淡江大學資訊工程學系碩士班

碩士論文

指導教授： 陳瑞發 博士

以同儕運算為基礎之可擴展式網路型態虛擬環境

Scalable Peer-to-Peer Networked Virtual Environment

研究生： 胡舜元 撰

中 華 民 國 九 十 四 年 一 月

淡江大學資訊工程學系碩士班

碩士論文

指導教授： 陳瑞發 博士

以同儕運算為基礎之可擴展式網路型態虛擬環境

Scalable Peer-to-Peer Networked Virtual Environment

研究生： 胡舜元 撰

中華民國九十四年一月

淡江大學資訊工程學系碩士班

碩士論文

指導教授： 陳瑞發 博士

以同儕運算為基礎之可擴展式網路型態虛擬環境

Scalable Peer-to-Peer Networked Virtual Environment

研究生： 胡舜元 撰

中華民國九十四年一月

論文名稱： **頁數：63**

以同儕運算為基礎之可擴展式網路型態虛擬環境

校系(所)組別： 淡江大學 資訊工程學系碩士班 A 組

畢業時間及提要別： 九十三年度第一學期 碩士學位論文提要

研究生： 胡舜元 **指導教授：** 陳瑞發 博士

關鍵字： 網路虛擬環境、同儕運算、多人線上遊戲、Voronoi、擴展性、有效互動空間管理、VON

論文提要內容：

「網路虛擬環境」(Networked Virtual Environment, 簡稱 NVE) 是整合 3D 圖學及電腦網路的新興領域。它使分散各地的使用者，能在同一個虛擬空間中即時進行互動。從 80 年代的軍事模擬到近年來熱門的「多人線上遊戲」(Massively Multiplayer Online Game, 簡稱 MMOG) 皆為「網路虛擬環境」之實例。目前多數系統採用「主從式架構」 (client-server) 為其通訊模式。但主從式架構之總資源 (運算或頻寬資源) 通常有其上限，若要發展下一代，可由百萬人同時使用的系統，則會面臨「擴展性」 (scalability) 方面的問題。

本論文提出一種完全分散、以同儕運算 (peer-to-peer computing) 及 Voronoi diagram 為基礎之虛擬網路 (Voronoi-based Overlay Network, 簡稱 VON)，來解決「網路虛擬環境」之擴展性問題。此方式較同類型架構簡易且有效率，並善用虛擬環境中，人物互動範圍有限之特質。VON 可用來建構高度擴展性、低延遲、且可容錯的網路虛擬環境。

Title of Thesis: Total pages: 63

Scalable Peer-to-Peer Networked Virtual Environment

Keywords:

Networked Virtual Environment (NVE), peer-to-peer (P2P), massively multiplayer (MMP), Voronoi diagram, scalability, interest management、VON

Name of Institute:

Department of Computer Science and Information Engineering
Tamkang University

Graduate date: 01/2005 **Degree conferred:** Master

Name of student: Shun-Yun Hu 胡舜元 **Advisor:** Dr. Jui-Fa Chen 陳瑞發 博士

Abstract:

Networked Virtual Environment (NVE) is an emerging discipline that combines the fields of computer graphics and computer network to allow many geographically dispersed users interact simultaneously in a shared virtual environment. From the early military simulation to the recently popular Massively Multiplayer (MMP) Online Games, we see examples of NVE ever more integrated to aspects of our lives. Currently client-server is the predominant architecture for NVE systems, however, it faces inherent scalability problem as it usually has a limited total amount of system resources (i.e. CPU and bandwidth), which makes it unsuitable for constructing a NVE system sharable by millions of users.

We propose a fully-distributed peer-to-peer architecture to solve the scalability problem of Networked Virtual Environment in a simple and efficient manner. Our method exploits locality of user interest inherent to such systems and is based on the mathematical construct *Voronoi diagram*. The proposed Voronoi-based Overlay Network (VON) allows scalable, responsive, fault-tolerant NVE be constructed and deployed in an affordable way.

To 88<

For sharing with me
the *best time* in my life

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would first like to thank my advisor, Dr. Jui-Fa Chen (陳瑞發老師), for taking me as his student and giving me responsive feedbacks to improve both the content and presentation of this thesis. I am also grateful to Dr. Wei-Chuan Lin (林偉川老師), for his feedbacks during group meetings and the suggestion to devise metrics to compare various P2P approaches. Feedbacks and comments given by members of the Alpha Lab, in particular, Hua-Hsen Bai (白華勝), were also much appreciated.

This work originated with my collaboration with Guan-Ming Liao (廖冠名). We had many exciting discussions and interesting brainstorm sessions while traveling between Chia-Yi and Taipei, and during our regular weekend hiking. We discussed and wrote the code for the early prototype together. Specifically, I would like to thank Guan-Ming for the very robust and reliable implementation of the Voronoi algorithm.

This research would not have attained its scale without the many people who gave timely feedbacks and criticisms at various stages. These valuable comments and suggestions had broaden and deepen my understanding of the problem: Joaquin Keller and Simon Gwendal of France Telecom R&D (authors of Solipsis), Bart Whitebook of butterfly.net, Jon Watte of There.com, Lili Qui of Microsoft Research, Crosbie Fitch of Cyberspaceengineers.org, Chun-Wen Chen (陳俊文) of TKU WISE Lab and the anonymous reviewers for my Netgame2004 workshop paper.

I am thankful to my previous advisor, Prof. Wen-Bing Horng (洪文斌老師), for giving me the freedom and secured environment to work on a research topic that I felt greatly interested. I would also like to give special thanks to Prof. Jiung-yao Huang (黃俊堯老師), for introducing the field of Networked Virtual Environment to me, and for giving me the opportunity to participate in my first NVE-related project.

Sincere thanks go to my father, Dr. Chin-Kun Hu (胡進錕老師), for his consistent encouragement and support to me on embarking this research when prospects of definite results were still uncertain (he believes that when given adequate support, young people are capable to make important research contributions); and to my mother, whose persistent efforts made possible my pursuit of graduate studies.

This work was supported by Laboratory of Statistical and Computational Physics (LSCP) of Institute of Physics, Academia Sinica, Taiwan.

1. Introduction

Computer simulation of real-world environments is an important field that has wide applications in military and corporate training, science, education, and entertainment. As hardware and software technologies progress, we see ever more detailed and realistic simulations. Advances in processing power, network bandwidth, and 3D graphical acceleration have enabled a new class of sophisticated networked simulations that is visually presented in 3D, and sharable by many users in real-time. This emerging field is loosely called *Networked Virtual Environment* [1][Singhal 99] (or NVE for short). Applications of NVE have evolved from military training simulation in the 80's to the recently booming massively multiplayer online games (MMOG) in the 90's.

One important challenge in NVE is to allow as many people as possible to interact in the same environment, in a smooth and responsive manner. Allowing thousands or even millions of people in the same virtual world creates potential for new types of Internet applications and social phenomena. However, while we have seen improvements in the *scalability* of NVE systems, so far there has not been a truly massive and scalable NVE that exists on a global scale.

To create a NVE, a number of important issues must be considered, namely:

Consistency / Synchronization - For meaningful interactions to happen, each user's experiences in the virtual world must be more or less consistent. This includes maintaining shared states and keeping events synchronized.

Responsiveness - NVEs are simulations of the real world. Responsiveness therefore is important for immersion. However, requirements vary between applications (e.g. latency less than 200ms is required for a fast-paced first-person computer game, yet up to several seconds can be tolerated for a real-time strategy game [Knutsson 04]).

Security - Most NVEs allow people to engage competitively (e.g. combat or treasure hunt). User authentication and fairness against cheating therefore are required. In fact, this is often the most concerned aspect for commercial NVE developers.

[1] Other names include *Distributed Virtual Environment* (or DVE) [Stytz 96], *Collaborative Virtual Environment* (or CVE), *Computer-Supported Collaborative Work* (or CSCW). In this thesis we will use the abbreviation *NVE* to describe all networked simulations of virtual worlds.

Scalability – Scalability usually concerns with the number of simultaneous users in NVE [Singhal 99]. It is important in two respects:

1. **Content possibility**. Certain game plays are only realizable when many people can participate, such as community and social-oriented game play.
2. **Service availability.** Large-scale NVEs are similar to websites, where usage may increase dramatically and unexpectedly. Systems will break if they are not scalable.

Persistency - To allow sophisticated contents, certain data, such as user profile and valuable virtual objects, must be persistently stored and accessed between user sessions.

Reliability / Fault-tolerance – User experience is negatively affected if a simulation session suddenly breaks down due to server failure. Reliability is thus important to make NVE a service with quality.

We consider *scalability* as the most important issue if we plan to build truly massive worlds and applications, which millions of people can participate and enjoy. Therefore, this thesis focuses on finding a feasible solution for the scalability problem in NVE that may also be the foundation for solving other issues.

Existing approaches to improve scalability mainly rely on enhancing server capacity in *client-server* architecture. However, client-server architecture has an inherent upper limit in its available resource (i.e. processing and bandwidth capacity), it is also expensive to deploy and maintained. On the other hand, *peer-to-peer* (P2P) architecture has emerged in recent years as an alternative that promises scalability and affordability. Since its introduction, it has become highly publicized by large-scale distributed processing application such as SETI@Home [Korpela 01] (a University of Berkeley project that allows people around the world to donate spare CPU time to analyze astronomical data with a simple screen saver), file-sharing applications such as Napster [Napster], Gnutella [Gnutella], FreeNet [Clarke 00], eDonkey [eDonkey], and Voice-over-IP (VoIP) application such as Skype [Skype].

We attempt to apply P2P architecture to NVE design, in hope that it may solve the scalability problem in an efficient way. Our approach uses a mathematical construct called Voronoi diagram [Guibas 85] for constructing a unique P2P network that supports virtual environment applications. The proposed architecture is called *Voronoi-based Overlay Network* (abbreviated as VON).

1.1 Thesis statement

Scalable, responsive, and fault-tolerant Networked Virtual Environment can be constructed with peer-to-peer architecture based on Voronoi diagram.

1.2 Motivation and goal

We believe that a massive, persistent 3D virtual environment which allows millions of people to participate simultaneously may eventually happen on Internet as a major communication medium. There are many technical, architectural issues that need to be resolved before such a *true cyberspace* can materialize. Chief among the issues is a scalable architecture that accommodates large number of simultaneous users. We therefore choose to devise a feasible architecture for constructing scalable virtual environments.

We note that peer-to-peer architecture holds better promise to achieve scalability than client-server architecture. The goal for this thesis, therefore, is to devise a suitable peer-to-peer architecture for constructing NVE that may be scalable to millions of users.

1.3 World model and assumptions

In order to simplify the problem for this thesis, we assume that the virtual environment is a 2D coordinate space with specific width and height. Events in the world are modeled with discrete time-steps. Participants in the NVE consist of many computers (referred to as *nodes*) scattered around the globe. All nodes are connected to the Internet and may establish direct TCP connection (a peer-to-peer connection) with each other. Each node is identified by a system-wide unique ID and a 2D coordinate to indicate its current position. Nodes may change positions and move at a certain velocity in any direction at each time-step.

To simplify our analysis, object states (or *game states*) in a complex virtual environment such as MMOG (for example, player health points, virtual items, and computer-controlled agents) are not considered. The only state update we consider is position update that happens at regular intervals (for example, 5 – 10 times per second). Network delay (or *latency*) is also not considered, and we assume that packets are received and processed by the remote nodes immediately after they are generated.

We also assume that all computers can be trusted (e.g. there are no malicious attempts at cheating or non-compliance to the proposed protocols). In other words, among the criteria for constructing NVE, we currently do not consider event consistency, security, and persistency issues.

1.4 The scalability problem

Scalability is a phenomenon observed in many natural and artificial systems. We see systems that accommodate components (or nodes) in a wide range of numbers as being "*scalable*". There are two main characteristics in scalable systems:

- **Joinability:** components (or nodes) may be added to the system.
- **Maintainability**: system remains functional after various nodes enter or leave the system.

Existing resources in any given system is usually finite, and are consumed at end-point when a new node is added. For example, when we add new routers to Internet, bandwidth of existing routers is consumed. A system is "joinable" only when nodes that accept new nodes have enough spare resources. Likewise, maintainability is sustained only when resource is not depleted after new nodes join. Two more properties exist to counter the problem of resource depletion:

- **Resource-growing**: useful system resources (i.e. resources at the accepting nodes) increase with the addition of new nodes
- **Decentralized end-point resource consumption**: addition of a node does not consume some "centralized" resource.

Resource-growing is a general strategy found in almost all scalable systems (reducing consumption works to similar effect). In the Internet example, although adding routers consumes bandwidth, it also contributes new resource that can be used to accommodate additional routers. Decentralized resource consumption, on the other hand, is a weaker requirement. As long as resources are available at the accepting nodes, a system can still be "joinable" and "maintainable" even if it is done in a centralized fashion. For example, in a server-cluster, as long as server resources (bandwidth and processing capability) can be increased, then scalability is maintained [Butterfly 03]. However, such a design can be costly and complicated to maintain in practice, and most truly scalable systems (such as Internet) exhibit decentralized resource consumption.

From the above discussion, we expect that to build a *truly scalable* NVE, one that may accommodate more users by orders of magnitude than existing systems, we need architectures that can grow its resource, and does not require centralized resource when additional users join.

1.4.1 Problem definition

The main problem that we try to address is: given some points (or *nodes*) moving continuously on a 2D plane, each has a radius that defines its *Area of Interest* (or *AOI,* see Figure 1). If all nodes must exchange messages with nodes within its AOI (called *AOI neighbors*), how can it be done in a scalable and efficient manner?

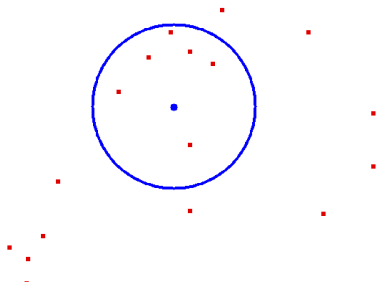


Figure 1: Concept of Area of Interest (AOI)

Each dot represents a node in the virtual world, and the circle represents the *Area of Interest* (AOI) of the center node.

