 0.9
$q_{a,b;c,d}$	User movement probability	uniformly distributed

Table 5.4: Parameter Values Used in the Numerical Examples

Two values for m_i, the number of processors at each basic system, are considered: 1 and 2. N is chosen to be 3000 when $m_i = 1$. When m_i is increased to 2, the overall capacity of each basic system is doubled, and the number of supported users N is increased to 6000. The size of VE considered is 150 (or $E = 150$). This would result in a reasonable density of avatars. Finally, the number of basic systems required K_{min} is determined using the results in Section 3.4.2. Each basic system is assigned an equal number of users, given by N/K_{min}.

We first consider the case of one processor at each basic system ($m_i = 1$). The results for the 95th-percentile of the overall server delay s_{95} are shown in Table 5.5. We also show K_{min}, the number of basic systems required, and ρ_i, the traffic intensity at BS$_i$ (which is the same for all the basic systems). We observe that s_{95} is at most 16 ms for the cases considered. We further observe that a higher ρ_i results in a larger s_{95}. This is as expected because a higher traffic intensity means a higher load at a basic system, leading to a higher server delay.

	$y = 0.8$			$y = 0.9$		
D	K_{min}	ρ_i	s_{95}	K_{min}	ρ_i	s_{95}
2	2	0.72	5.0	2	0.72	5.0
4	3	0.77	6.5	3	0.77	6.5
6	5	0.77	6.8	4	0.84	9.8
8	8	0.76	6.2	5	0.90	14.9
10	11	0.76	6.6	7	0.90	16.0

Table 5.5: Results for the 95th-percentile of the Overall Server Delay s_{95} (in ms), $m_i = 1$

Note that an increase in D yields a higher total arrival rate of packets to each basic system (because of the higher syn packet traffic). This in turn leads to a larger K_{min}, the number of basic systems required. The actual traffic intensity is affected by N and K_{min} and may not increase with D. Hence, the impact of D on the server delay percentile is not easy to characterize.

We next consider the case of 2 processors at each basic system ($m_i = 2$). The corresponding results are shown in Table 5.6. We observe that the behavior of s_{95} is similar to that for the case of $m_i = 1$. Comparing the results in Tables 5.5 and 5.6, we note that the 95th-percentile of the overall server delay s_{95} becomes smaller when m_i is increased from 1 to 2. This is because the service rate of each basic system is doubled when m_i is increased from 1 to 2.

	$y = 0.8$			$y = 0.9$		
D	K_{min}	ρ_i	s_{95}	K_{min}	ρ_i	s_{95}
2	3	0.69	2.8	2	0.85	4.7
4	5	0.78	3.8	4	0.85	5.3
6	9	0.78	4.0	7	0.86	5.6
8	14	0.79	3.7	10	0.88	6.9
10	21	0.78	3.9	14	0.90	8.2

Table 5.6: Results for the 95th-percentile of the Overall Server Delay s_{95} (in ms), $m_i = 2$

The numerical examples presented above show that if the traffic intensity at each basic system is maintained below 0.9, the 95th-percentile of the overall server delay could be well below 20 ms. This is a small fraction of the requirement of 100 ms for a DVE system. This also means that when we consider the network delay, the magnitude of the delay could be as high as 80 ms without violating the 100 ms requirement. An in-depth investigation of network delay will be presented in the next chapter.

5.5 Concluding Remarks

In this chapter, we derived approximate analytic results for the server delay distribution. Comparisons with simulation show that our approximate analysis yields accurate results. Through numerical examples, we showed that if the traffic intensity at each basic system is kept below 0.9, the 95th-percentile of the overall server delay could be well below 20 ms. This result provides valuable insights because the mean service time used in the numerical examples is computed from measurement data on an existing DVE server.

Chapter 6

Network Delay

In this chapter, we investigate the network delay in our two-level hierarchical architecture. In this architecture, the basic systems may be co-located or geographically distributed. For both scenarios, we develop performance models and derive analytic results for the network delay distribution. Numerical examples are then used to characterize the conditions under which geographical distribution of the basic systems will lead to performance advantage. We also propose an effective algorithm for determining the locations of basic systems in a network.

6.1 Co-located Basic Systems

6.1.1 Network Model

In this section, we develop a network model for the scenario of co-located basic systems.

We assume that each user workstation is connected to some network "end-point" which can be a point of presence (POP) to the Internet, the user's Internet Service Provider (ISP), or a DSL/cable modem owned by the user. Multiple users may be connected to the same end-point. All basic systems are assumed to be connected to a single network end-point which can also be a POP, an ISP, or a modem.

The network delay between a user and a basic system is measured between their respective end-points. For convenience, the delay between a user workstation (or a basic system) and its connected end-point is not included in our model. Since the basic systems are co-located, we assume that the network delay for syn packets transmitted among these systems is minimal and is also not included in the model. The last two assumptions imply that there is no need to model the transmission of syn packets for the scenario of co-located basic systems.

The network topology is a general, connected graph. An example topology is illustrated in Figure 6.1. In this figure, an end-point to which user workstations are connected is represented by a solid circle, whereas a shaded circle denotes the end-point to which the basic systems are connected. All intermediate nodes (e.g., routers, switches, and multiplexers) are represented by hollow circles. We assume that the network has sufficient bandwidth so that the queueing delay at the various channels is negligible. Packet processing time at the intermediate nodes is also assumed to be negligible. The above assumptions are reasonable in view of advance in high-speed network technologies. These assumptions also imply that the delay at a communication channel is simply given by the sum of signal propagation delay and packet transmission time.

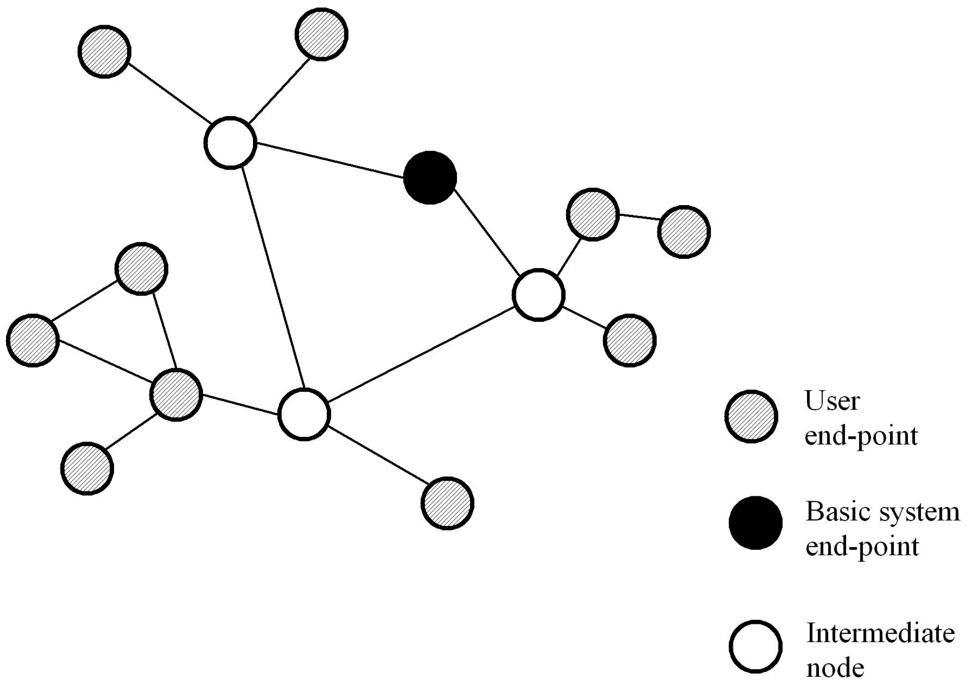


Figure 6.1: Network Model in the Co-located Scenario

We assume that fixed shortest path routing is used. The network delay between a user and a basic system is given by the delay between their respective network end-points, along the shortest path. For simplicity, the network protocols used for the transmission of update and syn packets are not included in our model. Also, we assume that no packet is lost during the transmission.

6.1.2 User Movement Model

When an update packet is processed, an affected user may be located anywhere in the network. In order to determine the network delay experienced by each affected user, we need to model the behavior of individual users explicitly. This can be done by a straightforward extension of the user movement model discussed in Chapter 3 and the relevant measures are defined below:

- $p_{a,b}^{(i,u)} = Pr$[user (i,u) is at location (a,b)]
- $h_{i,u;j,v}(a,b) = Pr$[user (j,v) is within the vision domain of user (i,u), conditioned on user (i,u) being at location (a,b)]

where (i,u) denotes user u at BS$_i$ and, similarly, (j,u) denotes user v at BS$_j$.

$p_{a,b}^{(i,u)}$ and $h_{i,u;j,v}(a,b)$ can readily be obtained by extending the results in Equations 3.1 and 3.3, respectively, to the case where each user is represented explicitly. In particular, $h_{i,u;j,v}(a,b)$ can be written as:

$$h_{i,u;j,v}(a,b) = \sum_{x=x'}^{x^*} \sum_{y=y'}^{y^*} p_{x,y}^{(j,v)} \tag{6.1}$$

where $x' = max\{0, a - \frac{U}{2}\}$, $x^* = min\{A, a + \frac{U}{2}\}$, $y' = max\{0, b - \frac{V}{2}\}$, $y^* = min\{B, b + \frac{V}{2}\}$, U and V are the width and height of the vision domain, and A and B are the width and height of the VE.

Removing the condition on location (a, b), we obtain the following expression for $\psi_{i,u;j,v}$, the probability that user (j, v) is within user (i, u)'s vision domain:

$$\psi_{i,u;j,v} = \sum_{a=0}^{A} \sum_{b=0}^{B} h_{i,u;j,v}(a, b) p_{a,b}^{(i,u)} \tag{6.2}$$

for all (j, v). This probability will be used in the next subsection when we derive analytic results for the network delay distribution.

6.1.3 Network Delay Distribution

We now derive analytic results for the network delay distribution. For convenience, we assume that the delay to transmit an update packet from user (i, u) to BS_i is the same as that to transmit an update packet from BS_i to user (i, u), and we use $z_{i,u}$ to denote this delay. The above assumption is accurate if update packets between user (i, u) and BS_i are transmitted along the same path (but in the opposite direction), these packets have the same size, and each communication channel along the path has the same capacity in both directions. Suppose user (i, u) submits an update. This update would affect all users within his vision domain. We need to consider two cases for these affected users.

Case 1. An affected user is connected to the same local basic system as user (i, u);

Case 2. An affected user is connected to some other basic system, say BS_j.