1.4.2 Previous approaches

Scalability for NVE generally concerns with whether the system can accommodate a large number of simultaneous users [Singhal 99]. Various approaches have been taken, and they generally fall into either the *increase resource* or the *reduce consumption* categories:

Increase Resource.

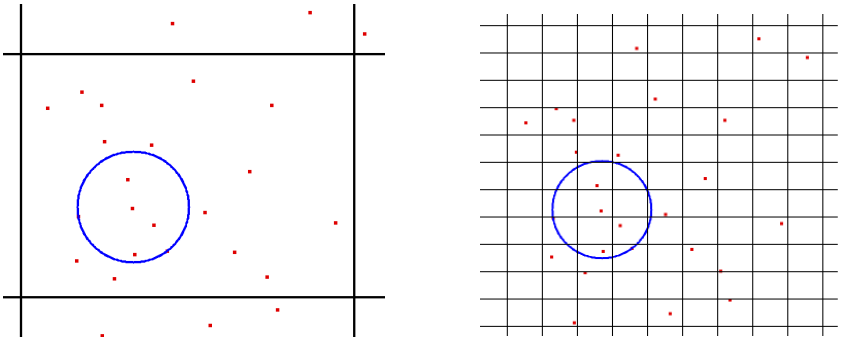
Using multiple servers to host multiple worlds or deploying server-cluster to maintain a single world has become a popular approach, especially for commercial NVEs [Butterfly 03] [Zona 03]. For example, commercial MMOGs are set up with multiple servers for the same game, each serving a pre-determined maximum number of users. When a server is full, it simply denies additional connections. Total number of players can thus be very large. For example, a record of 160,000 concurrent users was reported for the MMOG *Lineage* in 2002 in Taiwan [NCSoft 04]. However, users between different servers may not interact, and some systems do not even share user profiles (so users need to create separate accounts on each of the servers). *Server-cluster* [Funkhouser 95], on the other hand, divides the virtual world into

regions or *zones*, and supports what appears to users as a single coherent world. Since server-cluster offers good scalability with tight, centralized controls for account and security management, it has become the trend for building large-scale NVEs. However, server-centered approach requires large investment in server-side bandwidth, hardware, and maintenance, which creates significant entry barriers for potential NVE developers.

Decrease Consumption.

The central theme to this approach is *interest management* [Morse 00]. While other techniques to economize bandwidth exist, such as packet compression or aggregation [Singhal 96], we consider interest management more relevant. Messages are generated by user actions (for example, moving to a particular location) and are communicated among nodes to maintain consistency. However, if messages are sent to all users in the system, the amount of transmission and processing grows at $O(n^2)$, which is clearly not scalable. Real-world observation tells us that each individual only has a limited visibility or "sphere of interaction". In other words, our interest is localized [Morse 00]. Interest management therefore deals with relevant information filtering, to reduce network resource consumption while maintaining adequate interactivity. Early NVEs did not have interest management, and were set up by hosts broadcasting messages in the same LAN [Miller 95]. To provide interest management, later systems adopt the client-server model, where clients send messages to the server, which acts as *interest manager* and sends back filtered messages (i.e. the messages that are necessary to maintain synchronization for the client). Interest management can be based on various criteria. It can be distance-based (by geography), class-based (by object or user attributes), or some combination of both [Morse 00]. The concept of AOI is thus central in distance-based filtering. Only messages generated within AOI are relevant to the user, and *AOI-radius* becomes the filtering criterion.

A common technique in interest management is to divide the world into various *regions*. Each user only receives messages (position update or interaction message) from relevant regions. This can be done by server-side message filtering, or via network support such as multicast [Macedonia 95-1]. However, region size can be difficult to determine (see Figure 2). If it is larger than AOI, irrelevant messages are received; while if it is smaller than AOI, maintaining regions could be inefficient (e.g. subscribing to too many multicast address). Ideally, region size and shape would be dynamically adjustable to coincide with a user's AOI. The real challenge then is to create *individualized* region that moves with the user.



(a) (b)

Figure 2: Difficulty in choosing region size

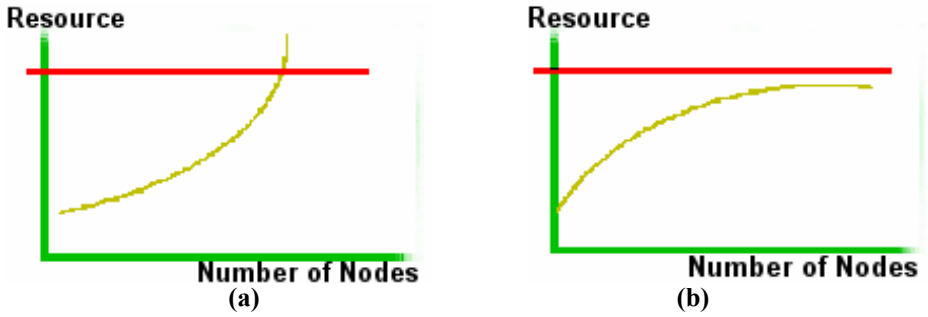
(a) AOI is smaller than region. (b) AOI is larger than region.

1.4.3 Scalability analysis

We may also look at the scalability problem from a resource-limitation perspective. Two main types of resource in NVE systems are the processing (CPU) and network (bandwidth) resource. As bandwidth is usually the limiting factor to NVE scalability in real systems, we will focus our discussions on network resource.

In any type of expandable system (centralized or distributed), there are always some limiting components that pose as the bottleneck. If resource runs out at this component, then the system will cease to be scalable. For example, in a client-server system, the server is usually the limiting component. For distributed system such as P2P, the limiting component is the first node that exhausts its bandwidth capacity.

We may use graphs to visualize scalable and non-scalable systems. Resource consumed at the limiting component usually grows with the size of the system. If the consumption increases without limitation, at some point it will exceed the resource limit, making the component unable to accommodate more nodes (see Figure 3a). On the other hand, if consumption growth may slow down and level off before reaching the resource limit, then the system can continue to scale up (see Figure 3b).



(a) (b)

Figure 3: Scalability analysis

X-axis denotes the number of nodes; y-axis denotes resource consumption at the limiting component; top horizontal line indicates the resource limit.
(a) Non-scalable system. (b) Scalable system.

1.5 A note on consistency

The concept of *consistency* is defined differently in different fields of computer science. Both NVE and P2P assign different meanings to the term "consistency". It is therefore important to first define what we mean by *consistent* in the context of P2P-based NVE.

Consistency in NVE generally refers to the idea of *event* or *state synchronization* (or simply, *event consistency*): whether various participants of the simulation see the same events happening, and whether the events occur in the same order. Zhou *et al.* [Zhou 04] classifies NVE event consistency into two main groups:

- *Casual order consistency*: Events must happen in the same order as they occur, as humans have deeply-rooted concept about the logical order for event occurrence. An example is that we would expect to see canon-firing before an explosion. If we see explosion (the effect) happens before the firing (the cause) then we would feel something "weird" is going on.

- *Time-space consistency*: In an NVE system, messages are sent to notify for position updates. However, due to network *latency* and *clock asynchrony* (i.e. unsynchronized clocks) on various computers, it is possible for hosts to receive updates at different times and interpret the order of event occurrence differently (thus also display the entity positions differently). Inconsistencies therefore could occur for entity positions at a given logical time.

Note that casual order consistency and time-space consistency are not necessarily related to each other (i.e. it is possible to preserve casual order consistency but violate time-space consistency). This problem is particularly evident when predications are used in place of missing update messages.

On the other hand, the concept of consistency in P2P generally refers to what may be called *topology consistency*, which is whether each node in the P2P system holds consistent views of the parts of the network they share (note that each node only maintains a local view of the complete topology). Topology consistency is a common issue for all P2P applications, which also include P2P file-sharing and *distributed hash table* (DHT, described later in section 2.2.2) applications. On the other hand, event consistency is particular to NVE applications.

Consistency in NVE and in P2P, therefore, refers to different concepts, and one has to be careful to specify which consistency is in discussion, in the context of a P2P-based NVE. It should be noted that in P2P NVE, the two types of consistency are related in the sense that topology consistency is a prerequisite for event consistency. Also, if entity position is treated as a type of state update, then topology consistency can be seen as a subset of event consistency.

We will distinguish between the two types of consistencies by referring to them as *event consistency* (in the NVE sense of synchronization) and *topology consistency* (in the P2P sense). This thesis focuses on topology consistency, event consistency is not currently considered.

2. Related Work

As our study attempts to solve the scalability problem in NVE with P2P architecture, we will now survey existing NVE systems, P2P systems, and some of the recently proposed P2P NVE systems. We will focus on their unique characteristics and how scalability is addressed in these systems.

2.1 Networked Virtual Environment (NVE) systems

SIMNET

SIMNET (simulator networking) [Miller 95] was the original large-scale networked virtual environment, developed jointly by the U.S. Army and Defense Advanced Research Project Agency (DARPA) between 1983 and 1990. The goal was to create a large number of low-cost simulators suitable for combat training. It was set up in a LAN environment, with each node broadcasting event updates to all other nodes. Scalability of 850 simulated objects was achieved in one exercise [Singhal 99]. However, LAN bandwidth would be saturated at a rate of $O(n^2)$, making scalability expensive and costly to maintain.

NPSNET

NPSNET [Zyda 92, Macedonia 95-2] is the long-standing academic NVE project at U.S. Navel Postgraduate School (NPS) that began in 1987. Like SIMNET, its goal is also to develop virtual environment technologies suitable for military purposes. The project has a comprehensive scope that addresses many NVE-related research topics such as scalability, extensibility, interoperability, visualizations, computer-controlled agents, and human-computer interactions.

The main contribution from the NPSNET project on scalability is the proposal to use *multicast* as a filtering infrastructure for large-scale NVE [Macedonia 95-1]. The idea is to partition the virtual environment into many fixed-size *regions*, each assigned a unique multicast address. As entities (nodes) move inside the environment, they would subscribe to various multicast addresses that correspond to regions within the node's AOI. This way the amount of message received will be reduced significantly to those within the current area of interest (see Figure 4).

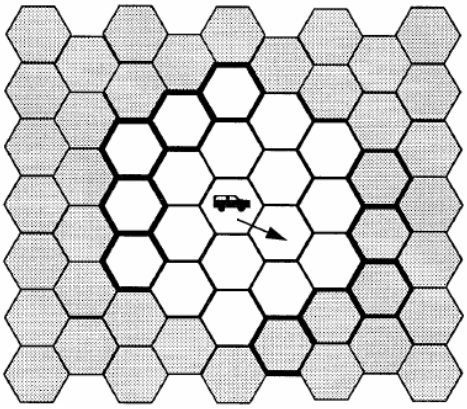


Figure 4: NPSNET region division (source: [Macedonia 95-2])

Each hexagon cell represents a region associated with a multicast address. Entities subscribe to all the cells that are within its Area of Interest (AOI, indicated by the white cells).

The main problem with the multicast approach is that it has not been widely deployed on Internet, and with fixed-size region, there is always the problem of determining the appropriate region size (refer back to Figure 2 in section 1.4.2). If too many nodes are present in the same region, then *crowding* could cripple the system (see section 3.2.5).

RING

To address the resource limitation on a single server, Funkhouser proposed RING in 1995 [Funkhouser 95]. RING was the first attempt in using multiple servers to host a single NVE, laying the foundation for future variants of the *server-cluster* approach (see Figure 5). With multiple servers, each manages a pre-specified partition (a *region*) of the virtual environment, computational and bandwidth load are effectively spread out across the servers (assuming users are distributed evenly in the virtual environment). To facilitate user transfer when they move across different regions, each server would maintain contacts with one another over a high-speed LAN.

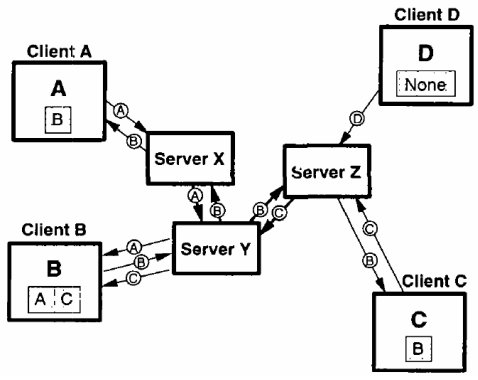


Figure 5: Server-cluster architecture (source: [Funkhouser 95])

Server-cluster has become the practical solution to address scalability to date. However, with high system complexity and maintenance cost (a large amount of bandwidth is required at the server-side to support many users, which could be costly), only organizations with sufficient budget and resource could afford it.

MiMaze

To address the inherent limitation of server-side network resource, Diot *et al.* proposed a fully-distributed architecture on multicast Internet (i.e. the MBone infrastructure) to improve scalability and responsiveness of NVE systems [Diot 99]. The main contribution of MiMaze, however, was not a demonstration of scalability, but the experimental studies to maintain *event consistency* in a distributed environment by using *bucket synchronization* (see section 5.3.3). Although scalability was not focused in the study (MiMaze only uses a single multicast address, with around 25 nodes in the experiment), bucket synchronization may be useful and applicable to distributed system such as P2P.

Massively Multiplayer Online Games (MMOG)

In mid-1990s, commercial companies began to launch games that allow thousands of players to participate in the same virtual universes. Ultima Online [UO], Everquest [EQ], Asheron's Call [AC], and Lineage [Lineage] are a few of the most well-known online games. Current generation MMOG adopts the server-cluster architecture, where each server manages one *region* or *zone*. When users move between zones, user profile is transferred from one server to another, causing a temporary delay. Asheron's Call was the first MMOG to provide *zoneless transition* where the user would not experience delay when transferring to a different zone. It is likely achieved by pre-fetching user data as the user approaches the zone border.