Case 1 is illustrated in Figure 6.2a. Suppose the affected user is (i, v). The network delay experienced by this user, denoted by $L_{i,u;i,v}$, is given by:

$$L_{i,u;i,v} = z_{i,u} + z_{i,v}. \tag{6.3}$$

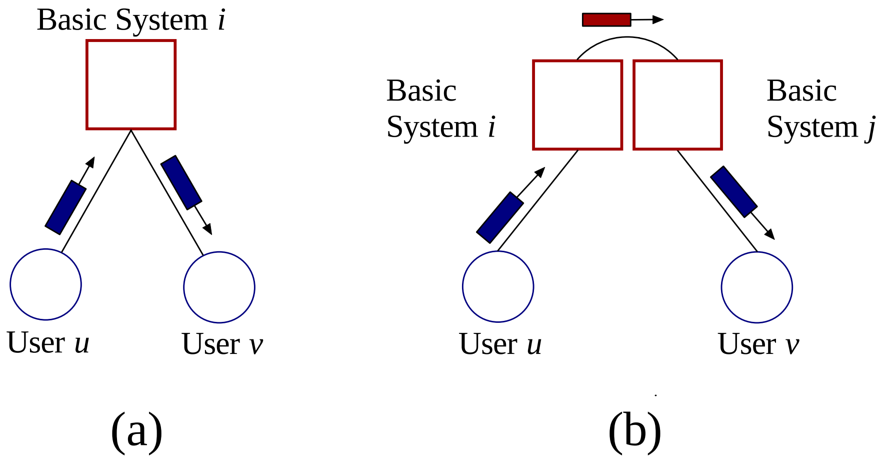


Figure 6.2: Network Delay for the Co-located Scenario

For case 2 (see Figure 6.2b), the network delay from user (i, u) to an affected user (j, v) is given by:

$$L_{i,u;j,v} = z_{i,u} + z_{j,v}. \tag{6.4}$$

Note that based on our previous assumptions, there is no need to include the network delay for transmitting a syn packet from BS$_i$ to BS$_j$.

Let us consider $z_{i,u}$ in detail. Given a network topology and locations of the basic systems and the users, $z_{i,u}$ is a constant and is given by:

$$z_{i,u} = \sum_{(m,n)\in\pi(i,u)} (d_{m,n} + t_{m,n}) \tag{6.5}$$

where $\pi(i,u)$ is the set of channels along the path between user (i,u) and BS$_i$, and $d_{m,n}$ and $t_{m,n}$ are the signal propagation delay and packet transmission time on the channel between nodes m and n, respectively.

Consider again the two cases. Equations 6.3 and 6.4 can be rewritten as follows. For case 1,

$$Pr[L_{i,u;i,v} \leq t] = Pr[z_{i,u} + z_{i,v} \leq t], \tag{6.6}$$

and for case 2,

$$Pr[L_{i,u;j,v} \leq t] = Pr[z_{i,u} + z_{j,v} \leq t]. \tag{6.7}$$

We next combine the results for these two cases to determine the distribution of the network delay L. Recall that ϕ is the rate at which a user submits state updates to his local basic system. For case 1, let $W_{i,u;i,v}$ be the rate at which updates submitted by user (i,u) would result in update packets sent to an affected user (i,v). $W_{i,u;i,v}$ can be written as:

$$W_{i,u;i,v} = \phi\psi_{i,u;i,v} \tag{6.8}$$

where $\psi_{i,u;i,v}$ is the probability that user (i,v) is within the vision domain of user (i,u) (given by Equation 6.2). Similarly, for case 2, we have the following result for the rate at which updates from user (i,u) are sent to an affected user at a remote basic system, say user (j, v):

$$W_{i,u;j,v} = \phi\psi_{i,u;j,v}. \tag{6.9}$$

Summing over all possible combinations of (i, u) and (i, v) for case 1, and all possible combinations of (i, u) and (j, v) for case 2, we have the following expression for the c.d.f. of the network delay distribution:

$$\begin{aligned} F_L(t) &= Pr[L \le t] \\ &= \frac{1}{W}\sum_{i=1}^{K}\left[\sum_{\substack{(i,u),(i,v),\\ u\neq v}} W_{i,u;i,v}Pr[L_{i,u;i,v} \le t] + \sum_{\substack{j=1,\\ j\neq i}}^{K}\sum_{(i,u),(j,v)} W_{i,u;j,v}Pr[L_{i,u;j,v} \le t]\right] \end{aligned} \tag{6.10}$$

where K is the number of basic systems,

$$W = \sum_{i=1}^{K}\left[\sum_{\substack{(i,u),(i,v),\\ u\neq v}} W_{i,u;i,v} + \sum_{\substack{j=1,\\ j\neq i}}^{K}\sum_{(i,u),(j,v)} W_{i,u;j,v}\right],$$

and $Pr[L_{i,u;i,v} \le t]$ and $Pr[L_{i,u;j,v} \le t]$ are given by Equations 6.6 and 6.7, respectively. Note that $W_{i,u;i,v}/W$ can be viewed as the relative frequency that user (i, u) interacts with (i, v) (referred to as "local interaction" because both users are at the same local system). Likewise, $W_{i,u;j,v}/W$ captures the relative frequency of "remote interaction" of user (i, u) with user (j, v).

6.2 Geographically Distributed Basic Systems

We next consider the scenario where basic systems are geographically distributed.

6.2.1 Network Model

The network model for the co-located scenario, as described in Section 6.1, can readily be extended to basic systems that are geographically distributed. Particularly, the basic systems may now be connected to different network end-points. An example network model with two basic systems is shown in Figure 6.3.

For basic systems that are geographically distributed, the delay experienced by syn packets sent between them is modeled explicitly. Syn packets are typically transmitted via a high-speed network. For this network, we make similar assumptions as those described in Section 6.1. These include no queueing delay, negligible packet processing time at the intermediate nodes, and shortest path routing. Consequently, the network delay for syn packets exchanged between two basic systems is simply the sum of the signal propagation delay and data transmission time, measured between their respective network end-points.[1] If two basic systems happen to be connected to the same network end-point, then the network delay between them is not included in our model, similar to our discussion of co-located basic systems in the previous section.

[1]Note that the resulting network delay is deterministic and is consistent with our discussion in Section 5.1.1 regarding the arrival process of syn packets to a remote basic system.

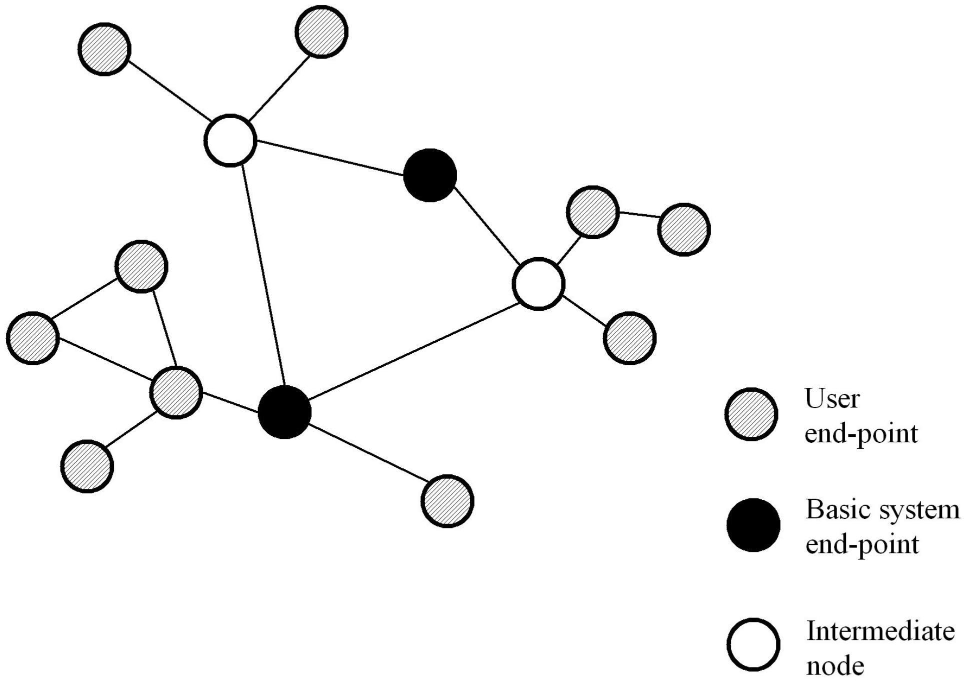


Figure 6.3: Network Model with Two Basic Systems in the Geographically Distributed Scenario

6.2.2 Network Delay Distribution

We now derive the network delay distribution for the geographically distributed scenario. As mentioned in Section 6.1, an update from a user may affect users at the same local basic system (case 1) and/or at some other basic systems (case 2). For case 1, the network delay distribution is the same as that for the co-located scenario (see Figure 6.4a). More specifically, the distribution of network delay from user (i, u) to an affected user (i, v) is given by Equation 6.6. For case 2, the update delay involves an additional component which is the network delay for transmitting a syn packet between the local and the remote basic systems. Suppose user (i, u) submits an update, and this update affects user (j, v) where $j \neq i$. This is illustrated in Figure 6.4b.

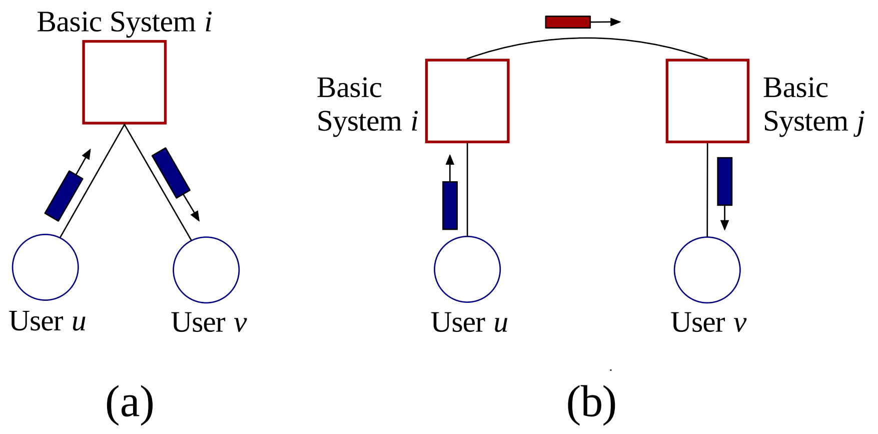


Figure 6.4: Network Delay for the Geographically Distributed Scenario

The network delay between user (i, u) and the affected user (j, v) is given by:

$$L_{i,u;j,v} = z_{i,u} + y_{i,j} + z_{j,v} \tag{6.11}$$

where $y_{i,j}$ is the delay in transmitting a syn packet from BS_i to BS_j. Again, we assume that $y_{i,j}$ and $y_{j,i}$ are the same. The derivation of $y_{i,j}$ is similar to that of $z_{i,u}$ in Equation 6.5 and is given by:

$$y_{i,j} = \sum_{(m,n)\in\pi(i,j)} (d_{m,n} + t_{m,n}) \tag{6.12}$$

where $\pi(i, j)$ is the set of channels along the path between BS_i and BS_j, and $d_{m,n}$ and $t_{m,n}$ are the signal propagation delay and packet transmission time on the channel between nodes m and n, respectively. We thus have

$$Pr[L_{i,u;j,v} \leq t] = Pr[z_{i,u} + y_{i,j} + z_{j,u} \leq t]. \tag{6.13}$$

Summing over all possible combinations of (i, u) and (j, v) for the two cases, we have the following expression for the c.d.f. of the network delay distribution for the geographically distributed scenario:

$$\begin{aligned} F_L(t) &= Pr[L \leq t] \\ &= \frac{1}{W}\sum_{i=1}^{K}\left[\sum_{\substack{(i,u),(i,v),\\ u\neq v}} W_{i,u;i,v} Pr[L_{i,u;i,v} \leq t] + \sum_{\substack{j=1,\\ j\neq i}}^{K} \sum_{(i,u),(j,v)} W_{i,u;j,v} Pr[L_{i,u;j,v} \leq t]\right] \end{aligned} \tag{6.14}$$

where K is the number of basic systems,

$$W = \sum_{i=1}^{K} \left[\sum_{\substack{(i,u),(i,v),\\ u \neq v}} W_{i,u;i,v} + \sum_{\substack{j=1,\\ j \neq i}}^{K} \sum_{(i,u),(j,v)} W_{i,u;j,v} \right],$$

and $Pr[L_{i,u;i,v} \leq t]$ and $Pr[L_{i,u;j,v} \leq t]$ are given by Equations 6.6 and 6.13, respectively.

6.3 Numerical Results

In this section, we present numerical results for the network delay distribution for the co-located and geographically distributed scenarios. The merit of distributing the basic systems geographically is also discussed.

6.3.1 Example Networks

Our results are based on a network consisting of multiple autonomous systems (AS's) [67], each of which contains a collection of nodes. An example is illustrated in Figure 6.5. This network can be viewed as having two levels. At the higher level, the AS nodes form a backbone network. Within each AS, the local nodes form a local network; at least one of these local nodes is connected to the AS node. Users can only be connected to the local nodes, whereas basic systems are connected to the AS nodes.

In our numerical examples, we consider networks with 10 AS nodes. The AS backbone networks have diameters of 7,500 km, 15,000 km, and 30,000 km, repre-

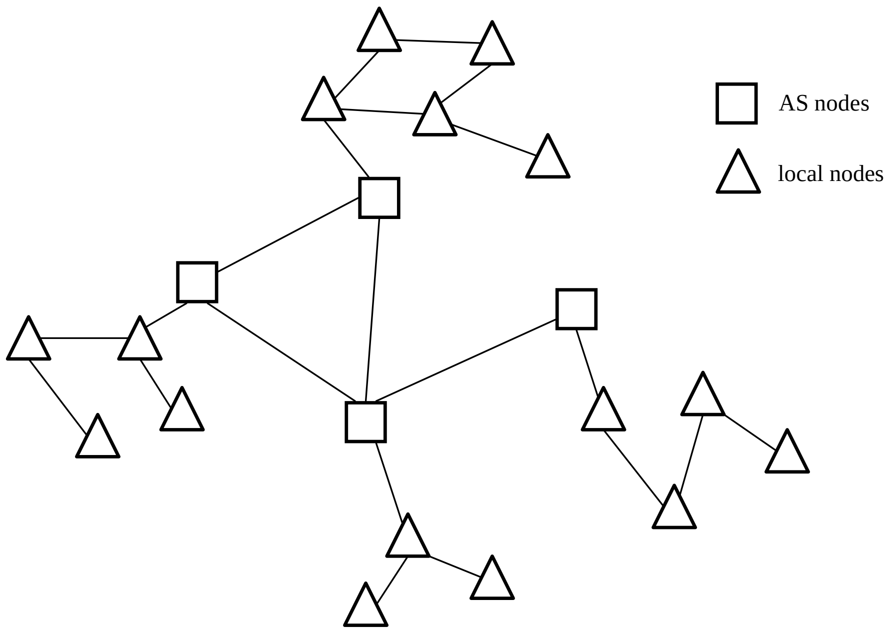


Figure 6.5: An Example Network Model Used in Our Numerical Examples

senting networks of different sizes. We refer to the three networks as networks 1, 2, and 3, respectively. For each local network of a AS, the number of local nodes is 5, and the diameter of the local network is assumed to be 5,000 km. We use a well-known universal Internet topology generator, called BRITE [68], to generate our network topologies. The network topologies are based on the Waxman model [69] with the same parameters as those selected in [70]. The use of the Waxman model is for illustrative purposes only. Using BRITE, the average signal propagation delays between two local nodes in the topologies generated for networks 1, 2 and 3 are approximately 30, 50, and 70 ms, respectively (assuming that propagation delay is characterized by 300 km/ms). We believe that such example networks would provide insights into the performance of different networking and scenarios of basic systems' locations.

For high speed networks and small packet size, the packet transmission time ($t_{m,n}$ in Equations 6.5 and 6.12) is small when compared to the signal propagation delay. For simplicity, the packet transmission time is assumed to be negligible. The network delay is then determined by the signal propagation delay only.

There are N users. The number of users connected to each local node is the same. Since our example networks have 50 local nodes, each local node has $N/50$ connected users. We also have K basic systems, which can be placed at any of the 10 AS nodes. For the co-located scenario, the K basic systems are connected to the same AS. When the basic systems are geographically distributed, each basic system can be connected to any of the AS nodes. Once the basic system locations are chosen, users are assigned to the closest basic systems.

6.3.2 User Model

As discussed in Section 6.1.2, a user, say (i, u), has probability $p_{a,b}^{(i,u)}$ of being at location (a, b) in the VE at steady state. For the case $p_{a,b}^{(i,u)} = p_{a,b}$ for all (i, u), this user model is the same as that defined in Section 3.2.

It is quite intuitive that for a given basic system, if more of its connected users are located in areas within the VE that are not well populated by users at the other basic systems, then the level of interaction between users at different basic systems will be reduced and the rate of syn packets being generated will be smaller. This would tend to favor the idea of locating a basic system close to its connected users. A parameter that indicates the level of interaction between users at different basic systems would be helpful in understanding the conditions under which distributing the basic systems geographically will lead to performance advantage. One such parameter can be defined as follows.

Consider first the case of two basic systems. For BS_i, $i = 1, 2$, all of its local users are assumed to be in the same region (denoted by RE_i). For convenience, we assume that RE_i is square with size $(r + 1)^2$ and each local user is equally likely to be at any location within RE_i. The two regions RE_1 and RE_2 may overlap horizontally and the fraction of overlap is characterized by a parameter Φ, $0 \leq \Phi \leq 1$ (see Figure 6.6). When $\Phi = 0$, the two regions are disjoint. This means that there are no interaction between users at BS_1 and BS_2. As Φ increases, the level of interaction also increases, and the maximum interaction occurs at $\Phi = 1$. The extension of the definition of Φ to three basic systems is shown in Figure 6.7. Again, there are no interaction between users at different basic systems when $\Phi = 0$, and the level of interaction increases with Φ. Similar extensions can be defined for more than three basic systems.

Note that when $\Phi = 1$, the user model is reduced to that defined in Section 3.2, i.e., $p_{a,b}^{(i,u)} = p_{a,b}$ for all (i, u). For $\Phi < 1$, the user model is dependent on the basic system to which a user is connected. We recognize that this assumption may not be realistic in practice. Nevertheless, by varying Φ, we can gain insights into the conditions under which geographical distribution of the basic systems will lead to performance advantage.

6.3.3 Performance Measure

Recall that the network delay distributions $F_L(t)$ for the two scenarios are given by Equations 6.10 and 6.14, respectively. $F_L(L_{max})$, the probability that the network delay is less than or equal to some given L_{max}, is of particular interest. L_{max} can be viewed as an estimate of the largest network delay that would result in realistic interaction. In considering the value of L_{max}, we note that the update delay T is given by $T = S + L$ where S is the server delay and L is the network delay. We also note that for realistic interaction, T should not exceed 100 ms. In our investigation of the server delay in Chapter 5, the results in Section 5.4 indicated that the server delay S could be well below 20 ms while maintaining a traffic intensity of 0.9 or less at each basic system. We thus consider $L_{max} = 80$ and 90 ms in our numerical examples.