Each server in a cluster can accommodate between a few hundreds to a few thousand users, allowing the total number of users in a single game to exceed 10,000. As MMOG are for commercial purpose, security is of strong concern to ensure fairness. Users usually do not have control over the calculations of game states. Instead, they send in event updates to the server, where the consequence of action commands are calculated and validated. The results are then sent to notify other users who are affected. As client machines may be malicious and cannot be trusted, all player data are stored at server-side databases. Thus, besides message filtering, the server also validates actions, detects collisions, and manages persistent data.

2.2 Peer-to-peer (P2P) systems

Peer-to-peer systems can be described as "distributed systems without any centralized control or hierarchical organization, in which each node runs software with equivalent functionality" [Stoica 03]. Many P2P systems have emerged in recent years, which include distributed processing, distributed file sharing, distributed hash table (DHT), among others. Compared to client-server architecture, P2P offers certain desirable traits:

- **Scalability**: as nodes join the system, they also bring in resource instead of simply consume existing resource. The distributed nature also may prevent any one node from overloading and becoming the bottleneck of scalability.

- **Affordability**: most P2P systems can be constructed using commodity hardware without requiring expensive servers or server-side bandwidth.

- **Fault-tolerance**: as there is no centralized authority in the system, true P2P systems do not have the single-point of failure problem in client-server architecture. The system may still function even if some nodes fail.

Peer-to-peer systems are constructed by connecting various computers (nodes) in a mesh-like fashion (see Figure 6) to form a virtual network on top of the physical Internet. The term *overlay network* is thus used to describe peer-to-peer systems (we will use the term *peer-to-peer* and *overlay network* interchangeably in this thesis).

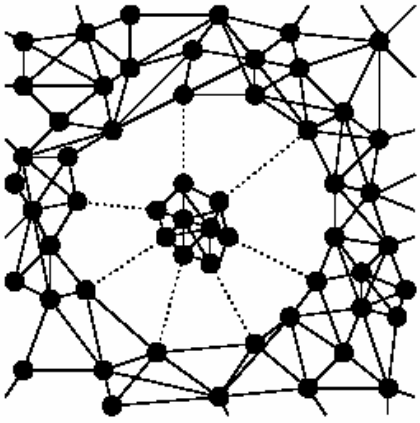


Figure 6: A peer-to-peer *overlay* network (source: [Keller 03])

We will next briefly describe two types of the popular P2P systems: file-sharing and Distributed Hash Table (DHT).

2.2.1 File-sharing systems

Napster

Napster [Napster] was the first widely used P2P file-sharing network that allows the sharing of music files. Although file-transfer is done between peer computers, file-discovery requires querying a centralized server. The server maintains a directory of the locations of each file currently shared on the P2P network. As centralized servers are used, Napster is not fully-distributed and thus faces some of the same problems as client-server: inherent upper limit to server-side resource (e.g. limited scalability) and a single-point of failure.

Gnutella

Gnutella [Gnutella] attempts to address Napster's single-point of failure problem by adopting a fully-distributed architecture. Each node in the Gnutella network maintains a contact list of other nodes. When users try to find a particular file, the search request is broadcasted (on the overlay network) to its list of contacts. If the file is not found, the contacts would forward the request to their own list of contact nodes. The search request thus spreads through the network in a ripple-like fashion. To avoid over-flooding the network, each request is attached a time-to-live (TTL) value, and is dropped when the value reaches 0 (e.g. the search is being given up).

Although Gnutella avoids having a single-point of failure, its broadcasting nature is time-consuming and wasteful. It trades efficiency with fault-tolerance capability, yet still does not pose to be truly scalable.

2.2.2 Distributed Hash Table (DHT)

To address the non-scalable nature of existing file-sharing P2P, a number of academic P2P overlays have been proposed in recent years: CAN [Ratnasamy 01], Chord [Stoica 03], Pastry [Rowstron 01], Tapestry [Zhao 04] and Hypercast [Liebeherr 02], to name a few. These overlay networks mainly deal with setting up a distributed hash table (DHT), which provides the function of mapping any given key to a node, and allows for efficient, robust content-lookup (which are required for file-sharing). However, virtual environments have different requirements, and DHT would not address effectively the problems in NVE. While DHT may be efficient at *querying* the location of a piece of data, it does not support frequent message exchange (or updates) which is necessary to provide interactivity. While it is possible to build NVE on top of DHT, as proposed by Knutsson *et al*. [Knutsson 04], it would incur certain overhead and latency (to be discussed next in section 2.3.1).

2.3 P2P-based NVE

Common questions to all P2P networks are: correct *topology maintenance* and efficient *content retrieval*. Since a single node has no knowledge of global topology or content location [Kung 01], these two issues are the central challenges to any P2P design. For topology maintenance, two aspects must be considered: whether it is *fully-connected* (described by Keller and Simon as the *Global Connectivity* property [Keller 03]) and whether all nodes have a *consistent view* of the topology (i.e. whether the local topology views of all nodes agree with each other and may merge into a globally consistent topology. A similar concept was described by Keller and Simon as *Local Awareness* [Keller 03], which indicates whether a node is aware of all its AOI-neighbors). Unlike file-sharing P2P, where the desired file may be located on any node according to user preference, in P2P NVE the desired content is more specific — messages generated by other users within the AOI. If only such messages are received, then message flow is managed optimally. The "content discovery problem" in P2P systems, therefore, translates naturally to a *neighbor discovery problem* in P2P NVE. Efforts to apply P2P on NVE began around 2002, with publications appearing in 2004. Below we describe three of the main proposals to the neighbor discovery problem.

2.3.1 SimMud (University of Pennsylvania)

Knutsson *et al*. describe P2P support for Massively Multiplayer Games by using Pastry and Scribe, a P2P overlay and its associated simulated multicast [Knutsson 04]. The virtual world is divided into regions of fixed-size (see Figure 7). Each region is managed by a promoted super-node called *coordinator*, which serves as the root of a multicast tree. Users inside the same region subscribe to the root node to receive updates from other users, so neighbors are discovered via the coordinator. Coordinators maintain links with each other, facilitating user transition to other regions

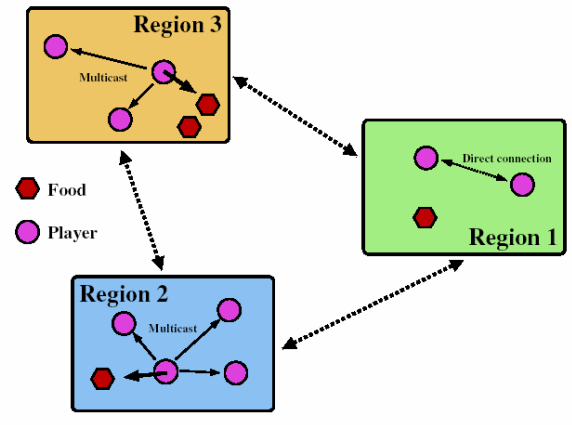


Figure 7: Fixed-size regions in SimMud (source: [Knutsson 04])

However, since fixed region size does not reflect user AOI, users cannot see across regions. If users decide to listen to more regions, as suggested in the paper, irrelevant messages beyond AOI will be received. If too many users crowd inside the same region, then the *coordinator* may be overloaded. A more serious problem is the latency penalty incurred by using the P2P overlay. As the overlay is not designed around the concept of AOI, message updates may need to be relayed by other nodes (less than 6 hops in most cases, but exceed 50 in other cases – a delay up to several seconds. Note that this is "virtual hops" on the P2P overlay, so it translates to more hops at the physical level). In short, the architecture does not fully utilize the true power of P2P — *direct connections*.

2.3.2 Neighbor-list exchange (University of Tokyo)

Kawahara *et al*. describes a fully-distributed scheme where each user keeps track of a fixed number of nearest neighbors [Kawahara 04]. Nodes constantly exchange neighbor list with their own neighbors (see Figure 8). After sorting through the list by proximity, each node may learn of new nodes and update its connection relationship (to reflect current topology).

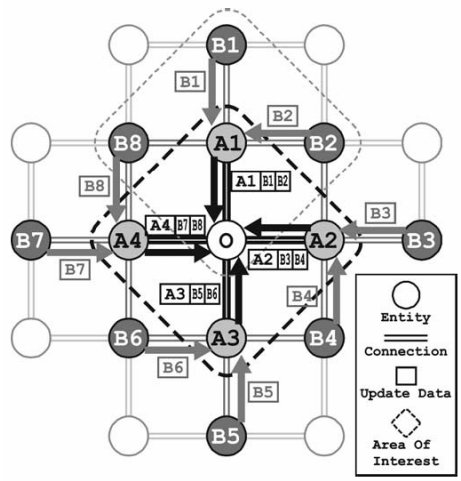


Figure 8: Neighbor-list exchange scheme (source: [Kawahara 04])
Neighbors (A1, A2, A3, A4) exchange neighbor list with the center node.

In this approach, direct links are maintained between neighbors, so transmission latency is minimized (e.g. messages are exchanged directly, not relayed through intermediate nodes like SimMud). However, constant exchange of neighbor list incurs network overhead (if 10 nearest neighbors are kept, one exchange requires receiving updates of 10x10 nodes). The more serious problem is keeping the topology fully-connected. Since only a finite number of nearest neighbors are maintained, groups of users may lose contact to each other if separated by a large distance. The underlying overlay can thus separate into isolated parts (an *overlay partition*) [Kawahara 04].

2.3.3 Solipsis (France Telecom R&D)

Solipsis [Keller 02, Keller 03] is also a fully-distributed system, where each node attempts to link with all the nodes within its AOI. Neighboring nodes serve as the "watchmen" for any approaching foreign nodes. Neighbor discovery is done by notification from known neighbors.

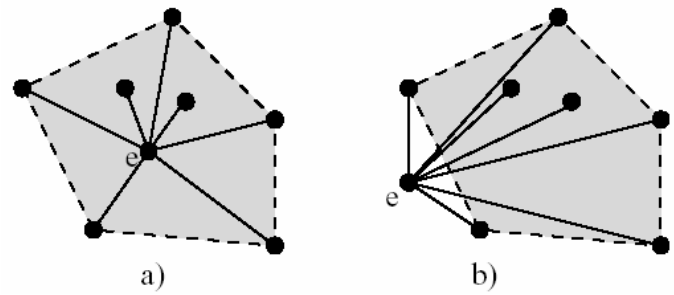
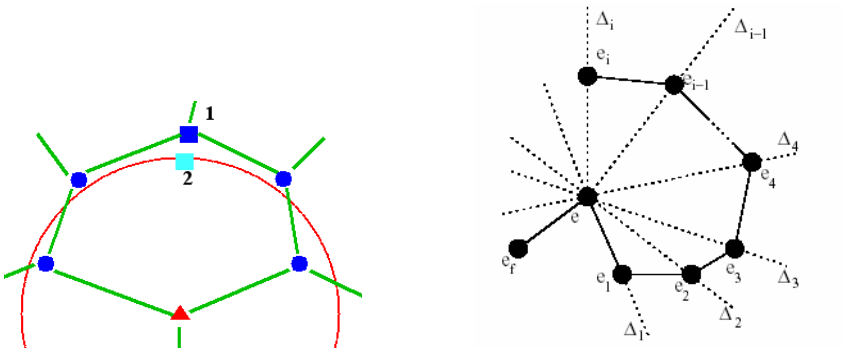


Figure 9: Design of Solipsis (source: [Keller 03])

Solipsis requires that all nodes are within a convex hull formed by neighboring nodes. In the figure above, (a) matches the requirement while (b) violates the requirement.

Like the neighbor-list exchange scheme, Solipsis also maintains direct links among neighbors (latency is thus minimized). Specifically, it requires that each node be inside a convex hull formed by its neighbors in 2D plane (see Figure 9a). This way the topology is guaranteed to be fully connected (i.e. *Global Connectivity* is kept). However, inconsistent topology may happen during normal operation occasionally, since an incoming node may be unknown to directly connected-neighbors, proper neighbor discovery is thus not guaranteed (i.e. *Local Awareness* is not kept, see Figure 10a). In some other cases, proper neighbor discovery could be slow as it may require a few queries (see Figure 10b).



(a) **(b)**

Figure 10: Potential discovery problems in Solipsis (source of (b): [Keller 03])

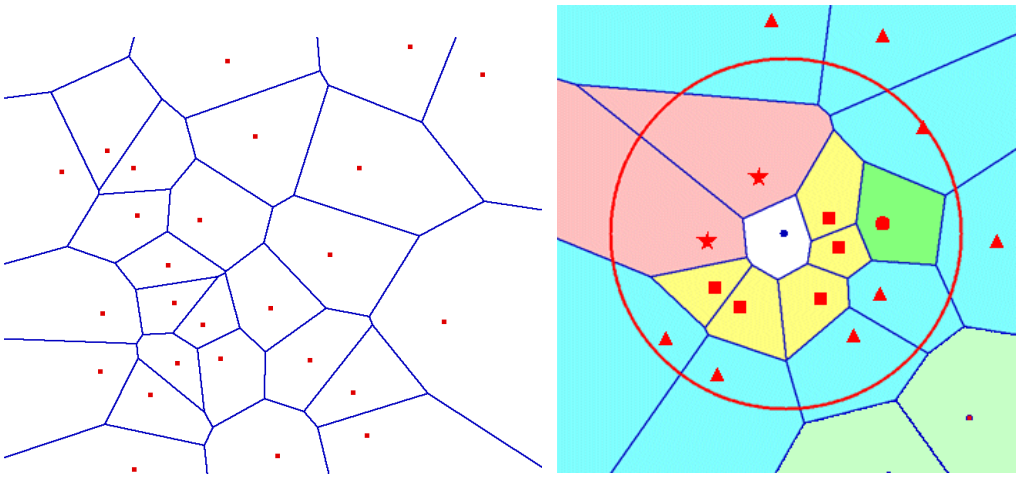
(a) Lines indicate connections. Square node is not discovered as it moves from position 1 to 2. Topology becomes inconsistent even though it is fully-connected and compiles to Solipsis' design. (b) It may take node *e* a few queries before properly discovering node e_i in the north.