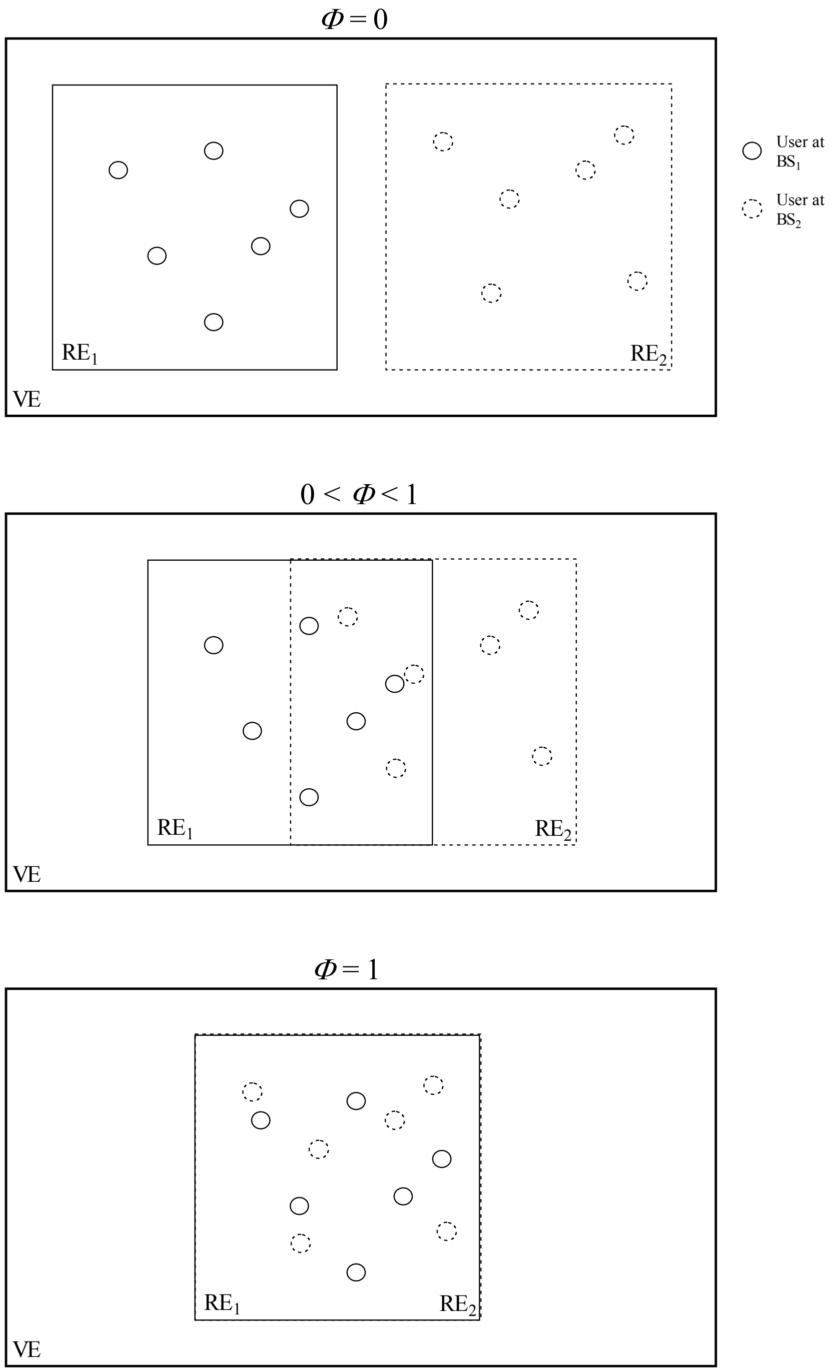


Figure 6.6: Overlap between Regions for $K = 2$

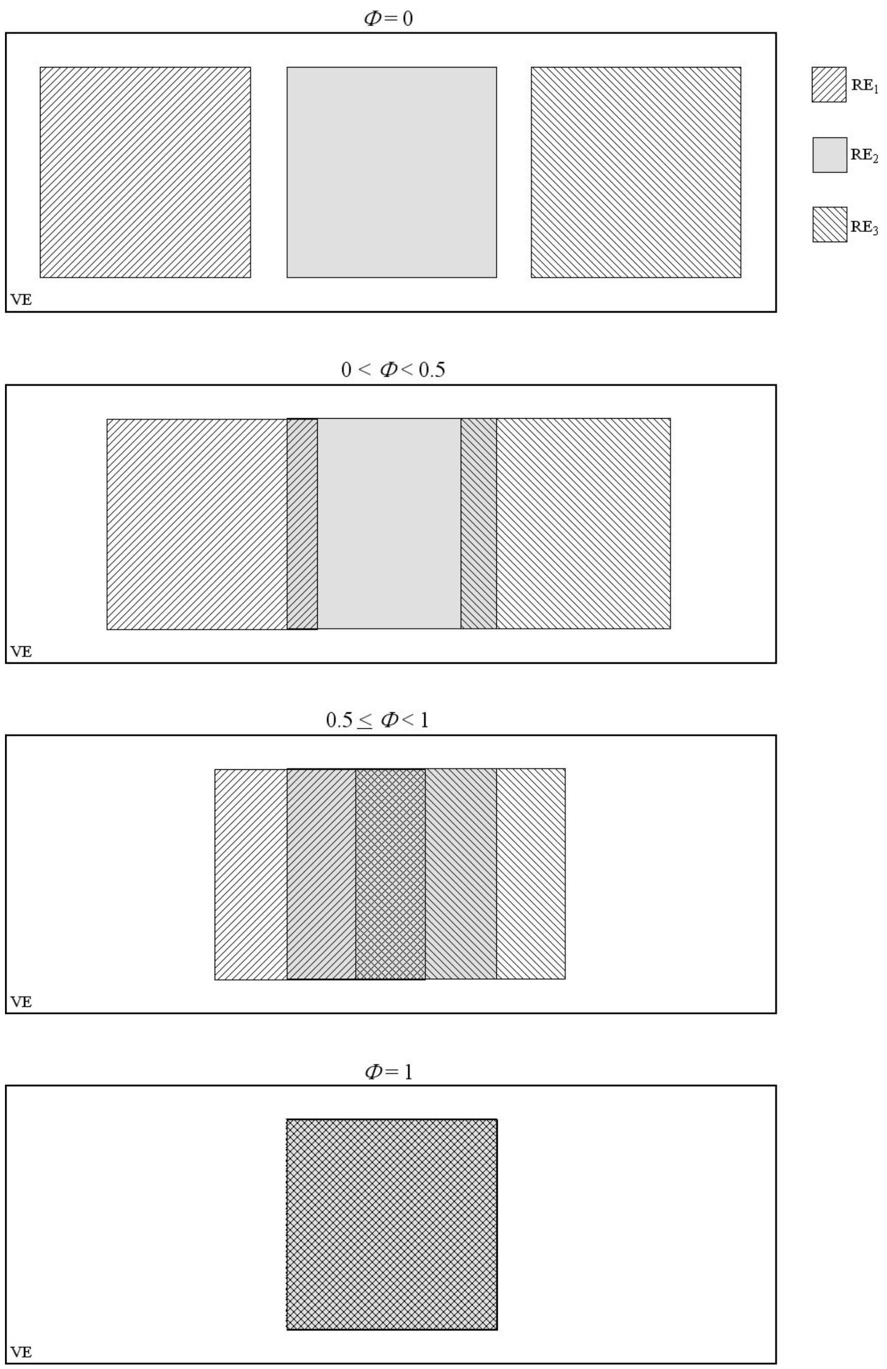


Figure 6.7: Overlap between Regions for $K = 3$

6.3.4 Preliminary Observations

Our initial experiments reveal that $F_L(L_{max})$ is not affected by the number of users N. This can be explained as follows. We have assumed that the number of user end-points is constant. The number of users connected to each end-point therefore increases proportionally with N. Changes in N would not have any impact on the frequency with which the network delay between any pair of user end-points is larger than L_{max}. Therefore, the network delay distribution is not affected.

Our initial experiments also reveal that the value of $F_L(L_{max})$ is not affected by the size of a region (or the value of r). This can be explained as follows. We have assumed that all the regions are of the same size and that users within each region are uniformly distributed. Changes in r would lead to the same percentage of increase in both $\psi_{i,u;i,v}$ and $\psi_{i,u;j,v}$. As a result, the network delay distribution is not affected.

6.3.5 Results and Discussions

As discussed in the last subsection, the network delay distribution is not affected by N and r. We use $N = 1000$ and $r = 10$ in our numerical examples. The size of the vision domain D is set to 2, and a uniform user movement probability distribution is assumed for each user. The number of basic systems considered is $K = 2$, 3 and 4.

The following approach is used to obtain results for our performance measure $F_L(L_{max})$. For each combination of input parameters, 20 networks are generated using the method described in Section 6.3.1. We then obtain the average as well as the best values of $F_L(L_{max})$ over all possible locations of the basic systems for both the co-located and geographically distributed scenarios.

Case 1: $\Phi = 1$

Consider first $\Phi = 1$. This corresponds to the case where $p_{a,b}^{(i,u)} = p_{a,b}$ for all (i, u), the user model defined in Section 3.2. The results for $F_L(L_{max})$ are shown in Tables 6.1 and 6.2 for $L_{max} = 80$ and 90 ms, respectively. We observe that increased network size (e.g., network 3 instead of network 1) has a negative impact on $F_L(L_{max})$ for both the co-located and geographically distributed scenarios. As an example, for network 3, $F_L(L_{max})$ for $L_{max} = 80$ ms is less than 0.40 in the best case and can be as small as 0.13 on average. A similar observation is made for $L_{max} = 90$ ms. The poor performance is a consequence of the challenge posted by geographical distance. When two users are far apart, the signal propagation delay between them may have already exceeded the constraint of 80 or 90 ms. This would certainly have a negative impact on the quality of the interaction.

The results are quite different for network 1, where $F_L(L_{max})$ for $L_{max} = 80$ (or 90) ms is 1.00 in the best case and the average value is at least 0.86 (or 0.89). This is very good performance in terms of meeting the delay constraint. Compared to networks 2 and 3, the performance improvement is mainly due to the fact that the signal propagation delay is well within the 80 (or 90) ms range.

The results in Tables 6.1 and 6.2 clearly show the performance advantage of the geographically distributed scenario over the co-located scenario. There are noticeable improvements in both the average and best values of $F_L(80)$ and $F_L(90)$. Consider, for instance, the average values of $F_L(80)$. The percentage improvement ranges from 6.14% to 7.92% for network 1 and 36.64% to 55.18% for network 3. One should note, however, that the improvement for network 3 is from $F_L(80) = 0.13$ to 0.20. The actual performance is still poor even though there is a 55.18% improvement. This is again due to the large size of network 3.

		average $F_L(80)$			best $F_L(80)$		
network	K	co.	dist.	% improvement	co.	dist.	% improvement
1	2	0.86	0.91	6.14%	0.89	0.99	11.14%
	3	0.86	0.91	6.84%	0.89	0.99	11.13%
	4	0.86	0.92	7.92%	0.89	1.00	11.52%
2	2	0.45	0.49	7.85%	0.64	0.81	27.23%
	3	0.45	0.50	9.73%	0.64	0.84	31.37%
	4	0.45	0.51	13.50%	0.64	0.85	33.42%
3	2	0.13	0.17	36.64%	0.24	0.33	36.89%
	3	0.13	0.18	45.01%	0.24	0.37	56.02%
	4	0.13	0.20	55.18%	0.24	0.39	65.04%

Table 6.1: Results for $F_L(80)$ when $\Phi = 1$

Additionally, we observe that for a given network, when more basic systems are used, a more significant performance improvement of the geographically distributed scenario is observed. This is because with more basic systems, we are able to place them closer to the users, thus reducing the network delay.

The performance advantage of the geographically distributed scenario can be illustrated by the following example. Figures 6.8a and 6.8b show example networks where two users, (i, u) and (j, v), are connected to co-located and geographically distributed basic systems, respectively. The label on each communication channel represents the signal propagation delay on that channel. Note that the signal propagation delay between two nodes is based on their geographical locations in

		average $F_L(90)$			best $F_L(90)$		
network	K	co.	dist.	% improvement	co.	dist.	% improvement
1	2	0.89	0.96	8.62%	0.90	1.00	11.12%
	3	0.89	0.97	9.01%	0.90	1.00	11.12%
	4	0.89	0.97	9.50%	0.90	1.00	11.21%
2	2	0.56	0.59	5.86%	0.72	0.91	26.08%
	3	0.56	0.60	7.17%	0.72	0.93	28.22%
	4	0.56	0.62	9.86%	0.72	0.93	28.99%
3	2	0.17	0.23	30.87%	0.31	0.42	33.48%
	3	0.17	0.24	37.12%	0.31	0.46	47.07%
	4	0.17	0.25	45.69%	0.31	0.49	55.98%

Table 6.2: Results for $F_L(90)$ when $\Phi = 1$

the network. Suppose user (i, u) submits an update and user (j, v) is the affected user. For the co-located scenario, the network delay from user (i, v) to user (j, v) is given by:

$$L = a + b + c + d,$$

while that for the geographically distributed scenario is given by:

$$L = a + e + d.$$

Since $e < b + c$, the geographically distributed scenario yields a smaller network delay.

Case 2: $\Phi < 1$

As mentioned in Section 6.3.2, the parameter Φ reflects the level of interaction between users at different basic systems. Consider the case of 3 basic systems

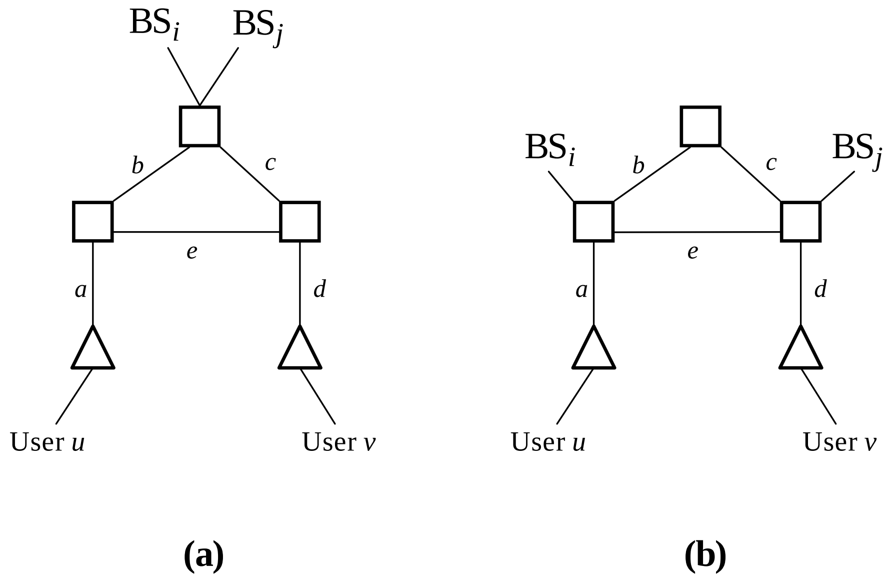


Figure 6.8: Example Networks

($K = 3$). In Figures 6.9, 6.10 and 6.11, we plot $F_L(80)$ as a function of Φ for the three networks considered. We observe that the geographically distributed scenario performs significantly better when Φ is small. This is consistent with our remark that a smaller Φ means less interaction between users at different basic systems and fewer syn packets being exchanged between these systems, thus reducing the overall network delay. We also observe that the larger the network size, the more significant is the performance advantage of the geographically distributed scenario. This is a consequence of placing the basic systems closer to their connected users. Similar observations can also be made for the case of $F_L(90)$ (results not shown).