3. Voronoi-based P2P NVE

We now describe the design and analysis of our P2P approach, which is based on a well-studied mathematical construct *Voronoi diagram* [Guibas 85, Aurenhammer 91]. We first explain what a Voronoi diagram is, then explain the design of our proposed architecture – *Voronoi-based Overlay Network* (VON). An analysis of VON's design is then provided, including comparisons with other P2P NVE systems and some problems in the current design. We conclude this part with some implementation considerations.

3.1 Voronoi diagram

Given *n* points on a plane, (each point called a *site*), a Voronoi diagram is constructed by partitioning the plane into *n* non-overlapping *regions* that contain exactly one site in each region. A region contains all the points closest to the region's site than to any other site (see Figure 11a). The entire plane is therefore divided into arbitrary sizes in a deterministic way. Voronoi diagram can be used to find the *k*-nearest neighbors of a specific site. By using Voronoi, we can identify *enclosing* and *boundary neighbors* for a given site. *Enclosing neighbors* are defined as sites whose regions share a common edge with the given site's own region. *Boundary neighbors* are defined as the sites whose regions overlap with the site's AOI boundary (see Figure 11b). Note that an enclosing neighbor may also be a boundary neighbor. These properties will help to solve the neighbor discovery problem described earlier.



(a) **(b)**

Figure 11: Voronoi diagram

(a) The dots indicate *sites*, and lines define boundaries for *regions*. (b) Squares (■) represent *enclosing neighbors*; triangles (▲) represent *boundary neighbors*; stars (★) are *both* enclosing and boundary neighbors; circle (●) represents a regular *AOI-neighbor*.

There exist a number of algorithms to construct Voronoi diagrams [Gowda 83, Dwyer 86, Fortune 86]. Most can construct a Voronoi diagram with $O(n \log n)$ time complexity given a number of sites. However, for our purpose, the specific construction method is not the focus of this thesis, and we assume that stable algorithms to construct Voronoi diagram exist and can be readily used.

3.2 System design

3.2.1 Overview

The basic idea of our approach is to let each node construct and maintain a Voronoi diagram, based on the spatial coordinates of neighbors within the node's AOI. Each node keeps P2P connections with all neighbors that constitute the Voronoi sites. Connections are based on spatial relationship in the NVE (not physical network proximity). Each node must minimally maintain all its enclosing neighbors to ensure that topology is fully-connected. In our basic model, we assume that all AOIs are of the same radius (a dynamic-AOI variation is described later in section 3.2.5), and are determined in an application-specific manner by the designer. Although a node only knows a limited number of neighbors, it can learn of other new neighbors with the help of its boundary neighbors. Each peer serves as the "watchman" for one another in discovering approaching neighbors.

When the node moves, position updates are sent to *all* neighbors maintained in the Voronoi. If the receiver of the update is a boundary neighbor (as determined by the sender), an overlap-check is performed. The receiver checks if the mover, with its new AOI, would enter into any of its enclosing neighbors' Voronoi regions. The receiver only notifies the mover if a *new* overlap occurs (i.e. previously non-overlapped region becomes overlapped). Receivers may also notify the mover if the mover has any new enclosing neighbors, to ensure that enclosing neighbors are maintained by each node at all times. The moving node may thus be aware of potentially visible neighbors outside its AOI with minimal network overhead (because no query messages are involved in neighbor discovery, and messages are exchanged only when necessary. In a way, the queries are embedded in the position updates). In case of a node leave or failure, its neighbors simply update their Voronoi after detection (through a loss of TCP connection or inactivity timeout). If the leaving node is a boundary neighbor, replacements are notified by the still-connected boundary neighbors during the next position update.

We will next describe the procedures for node joining, moving and leaving in VON. The emphasis is to maintain P2P topology consistency in a message-efficient manner.

Before we show the actual procedures, Table 1 describes the notation and definition for the pseudocode that describes the implementation details. Pseudocode for the JOIN, MOVE, and LEAVE procedures are given in Figure 13, Figure 16, and Figure 18, after the textual descriptions for each procedure.

Table 1: Definition and notation for the pseudocode

Variables for node *n*	**Description**
id	node *n*'s unique ID
aoi	node *n*'s AOI-radius
pos	node *n*'s coordinates (x and y)
neighbors	a list of the neighbors that node *n* maintains

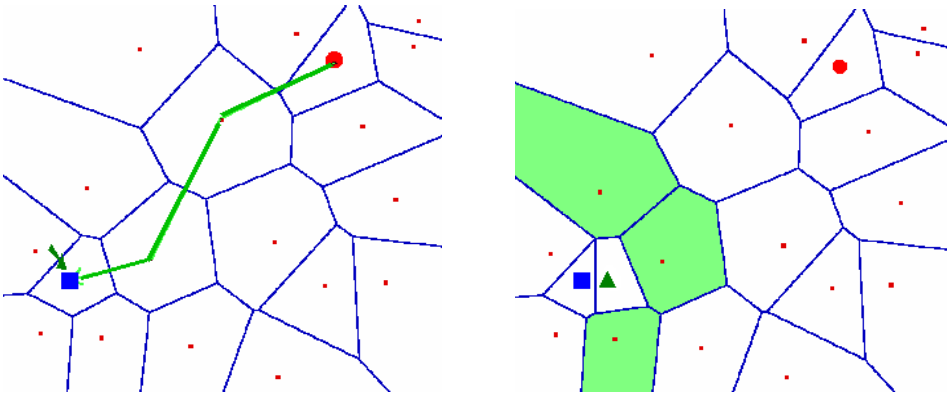
Basic functions for *n*	**Description**	**Returns**
contains(*pos*)	if *pos* is within *n*'s Voronoi region	*true/false*
overlaps(*node*)	if *n*'s AOI overlaps with *node*'s Voronoi region	*true/false*
closest(*pos*)	the neighbor with shortest distance to *pos*	*node_id*
is_EN(*id*)	if node *id* is an enclosing neighbor of *n*	*true/false*
is_BN(*id*)	if node *id* is a boundary neighbor of *n*	*true/false*
knows(*id*)	if node *id* is a currently known neighbor of *n*	*true/false*
insert(*node*)	store *node*'s *id*, *aoi*, and *pos* to *neighbors*	*none*
delete(*id*)	remove the node *id* from *neighbors*	*none*
update(*node*)	updates a node's *aoi* and *pos*	*none*

Note 1: in the following pseudocode, if node *n* is sent as a parameter, only its *id*, *aoi*, *pos* are sent, its *neighbors* is never sent in order to conserve bandwidth.

Note 2: the basic functions **insert**, **delete**, and **update** maintain the Voronoi diagram for a particular node *n*.

3.2.2 JOIN procedure

1. *Joining node* contacts the *gateway server* for a unique ID.
2. *Joining node* sends a join request with its own coordinates to any existing node (which can be the *gateway server*).
3. Join request is forwarded to the *acceptor region* (e.g. the region that contains the joiner's coordinates) via neighboring nodes with greedy forward (see Figure 12a).
4. *Acceptor node* sends back a complete list of its own neighbors.
5. *Joining node* contacts each neighbor on the list.
6. *Joining node* builds up a new Voronoi while the contacted nodes update their Voronoi to accommodate the *joining node* (see Figure 12b).



(a) **(b)**

Figure 12: JOIN procedure

(a) Forward of join request. Circle is *gateway server*. Arrow indicates the *acceptor node.* (b) Triangle (▲) is the new node, shaded regions are neighbors affected by join. Note that the effect is localized.

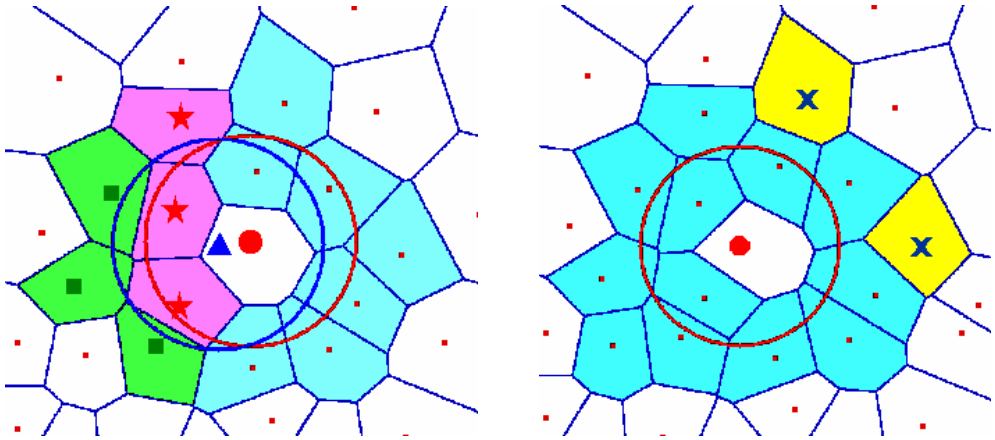
```
// enters p2p overlay given an initial gateway
n.join(gateway)
    id = gateway.assign_id();
    gateway.query(n);

// send my list of neighbors if I am the acceptor
// otherwise forward the query request
n.query(node)
    if(contains(node.pos))
        node.notify(neighbors);
    else
        n' = closest(node.pos);
        n'.query(node);
```

```
// unique-ID assignment (gateway only)
n.assign_id()
    id_counter++;
    return id_counter;

// notify a node of a list of nodes it should know
n.notify(node_list)
    for each node in node_list
        if(contains(node.pos))
            insert(node);
            node.insert(n);
```

Figure 13: Pseudocode for the JOIN procedure

3.2.3 MOVE procedure

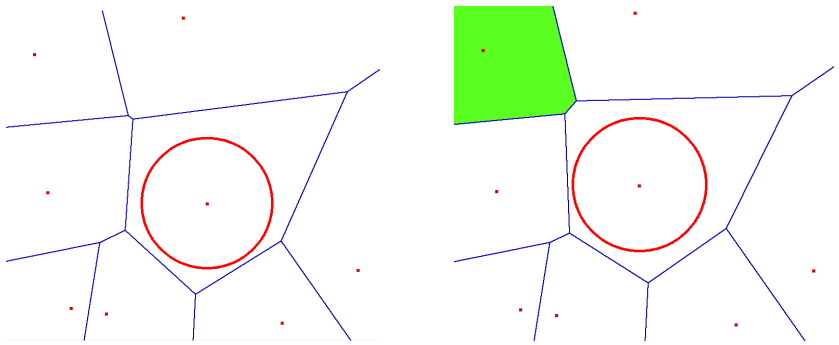
1. *Moving node* sends position coordinates to all neighbors (i.e. *boundary*, *enclosing*, and *AOI-neighbors*). Messages to *boundary neighbors* are specifically marked.
2. *Boundary neighbor* checks if the *moving node*'s new AOI overlaps with any of its *enclosing-neighbors'* Voronoi regions (see Figure 14a) or if the *moving node* comes to contact with any new *enclosing neighbor* (see Figure 15). If so then it sends a notification.
3. If a new neighbor is found, the *moving node* connects to it.
4. *Moving node* disconnects any *boundary neighbors* whose Voronoi region no longer overlaps with its AOI (see Figure 14b).



(a) (b)

Figure 14: MOVE procedure

(a) Triangle (▲) indicates the intended new position. Squares (■) are new neighbors about to be discovered. Stars (★) are the boundary neighbors (b) Crosses (╳) are the neighbors no longer overlap with AOI, therefore are disconnected.



(a) (b)

Figure 15: MOVE procedure (notification due to new enclosing neighbor)

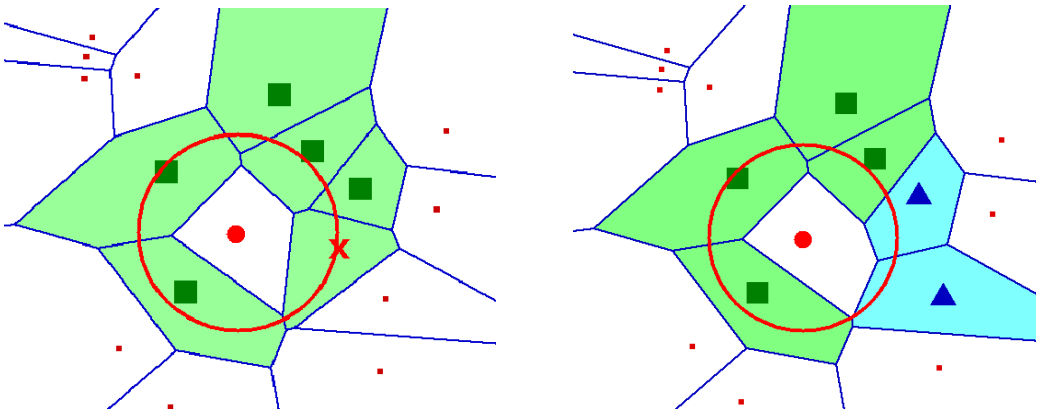
(a) The moving node's AOI does not overlap with any of its enclosing neighbors, yet connections are still maintained due to protocol requirement. (b) The moving node is notified of a new enclosing neighbor (the shaded region) by its existing enclosing neighbors. Note that its AOI does not overlap with the new neighbor.

```
// move to a new position (specified by new_pos)
n.move(new_pos)
   pos = new_pos;
   adjust_aoi();      // explained later in section 3.2.5
   remove_nonoverlapped();
   for each node in neighbors
      if(is_BN(node))
         node.has_moved(n, true);
      else
         node.has_moved(n, false);

// remove neighbors that no longer overlaps with n's AOI
n.remove_nonoverlapped()
   for each node in neighbors
      if(overlaps(node) is false)
         remove(node);
         node.remove(n);

// tell a neighbor of my new position, this neighbor will check for neighbor discovery
n.has_moved(node, notify_flag)
   update(node);
   if(notify_flag is true)
      for each enclosing_neighbor in neighbors
         if( (node.overlaps(enclosing_neighbor) or node.is_EN(enclosing_neighbor))
              and node.knows(enclosing_neighbor) is false
            node.notify(enclosing_neighbor);
```

Figure 16: Pseudocode for the MOVE procedure

3.2.4 LEAVE procedure

1. *Leaving node* simply disconnects (there is no distinction between proper and abnormal departures from the overlay network).
2. Neighboring nodes affected by the disconnection update their Voronoi. If a *boundary neighbor* leaves, replacements may be learned via still-connected *boundary neighbors* (see Figure 17).