Finally, we consider the extreme case of $\Phi = 0$, or no interaction between users at different basic systems. This case yields the maximum improvement of the geographically distributed scenario. In Tables 6.3 and 6.4, we show the results for $F_L(L_{max})$ for $L_{max} = 80$ and 90 ms, respectively. As expected, the geographically distributed scenario outperforms the co-located scenario. An interesting observation is that for network 3, the best values for $F_L(80)$ and $F_L(90)$ are 0.77 and 0.85, respectively. This implies that for large networks, the geographically distributed scenario could result in acceptable network delay if most interaction is between users connected to the same basic system.

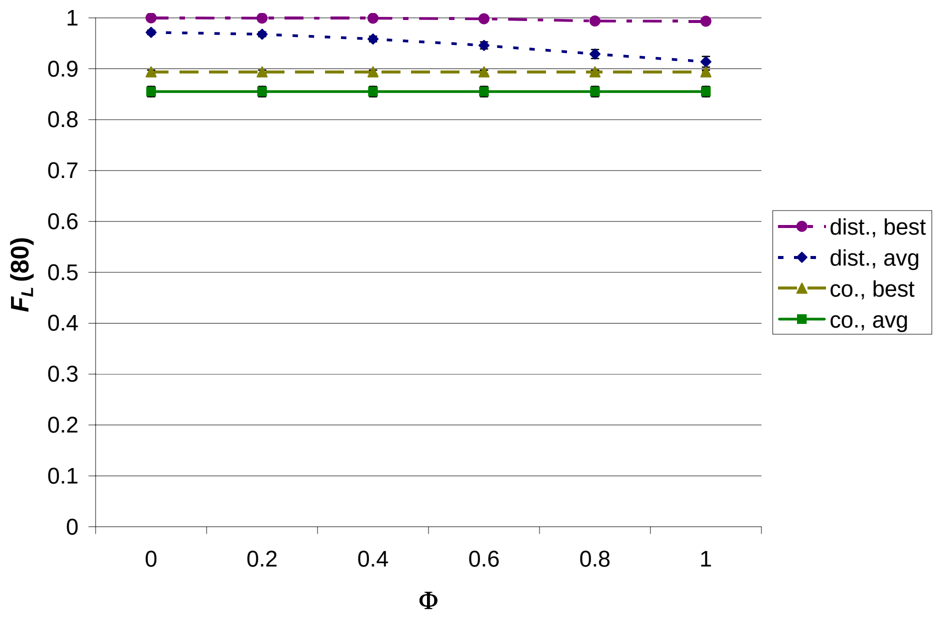


Figure 6.9: $F_L(80)$ vs. Φ for $K = 3$ in Network 1

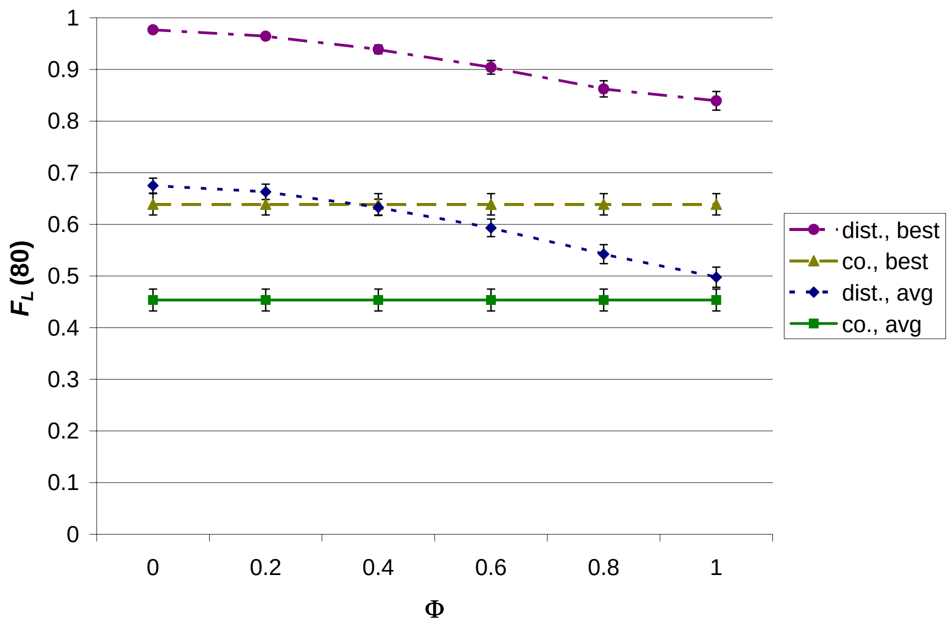


Figure 6.10: $F_L(80)$ vs. Φ for $K = 3$ in Network 2

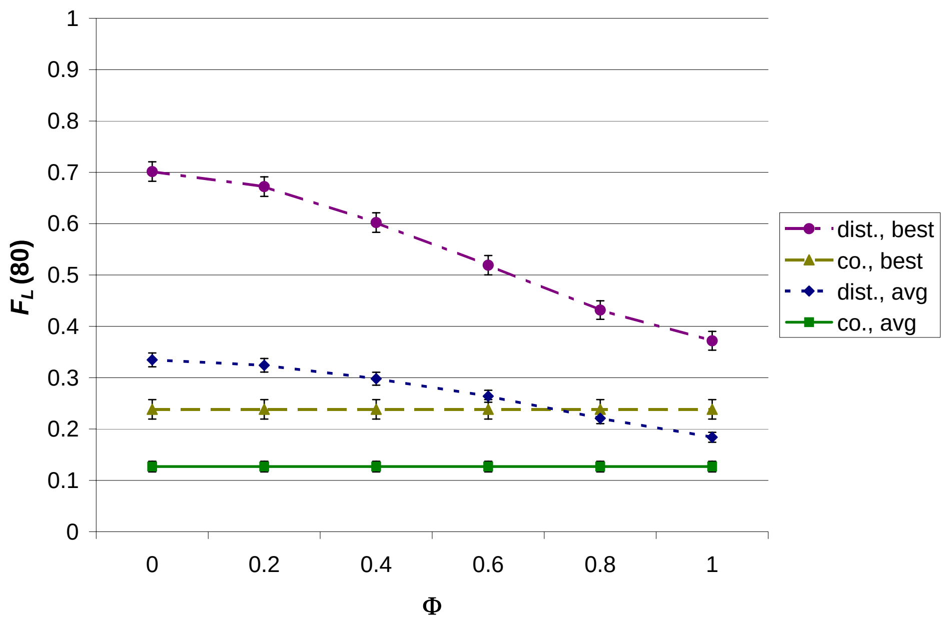


Figure 6.11: $F_L(80)$ vs. Φ for $K = 3$ in Network 3

		average $F_L(80)$			best $F_L(80)$		
network	K	co.	dist.	% improvement	co.	dist.	% improvement
1	2	0.86	0.97	12.91%	0.89	1.00	11.71%
	3	0.86	0.97	13.61%	0.89	1.00	11.87%
	4	0.86	0.98	14.09%	0.89	1.00	11.87%
2	2	0.45	0.64	41.98%	0.64	0.95	48.07%
	3	0.45	0.68	48.85%	0.64	0.98	52.89%
	4	0.45	0.70	53.72%	0.64	0.99	55.22%
3	2	0.13	0.28	119.95%	0.24	0.56	132.85%
	3	0.13	0.33	163.86%	0.24	0.70	194.28%
	4	0.13	0.38	199.10%	0.24	0.77	224.49%

Table 6.3: Results for $F_L(80)$ when $\Phi = 0$

		average $F_L(90)$			best $F_L(90)$		
network	K	co.	dist.	% improvement	co.	dist.	% improvement
1	2	0.89	0.99	11.63%	0.90	1.00	11.26%
	3	0.89	0.99	11.87%	0.90	1.00	11.26%
	4	0.89	0.99	12.00%	0.90	1.00	11.26%
2	2	0.56	0.73	30.92%	0.72	0.99	36.87%
	3	0.56	0.75	34.36%	0.72	0.99	37.49%
	4	0.56	0.77	36.89%	0.72	1.00	38.35%
3	2	0.17	0.35	104.65%	0.31	0.65	106.86%
	3	0.17	0.41	138.14%	0.31	0.79	151.93%
	4	0.17	0.46	164.19%	0.31	0.85	172.75%

Table 6.4: Results for $F_L(90)$ when $\Phi = 0$

6.4 Basic System Placement Algorithm

As discussed in the last section, the geographically distributed scenario has a performance advantage over the co-located scenario. An interesting question is how to obtain the optimal placement for the basic systems in the network. By optimal, we mean the best value for $F_L(L_{max})$.

The best values for $F_L(L_{max})$ presented in Section 6.3 were obtained by exhaustive search (i.e., by enumerating all possible locations). With exhaustive search, the number of possible placements is $\binom{Q-1+K}{K}$ where Q is the number of potential end-points to which basic systems are connected and K is the number of basic systems deployed. For each of these placements, we have to compute $F_L(L_{max})$ using Equation 6.14. Suppose such computation takes a constant time denoted by C. The complexity of exhaustive search is then given by $\Theta\big(\binom{Q-1+K}{K}C\big)$. Consider, for example, $Q = 10$ and $K = 4$; the computational requirement is around $715C$. When Q and K are large, the exhaustive search approach would become very costly. We therefore propose an efficient heuristic algorithm for the placement of basic systems.

The basic idea of our heuristic algorithm is as follows. We first place all the K basic systems at the same location that would result in a maximum value for $F_L(L_{max})$. Note that this initial step ensures that the placement obtained by our algorithm is as good as the co-located scenario. Then, we move $K-1$ basic systems one by one from this location to a new location. At each move, the basic system under consideration is placed at a location such that $F_L(L_{max})$ is maximized. Once this basic system is placed, its location can no longer be changed.

As to efficiency, our heuristic algorithm consists of K moves. In the first move, it takes Q steps to find the initial optimal location. For the rest of the moves, each requires $Q - 1$ steps to determine the best location of the basic system involved. Thus, the complexity of our algorithm is $\Theta(Q \times K \times C)$. Consider again the example of $Q = 10$ and $K = 4$. Using our algorithm, the computational requirement is around $40C$, which is a significant improvement over the exhaustive search approach.

We now compare the performance of our heuristic algorithm against the optimal results obtained by exhaustive search. Our evaluation is based on the same numerical examples discussed in Section 6.3. For illustrative purposes, we present only the results for $F_L(80)$, the probability that the network delay is less than or equal to 80 ms. In Tables 6.5, 6.6 and 6.7, we show the results for $F_L(80)$ obtained by our heuristic algorithm and by exhaustive search for different values of Φ for networks 1, 2 and 3, respectively. We observe that our heuristic algorithm is able to yield results very close to those obtained by exhaustive search.

	$K = 2$		$K = 3$		$K = 4$	
Φ	opt	heur	opt	heur	opt	heur
0	1.00	1.00	1.00	1.00	1.00	1.00
0.2	1.00	1.00	1.00	1.00	1.00	1.00
0.4	1.00	1.00	1.00	1.00	1.00	1.00
0.6	0.99	0.99	1.00	1.00	1.00	1.00
0.8	0.99	0.99	0.99	0.99	1.00	0.99
1	0.99	0.99	0.99	0.99	1.00	0.99

Table 6.5: Results for $F_L(80)$ in Network 1

	$K = 2$		$K = 3$		$K = 4$	
Φ	opt	heur	opt	heur	opt	heur
0	0.95	0.94	0.98	0.96	0.99	0.98
0.2	0.93	0.92	0.96	0.95	0.98	0.97
0.4	0.88	0.88	0.94	0.93	0.96	0.93
0.6	0.85	0.85	0.90	0.90	0.94	0.92
0.8	0.83	0.83	0.86	0.86	0.89	0.88
1	0.81	0.81	0.84	0.84	0.85	0.85

Table 6.6: Results for $F_L(80)$ in Network 2

	$K = 2$		$K = 3$		$K = 4$	
Φ	opt	heur	opt	heur	opt	heur
0	0.56	0.55	0.70	0.67	0.77	0.73
0.2	0.54	0.53	0.67	0.65	0.74	0.71
0.4	0.49	0.48	0.60	0.59	0.67	0.65
0.6	0.43	0.42	0.52	0.51	0.60	0.57
0.8	0.37	0.37	0.43	0.43	0.48	0.46
1	0.33	0.33	0.37	0.37	0.39	0.39

Table 6.7: Results for $F_L(80)$ in Network 3

6.5 Concluding Remarks

In this chapter, we investigated the network delay for the co-located and geographically distributed scenarios, and obtained the analytic results for the network delay distribution. The numerical examples showed that distributing the basic systems geographically has clear performance advantage. Note that our numerical examples were based on the assumption that users are assigned to the closest basic systems. However, if this is not possible, the performance advantage for the geographically distributed scenario may be less significant. We also proposed a heuristic algorithm that can be used to determine the placement of basic systems in a network. Our evaluation showed that this algorithm yields good results, and is efficient in terms of computational requirement.

Chapter 7

Summary and Future Work

7.1 Summary

In this thesis, we investigated the performance characteristics of the two-level hierarchical architecture for DVE systems. Our investigation began by studying the scalability of the two-level architecture. We developed performance models for the overall system, virtual environment and vision domain. Based on these models, we obtained analytic results on the workload experienced by the various basic systems, in terms of the arrival rates of packets submitted by the users. Our numerical examples provided valuable insights into the scalability of the architecture. More importantly, they showed that the architecture has good properties in terms of its ability to support an increasing user population, while keeping the traffic intensity (or the workload) at each basic system at a desired level.

We then studied the issue of consistency among copies of the virtual environment at the various basic systems. We demonstrated how inconsistency may occur and that consistency may be restored as a result of user movement. Furthermore, we proposed a new technique called the virtual vision domain to achieve weak consistency. Our simulation results showed that this technique is effective in improving consistency, but at the cost of a potential increase in the number of basic systems required.

We next focused on the characterization of the update delay, an important performance measure for a DVE system. The two main components of the update delay, namely the server delay and network delay, were investigated by analytic modeling. For the server delay, we first obtained the response time distribution at each basic system and then derived the analytic results for the overall server delay distribution. Using measurement data on an existing DVE server, we gained, by means of numerical examples, valuable insights into the 95th-percentile of the overall server delay.

We derived analytic results for the network delay distribution for two scenarios: co-located and geographically distributed basic systems. We presented numerical results that can be used to characterize the conditions under which geographical distribution would lead to performance advantage. We also proposed a heuristic algorithm for determining the placement of basic systems in a network. We showed that this algorithm yields good results, and is efficient in terms of computational requirement.

Taken together, the above results are important advances in the state of knowledge of the performance characteristics of the two-level hierarchical architecture for DVE systems.

7.2 Suggestions for Future Research

Areas for future work include the following.

7.2.1 Heterogeneous Workload

Our investigation so far has assumed that each basic system is assigned an equal number of users. This implies that the workload at each basic system is balanced and the capacity of the DVE system is best utilized. This assumption may not always be valid. For example, users logging on and out of basic systems may make the workload at these systems imbalanced. Furthermore, in terms of network delay, it may be beneficial to assign a user to the closest basic system. This may, however, lead to imbalanced workload. Therefore, it would be fruitful to investigate the performance of our two-level architecture under scenarios of heterogeneous workload at the basic systems.

7.2.2 Virtual Environment Model

Our investigation was based on relatively simple models of the virtual environment and user behavior. These models are useful in terms of illustrating the performance characteristics of DVE systems. In a complex DVE system, however, the users may behave differently. Hence, it would be be worth extending our investigation to other VE models. Characterizing user behavior, in view of model development and performance analysis, is also an interesting topic.

7.2.3 Global Synchronization

In Chapter 4, we studied the issue of consistency. Specifically, we proposed the virtual vision domain technique, as an alternative to periodic global synchronization, to improve consistency. Further work should be done to investigate periodic global synchronization, for example, how the length of synchronization interval would have an impact on consistency and processing costs. Also, comparison between this approach and our proposed virtual vision domain technique would help us better understand the merits of our technique.

7.2.4 Effective Capacity

Recall that in Chapter 5, we modeled our basic system by a single service facility with multiple processors. In this model, the capacity of a basic system can be expanded by increasing the number of processors (denoted by m_i). Nevertheless, as m_i increases, the processors may have to contend for some other resources, e.g., data access. When this happens, an increase in m_i may not always lead to a proportionate improvement in the capacity. A detailed investigation should be conducted to characterize the effective capacity of a basic system.

7.2.5 Partitioning of the Virtual Environment

To further improve scalability, partitioning of the VE, as described in Section 2.2.6, may be used. The idea is to organize the VE into a number of partitions. Processors are assigned to each partition; these processors are responsible for processing updates associated with avatars within their assigned partition. This approach not only allows the processors of different partitions to service updates in parallel, but also reduces the potential resource contention among them. An interesting question is how to partition a VE and how to allocate a pool of processors to the various partitions so that the update delay is minimized.

When a VE is partitioned, a user in one partition may cross the boundary and enter another partition. Such user movement may incur processing overhead. This may also lead to load imbalance among the processors responsible for the two partitions involved. When there is significant load imbalance, it may be desirable to re-define the boundaries between partitions [31, 33]. Issues such as the overhead of re-partitioning and the frequency with which the loads in the partitions become imbalanced are worth investigating.

Note that a user crossing from one partition to another is analogous to the handoff operation in a cellular network, in which a cellular device moves from one cell to another. The user mobility model investigated in the handoff operation and the related resource allocation schemes [71] would provide valuable insights into the issues related to re-partitioning.

7.2.6 Alternative Two-level Hierarchical Architectures

The two-level hierarchical architecture considered so far uses client-server at the lower level and peer-to-peer at the higher level. Peer-to-peer may also be used at both levels [72] (see Figure 7.1). In this architecture, each user workstation forwards update packets to its assigned basic system as well as to affected users at the same local basic system. Extra capacity is therefore required at the user workstation to transmit the update packets and process the update packets received. Alternatively, one of the local users can perform the function of the basic system. The resulting architecture is similar to the super-peer architecture presented in [73, 74].

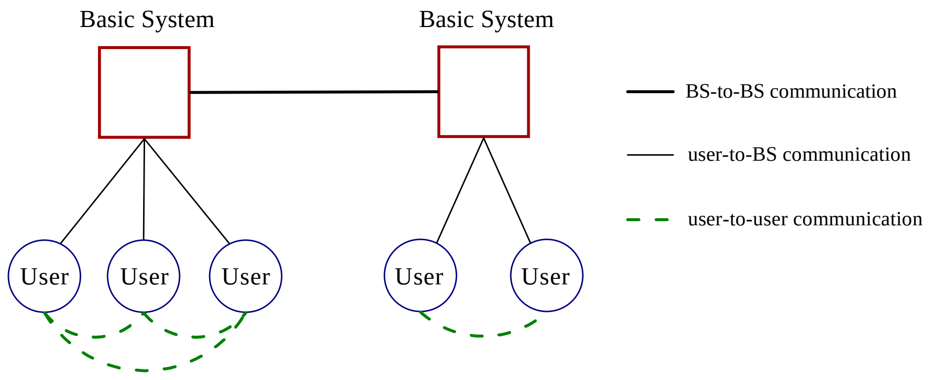


Figure 7.1: Alternative Two-level Hierarchical Architecture

A performance comparison of the two-level architecture investigated in this thesis and the alternative architectures discussed above is an interesting extension of our work.

Appendix A

Derivation of $M/M/m$ Response Time Distribution

In this appendix, we derive the analytic results for the response time distribution in an $M/M/m$ model.

Let $F_X(t)$ be the cumulative distribution function (c.d.f.) of the response time. Also, let $F_R(t)$ and $F_H(t)$ be the c.d.f. of the service time and the queueing delay, respectively. We know $F_X(t)$ is the convolution of $F_R(t)$ and $F_H(t)$, i.e.,

$$F_X(t) = F_R(t) \otimes F_H(t). \tag{A.1}$$

For an $M/M/m$ model, the number of parallel processors is m, the arrival process is Poisson with rate λ, and the service time distribution is exponential with c.d.f. given by:

$$F_R(t) = 1 - e^{-\mu t}. \tag{A.2}$$

We first derive the distribution of queueing delay. For an $M/M/m$ model, the arrival and departure rates of requests are:

$$\begin{aligned}
&\lambda(n) = \lambda && n \geq 0 \\
&\mu(n) = \min(m, n)\mu && n > 0,
\end{aligned}$$

respectively. Let the state be the number of requests in the system (denoted by n). The steady state probability can be shown to be [75]:

$$P(n) = P(0) \prod_{i=0}^{n-1} \frac{\lambda(i)}{\mu(i+1)} = \begin{cases} P(0)\frac{\sigma^n}{n!} & n < m, \\ P(0)\frac{\sigma^n}{m!m^{n-m}} & n \geq m \end{cases}$$

where $\sigma = \frac{\lambda}{\mu}$. Let

$$\begin{aligned}
G &= \sum_{n=0}^{\infty} \prod_{i=0}^{n-1} \frac{\lambda(i)}{\mu(i+1)} \\
&= \sum_{n=0}^{m-1} \frac{\sigma^n}{n!} + \sum_{n=m}^{\infty} \frac{\sigma^n}{m!m^{n-m}} \\
&= \sum_{n=0}^{m-1} \frac{\sigma^n}{n!} + \frac{m\sigma^m}{m!(m-\sigma)}.
\end{aligned}$$

We have

$$P(0) = \frac{1}{G} = \left(\sum_{n=0}^{m-1} \frac{\sigma^n}{n!} + \frac{m\sigma^m}{m!(m-\sigma)} \right)^{-1}$$

if G converges (or $\sigma < m$).