(a) **(b)**

Figure 17: LEAVE procedure

(a) Before node leave, cross (⨉) is the leaving node. (b) After node leave, triangles (▲) are the new boundary neighbors discovered with help of still-connected boundary neighbors (Squares ■).

```
// let all my neighbors update their Voronoi by removing me
n.leave()
   for each node in neighbors
      node.remove(n);
```

Figure 18: Pseudocode for the LEAVE procedure

3.2.5 Dynamic AOI adjustments

In real-world applications, objects or events of interest may cause certain parts of the map to have a high density of users (i.e. a *crowding* situation, see Figure 19. It is similar to the *flash crowd effect* in websites). The fixed-size radius of AOI in our basic model will not accommodate such situation, and nodes may become overwhelmed by connections or messages as the density of neighbors increase within their AOI.

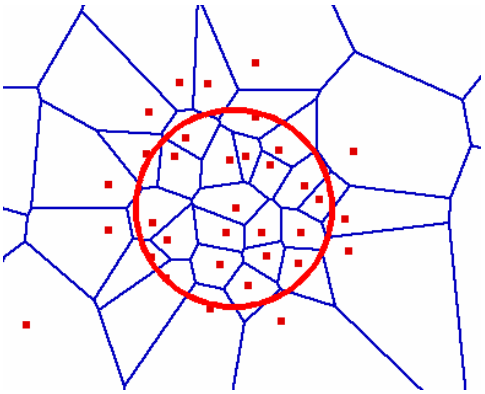


Figure 19: Example of a crowding situation

To address this problem, we propose an enhancement to the basic algorithm by adjusting AOI-radius dynamically in real-time. We specify for each node, a maximum number of allowable connections, then adjust the AOI-radius to ensure that this limit does not exceed by the actual number of connections. The adjustment conditions are:

- **AOI-radius decrease condition**
  - Number of currently-connected neighbors exceeds the maximum number of allowable neighbors
- **AOI-radius increase condition**
  - Number of currently-connected neighbor does not exceed maximum number of allowable neighbors.
  - Current AOI-radius is less than the preferred (e.g. the initial) radius.

These two simple rules will help to maintain a balance on the connection size for each node in the P2P network (see Figure 20 for the pseudocode).

```
n.adjust_aoi()
   if(sizeof(neighbors) greater than connection_limit)
      decrease aoi;
   else if(aoi less than preferred_aoi_radius)
      increase aoi;
```

Figure 20: Pseudocode for dynamic-AOI adjustments

3.2.6 The VON protocol

The above procedures are realized as a set of protocols. The basic data structure is shown in Table 2. Table 3 describes the message commands and their functions.

Table 2: The VON data structure
(Number inside bracket indicates the number of bytes used)

Data type	Components	Purpose
ID	Unique-id (4)	To identify each node uniquely
POS	x (4), y (4)	To identify a position in virtual environment
ADDR	IP-address (4), port (4)	The address and listen port of a node
AOI	AOI-radius (4)	The AOI-radius of a particular node

Table 3: The VON protocol
(Parameters inside the square bracket indicate that they are sent as a list.)

Command	Parameters	Return	Function
ID	ID		Unique ID assigned by gateway server
GREET	ID		Allow a connection-accepting node to understand the identity of a remote node
QUERY	ID POS ADDR	**NODE** (acceptor's own neighbors)	Sent by the joining node to indicate its initial position, in order to find the acceptor region. This command will be forwarded if the Voronoi region of the processing node does not contain the entry point. The address is used by the acceptor to contact the joining node.
HELLO	POS AOI ADDR	**HELLO**	Handshake by two nodes that try to establish connections with each other. (sent after **GREET**)
EN	[ID]	**NODE** (missing neighbors)	A list of unique-IDs of enclosing neighbors. Used after **HELLO** to check for missing neighbors (see section 3.2.7)
MOVE	POS AOI		Notifying connected-neighbor of the current position and AOI-radius
MOVE_B **MOVE_BD**	POS AOI	**NODE** (new neighbors)	Same as **MOVE**, but sent to boundary neighbors, which will then check for potential neighbor discovery. See section 3.2.7 for a description of **MOVE_BD**.
NODE	[ID POS ADDR]		Information about a particular node, for connection-establishing purpose.

3.2.7 Procedure enhancements

There are some situations where the above procedures do not work as correctly as expected, therefore some enhancements to the basic procedures are required. We discuss some of the enhancements below.

JOIN procedure enhancement

A fine-point in the JOIN procedure deserves some emphasis. The neighbor sets of the acceptor and the joining node (the *joiner*) may not be exactly the same in some cases (see Figure 21), and will cause incomplete neighbor discovery for the joiner.

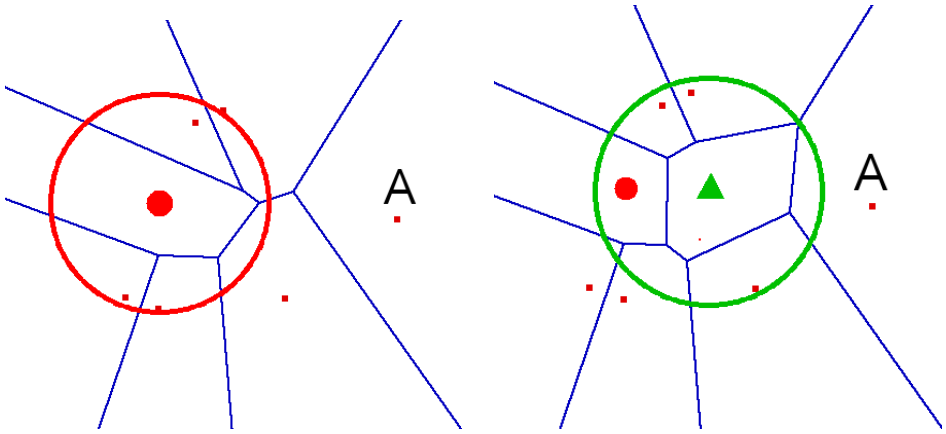


Figure 21: Dissimilar neighbor sets for acceptor and joiner

(a) Node A does not appear on the neighbor list for the acceptor node (circle). (b) However, Node A is a neighbor for the joining node (triangle). The neighbor list returned by the acceptor is not complete for the joining node.

To remedy this situation, we add an additional procedure when nodes initiate connections to newly discovered neighbors. It will send a list of the enclosing neighbors of the node to be contacted, so that the contacted node may check for any missing nodes from its enclosing neighbor set. This is presented as the *EN command* in the VON protocol (see Table 3 in section 3.2.6). This additional procedure will add some overhead to the VON protocol. However, the frequency is limited to only when establishing a new connection. Also, it will provide the additional benefit of fixing inconsistent local views of the topology among neighbors. If we assume each node knows its enclosing neighbors correctly, then this "fixing mechanism" will provide robustness to topology maintenance.

Dynamic AOI adjustments enhancement

When a node decides to shrink its AOI-radius, it may disconnect certain nodes that still consider the node as their AOI neighbors. This will cause inconsistencies in those other nodes with larger AOI-radii (see Figure 22)

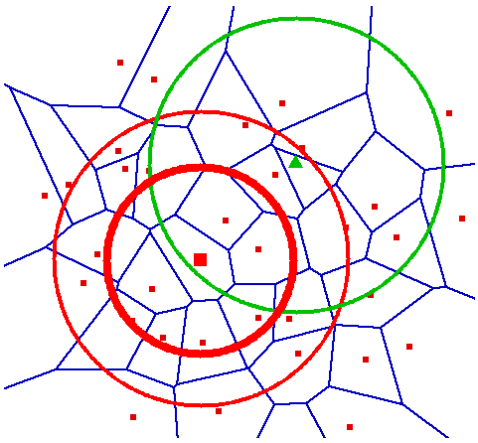


Figure 22: Topology inconsistency due to dynamic-AOI adjustment

The square node shrinks its AOI-radius (from the thinner to the thicker AOI) and disconnects the triangle node. The triangle node then has an incomplete/incorrect view of the topology.

To avoid inconsistent views of the topology, one possible solution is for the affected (i.e. disconnected) nodes to immediately shrink their AOI-radii, so that the initial AOI-shrinking node (i.e. disconnecting node) falls outside the AOI of the affected nodes. Topology consistency of the nodes being disconnected therefore will not be adversely affected.

We introduce the MOVE_BD message to the VON protocol as an enhancement to the MOVE_B message (which is used to notify boundary neighbors of position updates, see section 3.2.6). MOVE_BD is sent to those boundary neighbors that will be disconnected in the next time-step. Receivers of the message can therefore anticipate a possible future disconnection and shrink their AOI-radii preventively.

This remedy, however, will lower the number of visible nodes of the affected node when it has not yet reached its connection limit. It may or may not be desirable depending on the application type. A better solution is the topic for future research.

3.3 Design analysis

We will now analyze how VON matches the six criteria for NVE outlined earlier (see Introduction and section 1.5), in order to emphasize its specific characteristics and compare against other approaches. Currently identified problems with the Voronoi-based approach are then discussed.

3.3.1 Criteria for NVE construction

Topology consistency

We will discuss topology consistency in two aspects: whether the topology is *fully-connected* (e.g. *Global Connectivity* is kept) and whether each node shares *consistent views* of the world (e.g. *Local Awareness* is achieved properly).

Global Connectivity is kept as long as each node properly maintains its enclosing neighbor set. An intuitive explanation is that enclosing neighbors completely cover a node, therefore, if the set is kept, a node will always maintain links with at least *some* nodes in the P2P overlay. In fact, overlay partition itself indicates that enclosing neighbors are not kept properly.

Consistent views (or proper Local Awareness) are maintained as long as neighbor discovery is complete. This will happen if the procedures for JOIN/MOVE/LEAVE are followed correctly. The reason is that the combined Voronoi regions of all AOI and boundary neighbors, by definition, fully cover a node's AOI boundary. Any neighbor further away cannot enter the AOI-radius without first colliding with the boundary neighbor set (and become the boundary neighbors' enclosing neighbor). If the MOVE procedure is followed properly, then a node would be notified for all approaching unknown neighbors by its boundary neighbors (note that these unknown neighbors necessarily will become the enclosing neighbors of the boundary neighbors, if all boundary neighbors properly maintain their enclosing neighbor sets). The completeness of neighbor discovery is thus guaranteed.

However, neighbor discovery may become incomplete in the following cases:

1) When the acceptor and the joiner have different sets of neighbors.
2) When a node is disconnected due to AOI-radius shrinking.
3) When nodes are moving too fast.

Case 1) and 2) have been discussed and given solutions in section 3.2.7, also note that case 2) does not happen in the basic model (when AOI-radius is fixed). We consider case 3) an implementation issue and will discuss it in section 3.4.2. As will be shown in next part, the current design of VON is able to maintain a high-level of topology consistency.

Responsiveness

Responsiveness in an NVE system is influenced by many factors: hardware systems, software architecture, graphical subsystem, network subsystem, simulation complexity, etc. Many elements related to the responsiveness of an actual system are strongly depended on the implementation. Therefore, we only focus on the theoretical aspect of VON's design. In particular, we will define responsiveness in terms of the number of virtual hops a message travels before reaching its destination.

Here it is clear that due to the design of direct-connection between peers, VON achieves a hop-count of 1 for the whole system. Transmission latency is therefore minimized. Note that this hop-count is measured in terms of P2P topology, not the hop-counts of actual physical network links. A message could travel a number of hops physically before reaching its actual destination. However, this is the best scenario achievable in a NVE context, because users are necessarily scattered on the physical network, yet we cannot move them arbitrarily to optimize the network traffics.

Security

Security is an important and serious consideration for many applications of NVE, especially for commercial MMOG. However, it is also an inherently difficult problem in distributed systems such as P2P. As users have full access to the software system and network traffic, they could hack message packets and create modified software that acts maliciously. Some of the security issues for VON include:

1) **Uncooperative hosts**: correct P2P topology requires cooperative nodes to maintain, or neighbor discovery could become incomplete or incorrect. An uncooperative host could send out incorrect discovery information or refuse to notify relevant nodes, making the topology inconsistent.

2) **Game state manipulation**: For the various game states that are stored and exchanged among peers, a cheating node could artificially modify game states to the cheater's advantage.

As we note that security is a difficult problem, yet it is possible to solve other aspects of the P2P NVE first (such as scalability), therefore, we choose not to consider security issues in the present work.

Scalability

VON's design matches the characteristics of scalable systems, namely: resource-growing and decentralized resource-consumption. With dynamic AOI adjustment, a node requires only a fixed amount of network resource to operate. This characteristic allows any number of nodes to join the system as no node can become the "bottleneck" for the system. We will demonstrate VON's scalability with simulation results in section 4.

Persistency

We define persistency as the storage and maintenance of certain game states across user sessions. Persistency is also not considered in the present work. As a result, the applicability of our proposed system is limited. Certain applications that require persistency, such as commercial MMOG, cannot adopt the present architecture. However, there are other works that deal with persistent data storage on P2P network, such as OceanStore [Kubiatowicz 00]. We believe that persistency can be achieved with VON and this will be an important topic for future research (see section 5.3.4).