Now, consider a tagged arrival. First, let

$$\kappa = Pr[\text{the tagged arrival has to wait}].$$

κ can be obtained by:

$$\begin{aligned}\kappa &= \sum_{n \geq m} P(n) \\ &= P(0)\frac{\sigma^m}{m!}\sum_{n \geq m}\left(\frac{\sigma}{m}\right)^{n-m} \\ &= P(0)\frac{\sigma^m}{m!}\frac{1}{1-\frac{\sigma}{m}} \\ &= P(0)\frac{\sigma^m}{m!}\frac{m}{(m-\sigma)}.\end{aligned}$$

Then, let

$$\begin{aligned}y(n,t) =& Pr[\text{queueing delay} \leq t|\text{number of requests in system seen by tagged} \\ & \text{arrival} = n].\end{aligned}$$

Then,

$$\begin{aligned}F_H(t) &= Pr[\text{queueing delay} \leq t] \\ &= \sum_n y(n,t)P(n).\end{aligned}$$

Consider the following two cases:

- Case 1: $n < m$:

$$y(n, t) = 1$$

  because no queueing occurs.

- Case 2: $n \geq m$:

$$y(n, t) = Pr[(n - m + 1) \text{ requests finish their services}].$$

  $y(n, t)$ is given by $F_{E_{n-m+1,m\mu}}(t)$, the c.d.f. of the Erlang distribution with parameters $(n-m+1)$ and $m\mu$. It can be interpreted as the sum of $(n-m+1)$ independent random variables, each of which is exponentially distributed with parameter $m\mu$. We thus have:

$$F_{E_{n-m+1,m\mu}}(t) = \int_0^t \frac{(m\mu)^{n-m+1} x^{n-m} e^{-m\mu x}}{(n-m)!} dx.$$

Combining the above two cases, we have:

$$\begin{aligned}
F_H(t) &= \sum_{n<m}(1)P(n) + \sum_{n\geq m} F_{E_{n-m+1,m\mu}} P(n) \\
&= 1-\kappa + \sum_{n\geq m}\left[\int_0^t \frac{(m\mu)^{n-m+1}x^{n-m}e^{-m\mu x}}{(n-m)!}dx\right]\left[P(0)\frac{\sigma^n}{m!m^{n-m}}\right] \\
&= 1-\kappa + P(0)\frac{m\mu\sigma^m}{m!}\left[\int_0^t \left(e^{-m\mu x}\sum_{n\geq m}\frac{(\mu x\sigma)^{n-m}}{(n-m)!}\right)dx\right] \\
&= 1-\kappa + P(0)\frac{m\mu\sigma^m}{m!}\left[\int_0^t \left(e^{-m\mu x}e^{\mu x\sigma}\right)dx\right] \qquad (A.3) \\
&= 1-\kappa + P(0)\frac{m\mu\sigma^m}{m!}\left[\int_0^t e^{-\mu x(m-\sigma)}dx\right] \\
&= 1-\kappa + P(0)\frac{m\sigma^m}{m!}\frac{1-e^{-\mu t(m-\sigma)}}{m-\sigma} \\
&= 1-\kappa + \kappa\left(1-e^{-\mu t(m-\sigma)}\right) \\
&= 1-\kappa e^{-\mu t(m-\sigma)}.
\end{aligned}$$

Finally, substituting Equations A.2 and A.3 into Equation A.1, we obtain the following results for the response time distribution:

$$\begin{aligned}
F_X(t) &= F_R(t) \otimes F_H(t) \\
&= \int_0^t F_H(t-x) dF_R(x) dx \\
&= \int_0^t (1 - \kappa e^{\mu(t-x)(m-\sigma)}) \mu e^{-mux} dx \\
&= \int_0^t \left(\mu e^{-\mu x} - \kappa \mu e^{-\mu(t(m-\sigma)+(1-m+\sigma))x} \right) dx \\
&= \begin{cases} 1 - e^{-\mu t} - \kappa \mu e^{-\mu t(m-\sigma)} \left[\frac{1-e^{-\mu t(1-m+\sigma)}}{(1-m+\sigma)} \right] & \text{for } \sigma \neq m-1 \\ 1 - e^{-\mu t} - \kappa \mu e^{-\mu t} t & \text{for } \sigma = m-1. \end{cases}
\end{aligned}$$

Appendix B

Measurement of Service Time of Updates on a DVE Server

Recall that we use μ to denote the service rate of update and syn packets, and the mean service time of these packets is simply given by $1/\mu$. To better understand the service time of an update at a basic system, we conducted a measurement experiment on an existing DVE server, called RockyMud [66].

RockyMud is a text-based multi-player online game. It uses a client-server architecture; users access the game server via a telnet program on their workstations. The virtual environment in RockyMud comprises roughly 10,000 rooms and includes buildings, cities, dungeons, and open areas. Users chat, hunt, and travel within this environment.

Our objective is to measure the time required by the server to process an update sent by a user, specifically, the update resulting from a move operation. We measure the processing time spent in each of these "move" updates by integrating measurement code into the server program.

We set up a RockyMud game server on an Intel P3 1.2 GHz machine with 256MB RAM, which runs on a version 2.4.18 Linux kernel. The server is executed with a single thread. We also run an emulated user program on a SUN Ultra 60 machine with 512 MB RAM and 450 MHz processor, which runs Solaris 8. These two machines are connected directly by a 100 Mbps Ethernet network; the network delay between them is found to be minimal.

In our experiment, the emulated user program performs 10,000 moves (in random directions). This means that 10,000 move updates are sent and processed by the server. Upon receiving a move update from the user program, the server processes it and sends back an acknowledgment to this program, which then sends the next move update to the server.

Our experiment results show that the mean service time of a move update is around 0.33 ms and its standard deviation is 0.048 ms.

Appendix C

Summary of Notations

Symbol	Interpretation
A	The width of the virtual environment
B	The height of the virtual environment
BS_i	Basic system i
$C_{i,i}$	The rate at which updates are received by affected users at the same local system
$C_{i,j}$	The rate at which updates are received by affected users at a remote basic system BS_j, $j \neq i$
$d_{m,n}$	The signal propagation delay on the channel between nodes m and n
$h(a,b)$	The probability that a user is within the vision domain of a tagged user at location (a,b), assuming that every user has the same behavior
$h_{i,u;j,v}(a,b)$	The probability that user (j,v) is within the vision domain of user (i,u) who is at location (a,b)
$g_{j,k}$	The probability that a syn-packet is to be sent from BS_j to BS_k
$F_L(t)$	The c.d.f. of the network delay
$F_S(t)$	The c.d.f. of the overall server delay
$F_{S_i}(t)$	The c.d.f. of the server delay of updates submitted by users at BS_i
$F_{X_i}(t)$	The c.d.f. of the server delay of an update processed by BS_i
$F_{X_{i,j}}(t)$	The c.d.f. of the server delay of an update processed by BS_i and BS_j

Symbol	Interpretation
$I(t)$	The number of users who are in the inconsistent state at a given time t
$\overline{I}(t)$	The mean number of users who are in the inconsistent state at a given time t
$\overline{I}$	The mean number of users who are in the inconsistent state at steady state
I_{frac}	The fraction of users who are in the inconsistent state
I_{time}	The mean length of time that a user is in the inconsistent state
(i, u)	User u at BS$_i$
K	The number of basic systems deployed
K_{min}	The minimum number of basic systems required to support a given number of users while maintaining traffic intensity at each basic system below a given level
L	The network delay
$L_{i,u;j,v}$	The network delay between users (i, u) and (j, v)
L_{max}	The estimate of the largest network delay that would result in realistic interaction
m_i	The number of processors at BS$_i$
N	The total number of users in the DVE system
N_i	The number of users at BS$_i$
$p_{a,b}$	The steady state probability that a user is at location (a, b), assuming that every user has the same behavior
$p^{(i,u)}_{a,b}$	The steady state probability that user (i, u) is at location (a, b)
Q	The number of potential basic system end-points
$q_{a,b;c,d}$	The probability that a user moves from location (a, b) to (c, d) in one step
RE$_i$	The region of BS$_i$
r	The size of a region
S	The server delay
s_p	The p-th percentile of the server delay distribution
VE	The virtual environment
T	The update delay
$t_{m,n}$	The packet transmission time on the channel between nodes m and n
VE$_i$	The copy of the virtual environment maintained at BS$_i$
U	The width of the vision domain
V	The height of the vision domain

Symbol	Interpretation
$W_{i,u;i,v}$	The rate at which updates submitted by user (i,u) affect user (i,v) (at the same local system)
$W_{i,u;j,v}$	The rate at which updates submitted by user (i,u) affect user (j,v) (at a remote system BS_j)
y	The traffic intensity parameter
$y_{i,j}$	The network delay from BS_i to BS_j
$z_{i,u}$	The network delay from BS_i to user u who is connected to BS_i
γ_i	The arrival rate of update packets to BS_i
η_i	The arrival rate of syn packets to BS_i
$\eta_{j,i}$	The arrival rate of syn packets from BS_j to BS_i
λ_i	The total arrival rate of packets to BS_i
μ	The service rate of each processor at a basic system
μ_i	The processing capacity of BS_i
$\xi_{j,k}(n)$	The probability that n other users at BS_k are within the vision domain of a tagged user at BS_j
ρ_i	The traffic intensity at BS_i
Φ	The fraction of overlap between two adjacent regions
ϕ	The rate at which a user makes a move (or submits an update)
$\psi_{i,u;j,v}$	The probability that user (j,v) is within user (i,u)'s vision domain

Bibliography

[1] "World of WarCraft." http://www.worldofwarcraft.com/.

[2] "Everquest." http://www.everquest.com/.

[3] "Final Fantasy XI Online." http://www.playonline.com/ff11us/.

[4] "Lineage." http://www.lineage.com/.

[5] "Quake." http://www.idsoftware.com/games/quake/quake/.

[6] "Starcraft." http://www.blizzard.com/starcraft/.

[7] R. Baecker, *Readings in Groupware and Computer Supported Cooperative Work: Assisting Human-Human Collaboration*. Morgan Kaufman Press, 1993.

[8] R. Stuart, *The Design of Virtual Environments*. McGraw-Hill, New York, 1996.

[9] L. Zou, M. Ammar, and C. Diot, "An evaluation of grouping techniques for state dissemination in networked multi-user games," in *Proceedings of the 9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, (Cincinnati, Ohio), pp. 33–40, August 2001.

[10] C. Diot and L. Guatier, "A distributed architecture for multiplayer interactive applications on the Internet," *IEEE Network*, vol. 13, pp. 6–15, August 1999.

[11] B. Neuman, "Scale in distributed systems," *Readings in Distributed Computing Systems*, pp. 463–489, 1994.

[12] E. Cronin, B. Filstrup, and A. Kurc, "A distributed multiplayer game server system." UM EECS589 Course Project Report, University of Michigan.

[13] T. Das, G. Singh, A. Mitchell, P. Kumar, and K. McGee, "NetEffect: A network architecture for large-scale multi-user virtual worlds," in *Proceedings of ACM VRSTI'97*, (Lausanne, Switzerland), pp. 157–163, 1997.