Reliability

We will define reliability as whether the P2P NVE can sustain node failures and remain functional, and whether the recovery from failure is quick with minimal impact on normal operations. Currently there are two recovery mechanisms in VON:

1) During initial contact to a node, the new node would help to check for missing neighbors (the EN command, see section 3.2.7).
2) As nodes move around, boundary neighbors may help to discover new nodes. This behavior tends to correct inconsistency if at least enclosing neighbors are kept correctly by each node.

In terms of recovery time, if a node suddenly fails, its neighbors will take notice in the next time-step, and would update their own Voronoi accordingly. In most cases this happens in the next time-step and would restore the P2P topology. If the leaving node is a boundary neighbor, potential replacements are learned via still-connected boundary neighbors, topology is thus also restored in the next time-step. We will demonstrate the reliability of VON with simulation results in section 4.3.3 and 4.3.4.

3.3.2 Complexity issues

While the construction complexity for Voronoi is *O(n* log *n)* for the best algorithm, it will not be of concern in practical applications. The reason is that we expect each node only to maintain a limited number of neighboring nodes (less than 50), so an application may still be quite responsive even if it needs to construct Voronoi diagrams on a dynamic and regular basis.

On the other hand, we should note that VON effectively reduces system-wide communication cost to $O(n)$ from the potentially high cost of $O(n^2)$ in broadcast-type communication. As each node maintains only a finite set of neighboring connections, the total communication cost is N x C, where N is the number of nodes, and C is the maximum resource consumption at a particular node.

3.3.3 Comparisons with other systems

Here we will make some comparisons of VON with other types of existing systems (see section 2.3 for descriptions of these systems). We also summarize the comparisons in Table 4.

Table 4: Comparisons of P2P NVE systems

	SimMUD	**Neighbor-list exchange**	**Solipsis**	**VON**
Consistency (Topology)	Supernode	Neighbor-list exchange (partitioning)	Neighbor notify & query (undiscovery)	Neighbor notify (high consistency)
Responsive-ness	High overhead (message relay)	High overhead (list exchange)	Medium overhead (neighbor query)	Low overhead (notify only)
Scalability	Relied on supernode	Fully-distributed	Fully-distributed	Fully-distributed
Reliability	Long up-time	N/A	N/A	Self-organizing

SimMud

SimMud is based on the P2P overlay Pastry and pseudo-multicast Scribe. As Pastry is designed to be scalable, SimMud is also a scalable system. However, its main problem is the additional latency introduced by the "supernodes" (coordinators). In SimMud all messages must first be sent to the coordinator before being dispatched to the affected nodes individually. This creates the following problem:

- Increased load for the coordinator nodes
- Increased latency due to relay by the coordinator nodes
- Increased complexity in design for backup mechanisms of the coordinators

On the other hand, VON would not overload any particular node, as the system is fully-distributed, so no single node bears more responsibility than any other node. Latency is minimized as all peers make direct connections to each other, without any message relay. Also, because there is no super-node, no special back-up or recovery mechanism is needed, making the algorithm relatively simple and straight-forward.

However, by centralizing certain aspects of message delivery, SimMud is able to leverage message compression and aggregation techniques to reduce bandwidth consumption, which is one aspect that VON cannot support.

Neighbor-list exchange

Compared with the approach suggested by Kawahara *et al.,* our approach will not face the problem of overlay partition, as each node must minimally maintain its enclosing neighbor set. Enclosing neighbor set ensures that connections are maintained with nodes in all possible directions. In Kawahara's approach, maintaining a fixed number of nearest neighbor does not prevent losing contacts with neighbors in directions that lack spatially-close nodes.

Another advantage of VON is that topology is maintained without extra message transmission. Neighbor discovery requests are embedded in the normal message traffics of position updates, and notification messages are sent only when an actual neighbor discovery has occurred. This saves bandwidth compared to exchanging neighbor list periodically. In fact, neighbor lists usually contain redundant information that makes sending them a waste to bandwidth.

Solipsis

In terms of design concepts, VON is most similar with Solipsis, where each node maintains a certain number of neighboring nodes (according to some rules), and that neighbor discovery is done by mutual collaboration between neighbors. Both VON and Solipsis also make direct connections among peers, which makes message transmission efficient. Both are also fully-distributed, so do not need to worry about super-node failure or overloading of particular nodes.

The key difference between VON and Solipsis lies in which neighboring nodes are maintained. In Solipsis the rule is that each node must be contained within a convex hull formed by its neighbors (refer back to Figure 9 in section 2.3.3); for VON, the neighbors to keep are the ones whose Voronoi regions overlap with the node's AOI. Note that the enclosing neighbor set automatically constitutes a convex hull, so VON matches the requirement of Solipsis by default. However, there are cases where Solipsis may not discover a neighbor properly or efficiently (refer back to Figure 10 in section 2.3.3), yet the same scenario would not happen for VON. Therefore, we may consider VON as a more complete solution.

3.3.4 Problems with Voronoi-based approach

Despite the many good qualities for a Voronoi-based P2P design, there are still a number of problems. Here we discuss some that are inherent to P2P-NVE system, or to the Voronoi design.

Multiple messages

In a directly-connected P2P overlay, while it could be efficient to disseminate messages to relevant peers, a message must be sent redundantly to each connected peer. This certainly uses more bandwidth than client-server or the multicast approach, where the message is sent only once. In effect, VON's design trades bandwidth for improved responsiveness.

Lack of compression / aggregation techniques

Due to the lack of centralized processing units, fully-distributed architecture cannot utilize a popular bandwidth conserving technique: message compression and aggregation. Here we again see how P2P does not necessarily conserve bandwidth, but rather, it distributes the bandwidth and processing requirements from server to all the participating nodes. Note however that, if super-nodes are used (as in SimMud, see section 2.3.1) then compressions and aggregations can be utilized.

Worst-case neighbor size in Voronoi diagram

If there are many nodes that line up in a circular fashion (see Figure 23), then a node may be forced to make connections beyond its capacity. Such situation could overload a node if careful measures are not taken. However, we expect that such situations are rare. Certain remedies could also be devised, such as asking machines with spare capacity to host "dummy nodes" to artificially modify the Voronoi layout (a topic for future research).

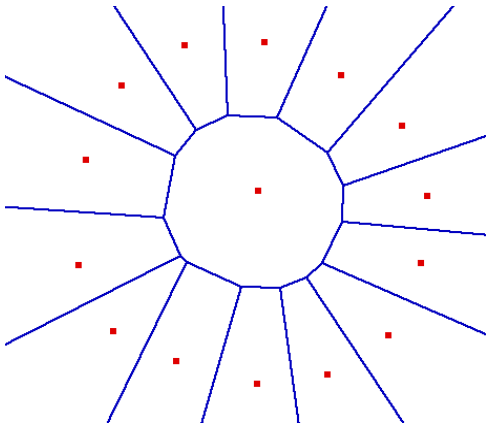


Figure 23: Worse-case scenario for connectable neighbors

3.4 Implementation considerations

The design described so far still requires some small modifications if it were to be implemented as a real system. The issues listed below are discovered either by analyzing the fine interaction of the VON algorithm, or during the implementation of the prototype library.

3.4.1 Delay counters

As nodes move around they tend to form and break connections frequently. To avoid nodes break off a recently formed connection, only to re-connect again when relative positions become favorable, a delay counter will help to ease the fluctuations. Delay counter is also useful when we adjust the AOI-radius. As frequent adjustments might cause undesirable fluctuation in connection patterns, we also set up a delay counter for AOI-radius adjustments.

The exact threshold for the counters can be application-dependent and might require a few trial-and-error experimentations to determine. In our prototype we set the disconnection delay to 3 time-steps (e.g. a non-overlapped neighbor is disconnected after it is seen as non-overlapped for 3 continuous time-steps), the AOI-radius adjustment delay to 6 time-steps, and they appear to work well.

3.4.2 Speed limit

If nodes in VON move too fast, beyond the coverage of its boundary neighbors' enclosing neighbors, then its boundary neighbors will not be able to notify the correct set of new neighbors (see Figure 24). To avoid such problem, we might need to impose a "speed limit" to node movements.

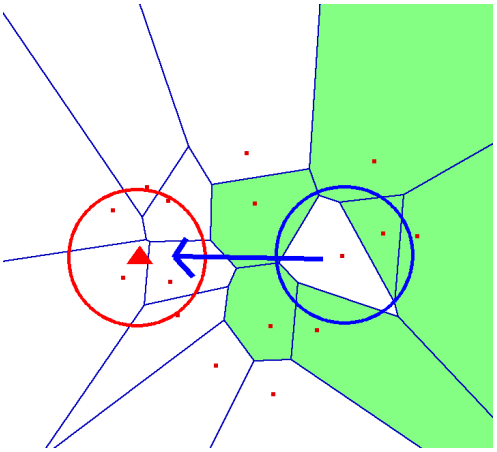


Figure 24: Potential incomplete neighbor discovery caused by fast-moving nodes

If a node moves beyond the coverage of its boundary neighbors (to the triangle position), boundary neighbors will not be able to notify the new neighbors properly.

This "speed-limit", however, is both an application-specific decision, as well as how much "movement buffer" the neighboring nodes can provide. It will also depend on network latency. For example, if the boundary neighbors of the moving node have large Voronoi regions, then the moving node is free to move at a high speed, as new neighbors will not come into view until a large distance has been traveled.

4. Simulation Results

We have implemented our Voronoi-based P2P design and verify VON's various properties with a simulation. In this part we will describe in detail, the experimental setup, metrics used in measurements and the simulation results.

4.1 Simulation setup

As it is impractical for us to test the design of VON on actual network (we would need thousands of volunteers), we have chosen to test the concepts with a simulator. We will describe both the software architecture and the hardware environment below.

4.1.1 Software architecture

We implement the VON protocol as a C++ library. Networking functions are based on the open source cross-platform library ACE [Schmidt 02]. Steve Fortune's sweepline algorithm [Fortune 86] is used for Voronoi constructions. Our prototype consists of a number of software components (C++ classes), listed in Table 5.

Table 5: C++ classes and respective functions of the VON prototype

Class	Function
von	Main interface of the VON class library
Voronoi	General interface for Voronoi-related functions, such as insertion and deletion of nodes, query and verification for enclosing and boundary neighbors.
SFVoronoi	Implementation of Steve Fortune's Voronoi construction algorithm.
NetworkAcceptor	Server-component that listens to a particular port.
NetworkHandler	Connection handler for incoming network transmission.
MessageHandler	Processing component for all incoming messages that compiles with the VON protocol (JOIN, MOVE, LEAVE, QUERY)
SimNode	Basic unit for running simulations. Acts as a simulated user in the NVE and performs movement behaviors.
Interface	Main interface to an outside simulator. It accepts parameters such as the number of nodes to be created, and the number of time-steps to be run.

4.1.2 Hardware environment and simulator

We run our simulation with a Pentium-4 2.2GB computer with 512MB of RAM. There is no actual network transmission during the simulation and all network transfers are simulated using *localhost* connections.

The software simulator runs a master program that creates each node as individual threads. It then executes the simulation with a given number of time-steps. For each time-step, the master simulator invokes each node to make one movement. Each node would send the MOVE message to all the neighbors that it currently knows, and allow some time for its neighbors to process the message. After a waiting period, the node returns execution to the master thread, which in turn invokes the next node. The behavior model for each node is that a node moves in a particular direction for the duration between 1 and 25 time-steps. At the end a new direction is randomly chosen. All nodes move with a constant velocity of 5 distance units per time-step.

To visualize the actual topology for a given simulation, we also implement a JAVA-based Graphical Users Interface (GUI) that uses the same C++ library as the simulator. A screenshot of a simulation in action is shown in Figure 25.

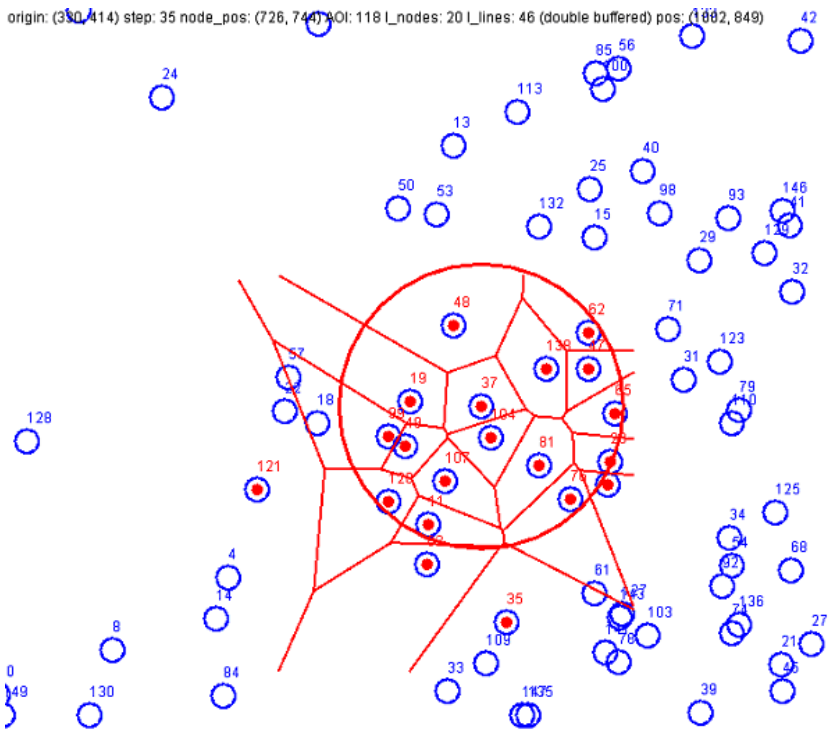


Figure 25: A screenshot of the VON simulation (JAVA GUI)

Small circles represent the nodes in the P2P network. Large circle is the AOI. Dots in the circle indicate established connections by node 37 (center). As can be seen, only a limited number of AOI-neighbors are maintained.

4.2 Metrics definition

To demonstrate VON's performance we need to define certain metrics. We decide to capture the following types of indicators during the simulation:

4.2.1 Topology consistency

To understand how P2P NVE may perform compared to client-server architecture, Kawahara *et al.* defines the metrics *consistency* [Kawahara 04]. The metrics is defined as:

$$\text{Consistency} = \frac{1}{N}\sum_{i=1}^{N}\frac{P(i)}{Q(i)},$$

Where N is the number of nodes, P(*i*) is the number of observed nodes of node *i*, and Q(*i*) is the number of the AOI neighbors of node *i*. In other words, the metrics attempts to find the ratio of neighbors *actually seen* versus the nodes that *should be seen* by a particular node. However, given that their system does not consider synchronization of events, the metrics is more appropriately termed as *topology consistency* (as opposed to the more comprehensive notion of *event consistency*). We therefore will use the term *topology consistency* to describe the metrics.

Note that *topology consistency* does not take into account the difference between the observed coordinates and the actual coordinates of a neighbor. Topology is considered consistent as long as connections are maintained with the proper AOI-neighbors. This is because it would be difficult to assign the difference in coordinates for a missing neighbor (we can assign a large value to missing nodes, but then it would impact consistency measurements in an arbitrary way, therefore it is not appropriate). The measurement for differences between the actual and observed coordinates is instead described by the next metrics.

4.2.2 Drift distance

Drift distance is described in the MiMaze paper [Diot 99] to determine how much difference exists in the views between two nodes in a distributed virtual environment system. We adopt the idea and will use it to measure how much difference exists between a node's local view and that of the correct global view of topology. Drift distance is defined simply as the difference in coordinates (in absolute value) between the observed coordinates and the actual coordinates of a node. To make the metrics useful, we average drift distance for all nodes for the duration of the simulation over the number of time-steps.

4.2.3 Average neighbor size

In order to understand how many connections are made during the simulation, as they are critically related to resource consumption and scalability of the system, we record the number of neighbors a node knows at each time-step. Two types of data are recorded: the number of *connected-neighbors* and the number of *AOI-neighbors*.

4.3 Results

We will present the results in three groups, to demonstrate the *scalability*, *topology consistency*, and *reliability* of VON. Scalability is discussed first as it is the focus of this thesis. However, scalability is not meaningful if consistency is poor, therefore, we will discuss topology consistency next. In real networking environments, packet delay and loss are inevitable, which necessarily will degrade consistency. Therefore, the key to maintain consistency in actual networks is the speed of recovery from inconsistency, which we will present as the first part of the results on reliability. In the second part of reliability discussion, we will study the effect of packet loss on VON's design. As mentioned in section 1.3, we do not consider latency currently, and assume that all messages are processed immediately once they are sent.

4.3.1 Scalability

We use the following simulation parameters to study both VON's scalability and topology consistency:

World dimension:	1000x1000
AOI-radius:	150
Packet loss rate:	0
Simulation time-step:	1000
Maximum connection per node:	10
Number of nodes:	10 to 250, with an increment of 20

Note that the parameters we choose will create a fairly dense area to test VON's performance under *crowding* situations. Also, if we assume message-update is 10 times per second (for comparison, a MMOG updates events at a rate of 3-5 times per second, while a fast pace first-shooter action game updates events at 10 – 15 times per second), then 1000 time-steps is equivalent to 100 simulated seconds, or about one and a half minute in the real world.

We show the transmission size of each node during one simulated second (10 time-step) for both the basic and dynamic AOI models. Average transmission size is shown in Figure 26 and the maximum size among all nodes is shown in Figure 27.

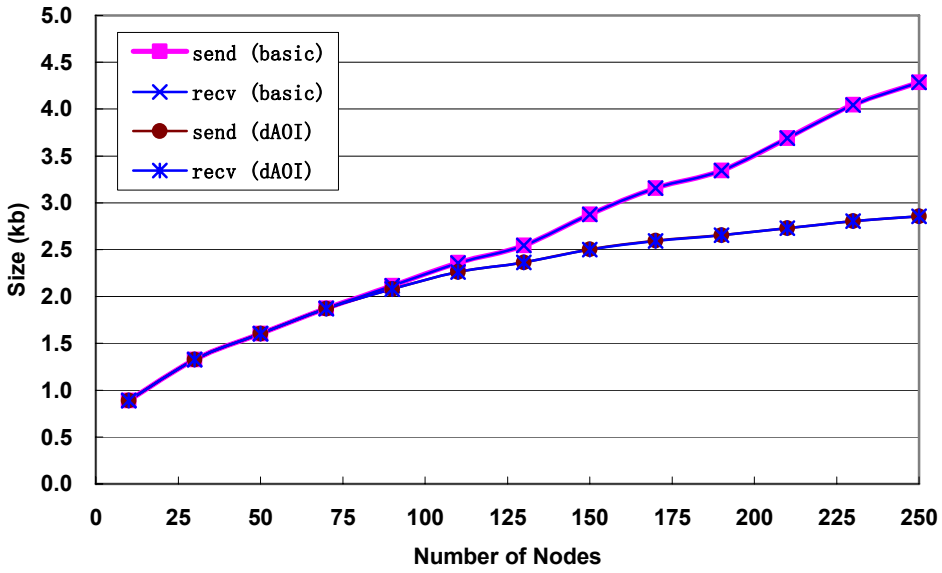


Figure 26: Average transmission size per node per second

We see that the average transmission size grows linearly for the basic model, which indicates that it will not be scalable as growth can progress *unbounded*. On the other hand, transmission size grows at a decreasing rate with dynamic AOI. Resource consumption at a given node therefore becomes *bounded*.

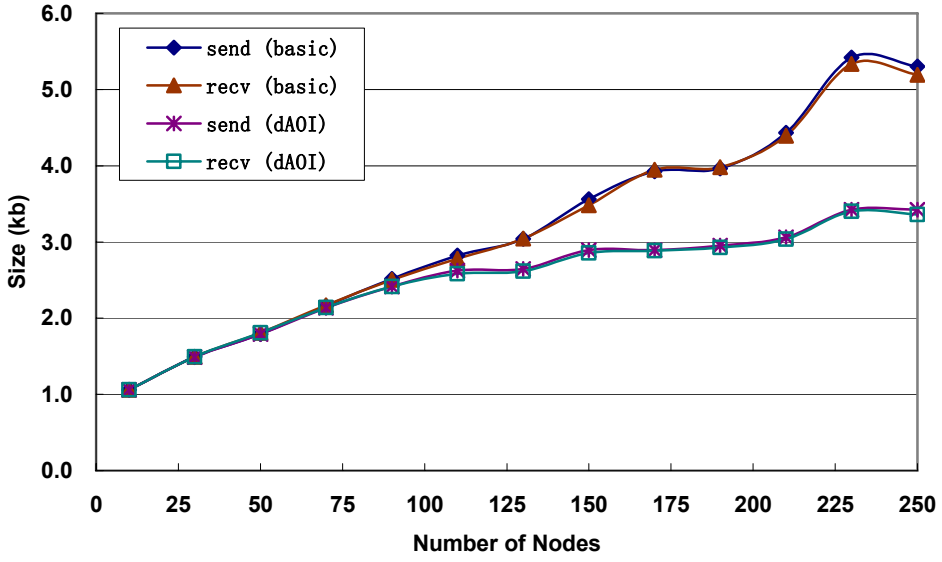


Figure 27: Maximum transmission size per second among all nodes

We see that there exists an upper-bound of about 3 kb per second for the average transmission, and less than 4kb for the maximum transmission. This shows that even the most loaded node can operate within the limits of current generation of broadband (which typically has a capacity of at least 256 kbits per second, or 32 kb per second).

Note that because we assume for zero packet loss, the average amount of messages sent and received are equal, and it is shown by the overlaps of the send and receive lines in Figure 26. In the basic model, the linear growth in transmission size is due to the fact that nodes are randomly and uniformly distributed. This makes the node density within AOI-radius to grow linearly with the total number of nodes.

The growth pattern in transmission size is explained by the number of neighbors maintained. Figure 28 shows clearly that upper bounds exist for neighbor-size in the dynamic AOI model, while the neighbor size grows linearly in the basic model. Number of average connected-neighbors approaches 10 (which is the connection limit), while the number of average AOI-neighbors approaches 6.

The difference between the average number of connected-neighbors and AOI-neighbors shows that there is a small overhead to maintain topology in VON. The additional connections are likely caused by the requirement to keep enclosing neighbors as the minimal set to ensure topology consistency. However, this overhead remains constant (at about 4 nodes) regardless of the number of nodes in the system.

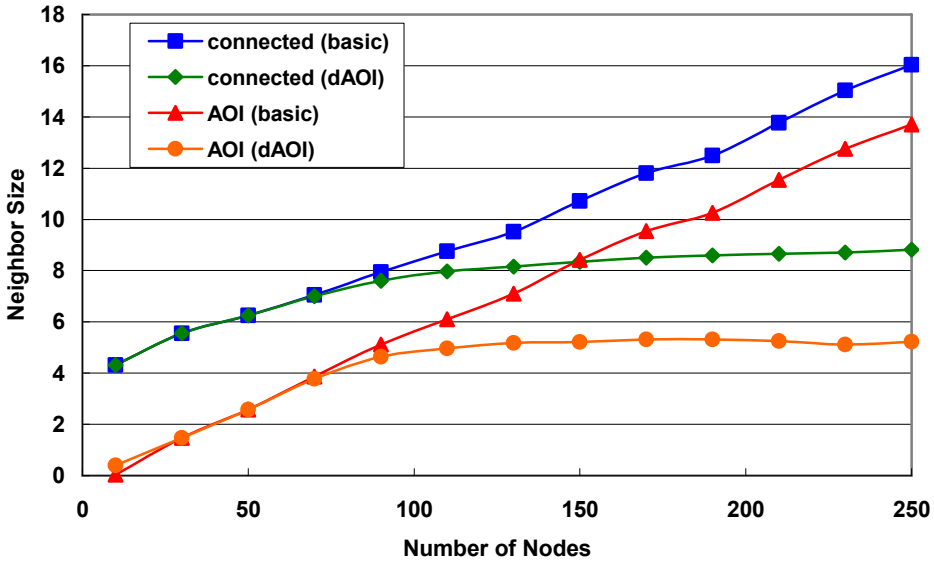


Figure 28: Average neighbor size for basic and dynamic AOI models

The benefit of P2P is more evidently demonstrated when compared to the client-server architecture. Figure 29 shows the transmission size per node per second for both client-server and VON. The server-side network resource consumption is calculated by estimation of equivalent functionality (by assuming each movement requires 8 bytes for x and y coordinates). Message size grows faster for client-server than for VON, as the server must deal with many clients simultaneously. Note also that the average message size grows faster for *send* than *receive* at the server. The reason is that while the server receives only one coordinate message from each client, it must send the coordinates of all AOI-neighbors of that client in return.

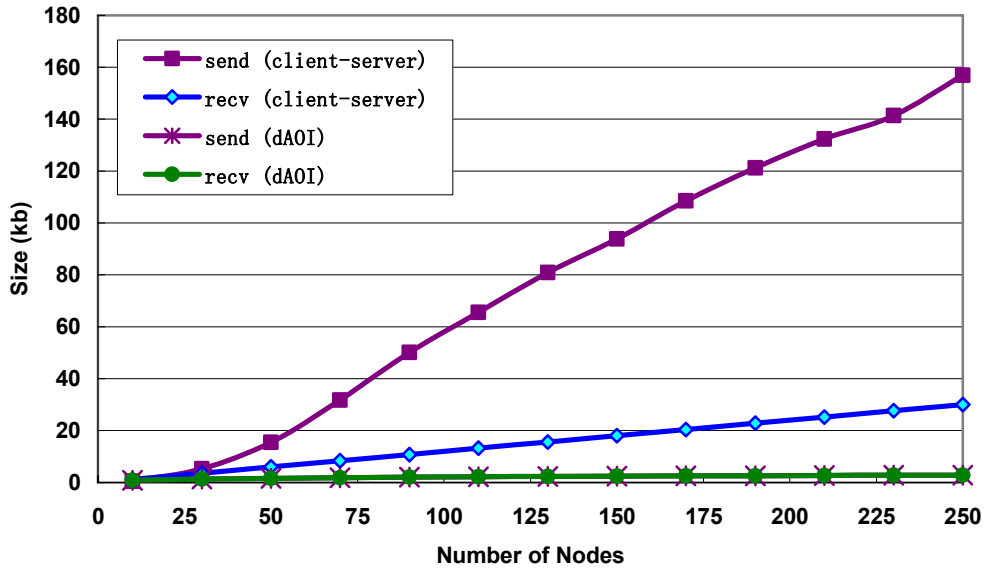


Figure 29: Comparison of transmission size between VON and client-server

Dynamic AOI adjustment thus plays an important role in achieving scalability in VON, and its effects on AOI-radius are shown in Figure 30. Here we see that the AOI-radius starts at 150 but gradually decreases to around 85 for 150 nodes. By shrinking the AOI-radius, node density within AOI thus becomes *bounded*.