[14] T. Funkhouser, "RING: A client-server system for multiuser virtual environment," in *Proceedings of SIGGRAPH '95*, pp. 82–92, April 1995.

[15] J. Muller, J. Metzen, and A. Ploss, "Rokkatan: Scaling an RTS game design to the massively multiplayer realm," in *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005)*, (Valencia, Spain), pp. 125–132, June 2005.

[16] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The SIMNET virtual world architecture," in *Proceedings of the IEEE Virtual Reality Annual International Symposium*, (Seattle, WA), pp. 450–455, 1993.

[17] IEEE, *ANSI/IEEE Std 1278-1993: Standard for Information Technology, Protocols for Distributed Interactive Simulation*. 1993.

[18] M. Matijasevic, "A review of networked multi-user virtual environment," Tech. Rep. 97-8-1, Center for Advanced Computer Studies, University of Southwestern Louisiana, 1997.

[19] E. Frecon, "DIVE: A scaleable network architecture for distributed virtual environment," *Distributed Systems Engineering Journal (Special Issue on Distributed Virtual Environments)*, vol. 5, no. 3, pp. 91–100, 1998.

[20] L. Guatier and C. Diot, "Design and evaluation of MiMaze, a multi-player game on the Internet," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pp. 233–236, June 1998.

[21] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz, "NPSNET: A network software architecture for large scale virtual environments," *Presence*, vol. 3, no. 4, pp. 265–287, 1994.

[22] IEEE, *Dead Reckoning Definition and Algorithms.* 1995.

[23] A. Katz and K. Grahman, "Dead reckoning for airplanes in coordinated flight," in *Proceedings of the 10th Workshop on Standards for the Interoperability of Defense Simulations*, vol. 2, (Orlando, FL), pp. 5–13, 1994.

[24] S. Singhal and D. Cheriton, "Exploiting position history for efficient remote rendering in networked virtual reality," *Presence*, vol. 4, no. 2, pp. 169–193, 1995.

[25] M. Bassiouni, M. Chiu, M. Loper, M. Garnsey, and J. Williams, "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, 1997.

[26] D. McGregor and A. Kapolka, "NPSNET-V: A new beginning for dynamically extensible virtual environments," *IEEE Computer Graphics and Applications*, vol. 20, no. 5, pp. 12–15, 2000.

[27] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, and P. Barham, "Exploiting reality with multicast groups," *IEEE Computer Graphics and Applications*, vol. 15, no. 5, pp. 38–45, 1995.

[28] M. Wathen, *Dynamic Scalable Network Area of Interest Management for Virtual Worlds.* Master's thesis, Naval Postgraduate School, Monterey, California, 2001.

[29] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation.* ACM Press, New York, 1999.

[30] T. Funkhouser, "Network topologies for scalable multi-user virtual environments," in *Proceedings of the 1996 IEEE Virtual Reality Annual Interaction Symposium (VRAIS)*, (San Jose, CA), pp. 222–228, 1996.

[31] J. Lui and M. Chan, "An efficient partitioning algorithm for distributed virtual environment systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 193–211, March 2002.

[32] J. Lui, M. Chan, O. So, and T. Tam, "Balancing workload and communication costs for a distributed virtual environment," in *Proceedings of the 4th Interna-*

tional Workshop on Advances in Multimedia Information Systems, (London, UK), pp. 130–135, Springer-Verlag, 1998.

[33] D. Min, E. Choi, D. Lee, and B. Park, “A load balancing algorithm for a distributed multimedia game server architecture,” in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, (Florence, Italy), pp. 882–886, 1999.

[34] G. Singh, L. Serra, W. Png, and H. Ng, “Bricknet: A software toolkit for network-based virtual environments,” *Presence*, vol. 3, no. 1, pp. 19–34, 1994.

[35] B. Ng, A. Si, R. Lau, and F. Li, “A multi-server architecture for distributed virtual walkthrough,” in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, (New York, NY, USA), pp. 163–170, ACM Press, 2002.

[36] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W. Yerazunis, “Diamond Park and Spline: A social virtual reality system with 3D animation, spoken interaction, and runtime modifiability,” Tech. Rep. TR-96-02a, MERL, November 1996.

[37] Y. Saito and M. Shapiro, “Optimistic replication,” *ACM Computing Surveys*, vol. 37, no. 1, pp. 42–81, 2005.

[38] O. Kariv and S. Harkim, “An algorithmic approach to network location problem I: The p-centers,” *SIAM Journal of Applied Mathematics*, vol. 37, no. 3, pp. 513–538, 1979.

[39] O. Kariv and S. Harkim, “An algorithmic approach to network location problem II: The p-medians,” *SIAM Journal of Applied Mathematics*, vol. 37, no. 3, pp. 539–560, 1979.

[40] M. Brandeau and S. Chiu, “An overview of representative problems in location research,” *Management Science*, vol. 35, pp. 645–674, June 1989.

[41] P. Dearing, R. Francis, and T. Lowe, “Convex location problems on tree networks,” *Operations Research*, vol. 24, pp. 628–642, 1976.

[42] P. Mirchandani and R. Francis, *Discrete Location Theory.* New York: John Wiley and Sons, Inc., 1990.

[43] P. Krishnan, D. Raz, and Y. Shavitt, “The cache location problem,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 568–582, 2000.

[44] B. Li, M. Golin, G. Italiano, and X. Deng, “On the optimal placement of web proxies in the Internet,” in *Proceedings of IEEE INFOCOM 1999*, (New York, NY), pp. 1282–1290, 1999.

[45] L. Qiu, V. Padmanabhan, and G. Voelker, “On the placement of web server replicas,” in *Proceedings of IEEE INFOCOM 2001*, (Anchorage, Alaska), pp. 1587–1596, April 2001.

[46] B. Li, F. Chen, and L. Yin, “Server replication and its placement for reliable multicast,” in *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, (Las Vegas, Nevada), pp. 396–401, 2000.

[47] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of Internet instrumentation," in *Proceedings of IEEE INFOCOM 2000*, (Tel-Aviv, Israel), pp. 295–304, March 2000.

[48] M. Ye and L. Cheng, "System-performance modeling for massively multiplayer online role-player games," *IBM Systems Journal*, vol. 45, no. 1, pp. 45–50, 2006.

[49] A. Pope and R. Scaffer, "The SIMNET network and protocols," Tech. Rep. BBN Rep. No. 7627, Cambridge, MA, June 1991.

[50] D. Miller, "SIMNET: The advent of simulator networking," *IEEE*, vol. 83, no. 8, pp. 1114–1123, 1995.

[51] K. Savetz, N. Randall, and Y. Lepage, *MBONE: Multicasting Tomorrow's Internet*. John Wiley and Sons Inc., 1996.

[52] R. Bangun, E. Dutkiewicz, and G. Anido, "An analysis of multi-player network games traffic," in *Proceedings of the IEEE 3rd Workshop on Multimedia Signal Processing*, pp. 3–8, September 1999.

[53] M. Borella, "Source models of network game traffic," *Computer Communications*, vol. 23, pp. 403–410, February 2000.

[54] J. Farber, "Network game traffic modelling," in *Proceedings of the 1st ACM Workshop on Network and System Support for Games (NetGames)*, (Bruanschweig, Germany), pp. 53–57, April 2002.

[55] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in Unreal Tournament 2003," in *Proceedings of the 3rd ACM Workshop on Network and System Support for Games (NetGames)*, (Portland, OG), pp. 144–151, September 2004.

[56] J. Nichols and M. Claypool, "The effects of latency on online Madden NFL Football," in *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, (Kinsale, County Cork, Ireland), pp. 146–151, June 2004.

[57] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The effect of latency on user performance in Warcraft III," in *Proceedings of the 2nd ACM Workshop on Network and System Support for Games (NetGames)*, (Redwood City, CA), pp. 3–14, May 2003.

[58] M. Claypool, D. LaPoint, and J. Winslow, "Network analysis of Counter-strike and StarCraft," in *Proceedings of the 22nd IEEE International Performance Computing, and Communications Conference (IPCCC)*, (Phoenix, Arizona), pp. 261–268, April 2003.

[59] W. Feng, F. Chang, and J. Walpole, "Provisioning on-line games: A traffic analysis of a busy Counter-Strike server," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement*, pp. 151–156, November 2002.

[60] A. Abdelkhalek, A. Bilas, and A. Moshovos, "Behavior and performance of interactive multi-player game servers," *Cluster Computing*, vol. 6, no. 4, pp. 355–366, 2003.

[61] A. Abdelkhalek and A. Bilas, "Parallelization and performance of interactive multiplayer game servers," in *Proceedings of the 2004 International Parallel and Distributed Processing Symposium (IPDPS2004)*, (Santa Fe, New Mexico), p. 72, April 2004.

[62] L. Pantel and L. Wolf, "On the impact of delay on real-time multiplayer games," in *Proceedings of the 12th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, (Miami Beach, Florida), pp. 23–29, May 2002.

[63] M. Kwok and G. Yeung, "Characterization of user behavior in a multi-player online game," in *Proceedings of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005)*, (Valencia, Spain), pp. 69–74, June 2005.

[64] M. Karsten, J. Song, M. Kwok, and T. Brecht, "Efficient operating system support for group unicast," in *Proceedings of the 15th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, (Skamania, Washington), June 2005.

[65] T. Lang, G. Armitage, P. Branch, and H. Choo, "A synthetic traffic model for Half-life," in *Proceedings of the Australian Network and Telecommunications Conference 2003*, (Melbourne, Australia), December 2003.

[66] "Rockymud." http://www.rockymud.net/.

[67] L. Peterson and S. Davie, *Computer networks: A system approach*. San Francisco: Morgan Kaufmann Publishers, 2000.

[68] "Boston university Representative Internet Topology gEnerator BRITE." http://www.cs.bu.edu/fa/matta/software.html.

[69] B. Waxman, "Routing of multipoint connections," *IEEE Journal of Selected Areas Communications*, vol. 6, pp. 1617–1622, December 1988.

[70] E. Zegura, K. Calvert, and M. Donahoo, "A quantitative comparison of graph-based models for Internetworks," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 770–783, December 1997.

[71] E. Halepovic and C. Williamson, "Characterizing and modeling user mobility in a cellular data network," in *Proceedings of the 2nd ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*, (New York, NY, USA), pp. 71–78, ACM Press, 2005.

[72] S. Singhal, B. Nguyen, R. Redpath, J. Nguyen, and M. Fraenkel, "Inverse: Designing an interactive universe architecture for scalability and extensibility," in *Proceedings of the 6th International Symposium on High Performance Distributed Computing (HPDC '97)*, (Portland, OR), pp. 61–70, August 1997.

[73] B. Yang and H. Garcia-Molina, "Designing a super-peer network," in *Proceedings of the 19th International Conference on Data Engineering*, (Banglore, India), pp. 49–63, March 2003.

[74] L. Xiao, Z. Zhuang, and Y. Liu, "Dynamic layer management in superpeer architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 11, pp. 1078–1091, 2005.

[75] L. Kleinrock, *Queueing Systems Volume I: Theory.* New York: John Wiley and Sons Inc., 1975.