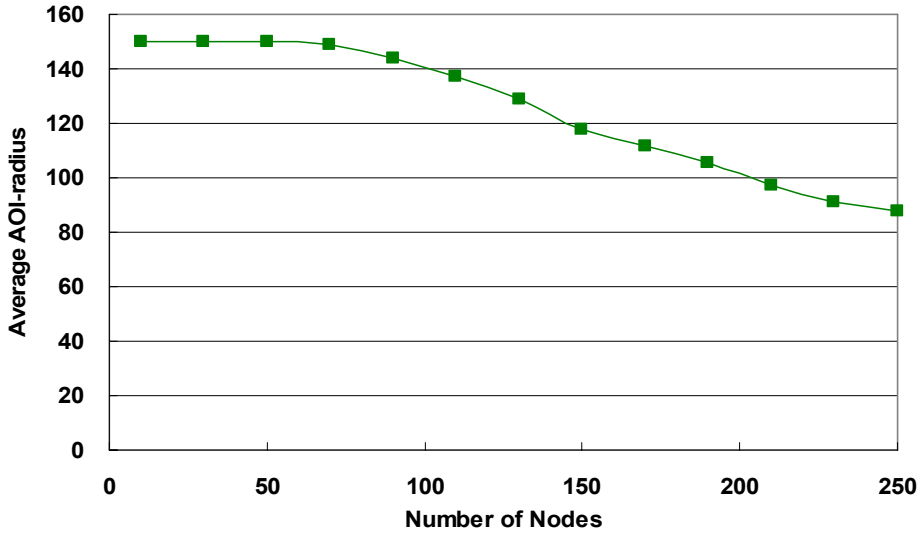


Figure 30: Effect of node increase on average AOI-radius

While the *average* number of neighbors maintained is bounded, one potential concern is that the *maximum* number of connections at a given moment might exceed a particular node's capacity. Table 6 shows the maximum and average number of neighbors kept for each simulation trial. We see that the maximum number of neighbors maintained is indeed much higher than the average (for example, 32 nodes were connected at one point versus the average of 8.7 nodes during the 230-node trial). However, Figure 27 has shown that the maximum transmission size is indeed bounded. The large connection size therefore indicates a temporary surge of connections that is quickly restored to normal. It also means that overall VON is still scalable, although connection patterns could at times be disruptive.

Table 6: Average and maximum neighbor size for basic and dynamic AOI models

			Number of Nodes												
Model			10	30	50	70	90	110	130	150	170	190	210	230	250
basic	connected	max	9	13	13	15	17	21	24	23	27	26	41	35	44
		avg	4.31	5.56	6.26	7.06	7.94	8.76	9.52	10.72	11.81	12.49	13.77	15.04	16.03
	AOI	max	3	6	11	13	15	19	20	20	24	24	30	31	31
		avg	0.04	1.47	2.58	3.87	5.12	6.10	7.11	8.42	9.54	10.26	11.54	12.76	13.71
dAOI	connected	max	9	13	13	14	16	19	24	22	22	26	23	32	28
		avg	4.31	5.56	6.26	7.01	7.61	7.97	8.15	8.34	8.51	8.59	8.65	8.70	8.82
	AOI	max	3	6	11	12	12	19	14	20	20	23	22	26	26
		avg	0.40	1.47	2.58	3.78	4.64	4.97	5.18	5.21	5.32	5.32	5.25	5.12	5.23

4.3.2 Topology consistency

Figure 31 shows the topology consistency measurements for both the basic and dynamic AOI models as a function of total number of nodes in the system.

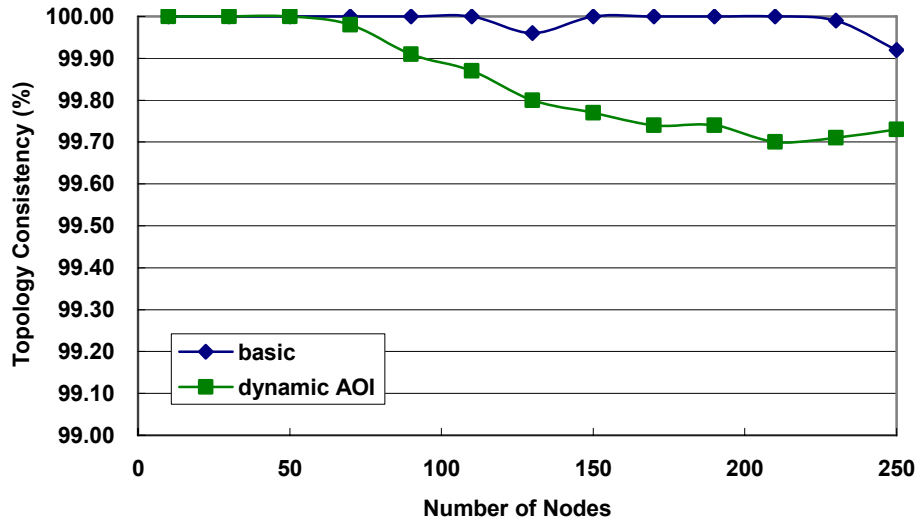


Figure 31: Topology consistency for basic and dynamic AOI models

Here it is clear that topology consistency remains close to 100% for all trials in the basic model. The slight consistency drop (for example, 99.92% for 250 nodes) is likely due to a simulator problem. As observed during the simulation, some nodes may occasionally stop to respond to other nodes, causing them to become increasingly inconsistent in their knowledge of neighboring nodes.

A larger decrease in topology consistency exists for the dynamic AOI model. However, it should be noted that the lowest value is still above 99.70%. The cause is likely due to asymmetric understandings of neighbor relationship after AOI-radius adjustments (refer back to Figure 22 in section 3.2.7). However, inconsistencies are bound to occur in real networks where latency and pack loss exist. Therefore, the more important question to ask is: can inconsistency be fixed quickly when occurred? We will answer this question in the next section.

The high degree of topology consistency in VON is further confirmed by the measurements of average drift distance (shown in Figure 32). Here we see that the average value of drift distance is very low (i.e. close to 0) for both the basic and dynamic AOI models.

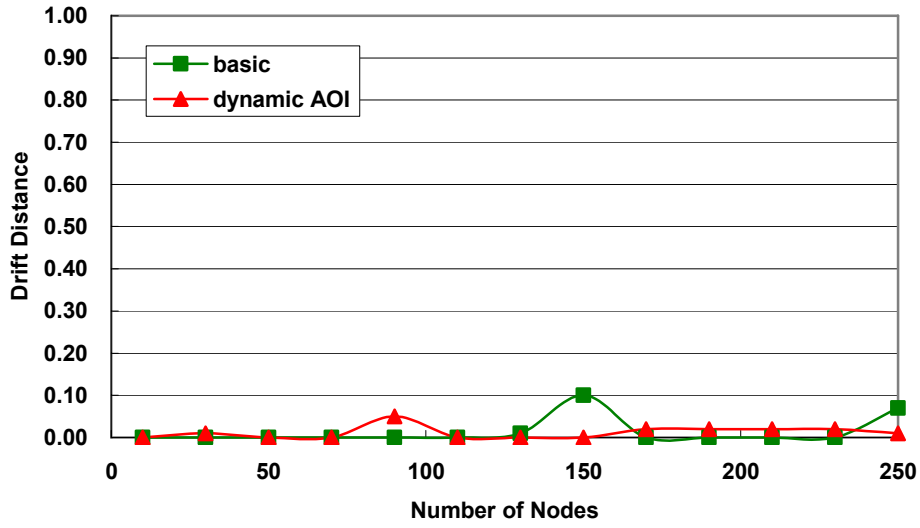


Figure 32: Average drift distance for basic and dynamic AOI models

4.3.3 Reliability (recovery from inconsistency)

Figure 33 shows a time-series of changes in topology consistency for 110 nodes between time-step 150 and 250. We see that the per-step average consistency remains high between 99% and 100%. For particular nodes, we do see drops in consistency occur occasionally. However, the time-series shows that the drops usually recover quickly within a few time-steps. Note that some of the low consistencies seen (for example, consistency for node 1 drops to 70% at one point) do not necessarily indicate poor performance, as topology consistency can become low easily when AOI-neighbors are few (for example, if only one of the two visible AOI-neighbors are seen, then consistency is calculated to be only 50%).

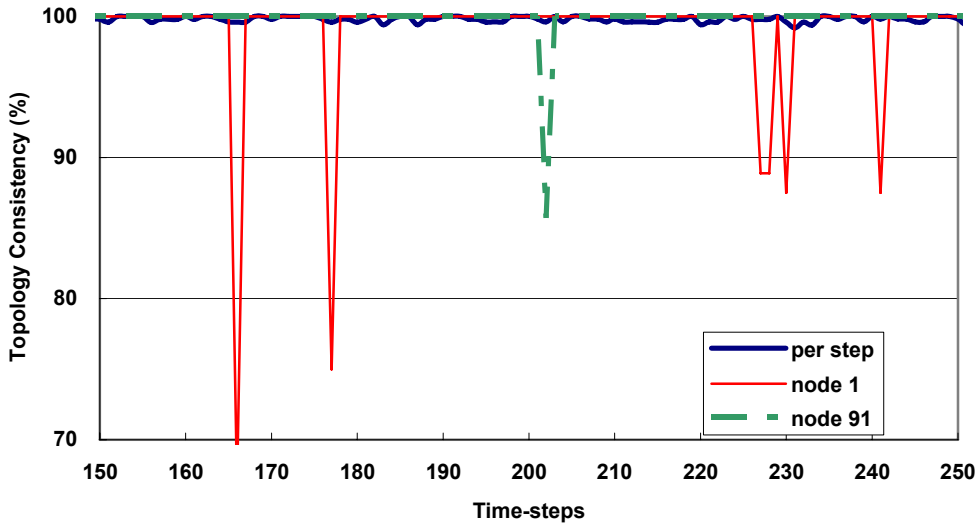


Figure 33: Change in topology consistency for 110 nodes (time-steps: 150-250)

A more relevant indicator then, is how many time-steps VON takes to recover from inconsistency, which is shown in Figure 34. We see that the average number of recovery-steps is about 1.3 to 1.5, and stays relatively constant for node size above 100. This shows that VON maintains good robustness against topology inconsistency.

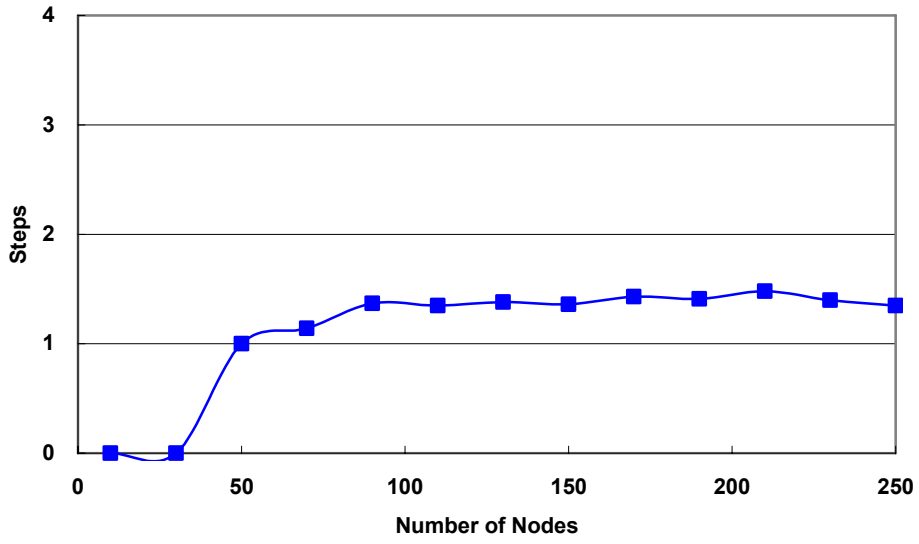


Figure 34: Average number of steps to recover from inconsistency

4.3.4 Reliability (effect of packet loss)

We will now discuss how VON performs in environments where packets may be lost during transmission. We use the following parameters for simulation:

World dimension:	1000x1000
AOI-radius:	150
Packet loss rate:	0% ~ 100%
Simulation time-step:	1000
Maximum connection per node:	10
Number of nodes:	150

Packet loss is simulated by generating a random number when processing a received message. If the random number falls below the specified loss rate, then the message is ignored. Packet loss is applied to MOVE, MOVE_B, MOVE_BD, and NODE messages (refer back to section 3.2.6 for the message protocol). Other message types are still delivered reliably, as they are essential to ensure the correct function of the overlay. Dynamic AOI model is adopted for the simulation as we want to see whether scalability can still be achieved when packet loss exists.

We first show the effect of packet loss on topology consistency in Figure 35. Topology consistency is maintained at a high level even when the loss rate is 50%. After a certain critical point (after 60% packet loss) consistency begins to drop down dramatically (which might indicate the existence of overlay partitions).

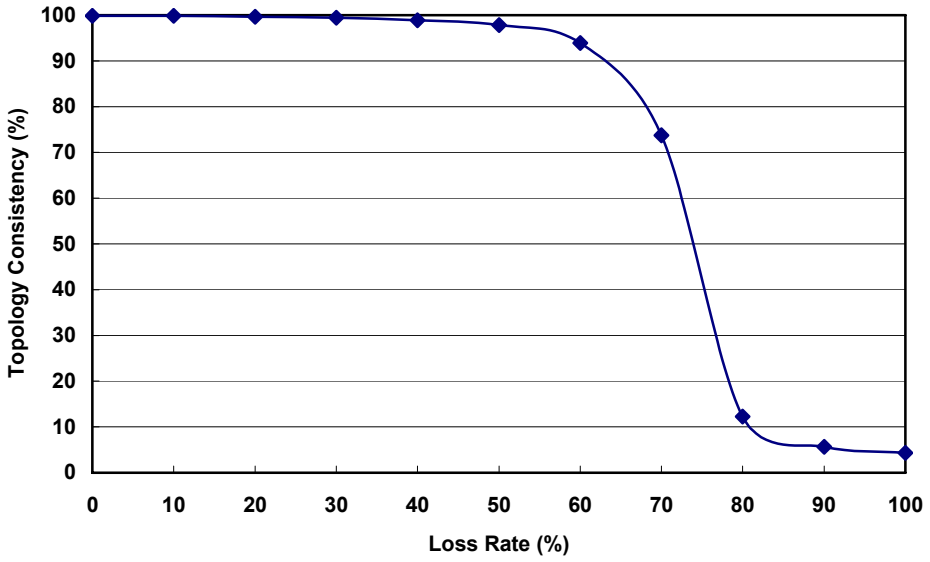


Figure 35: Effect of loss rate on topology consistency (dynamic AOI model)

The effect of packet loss on average drift distance is shown in Figure 36. We can see that for loss rate below 50%, the drift distance remains relatively low (but grows exponentially), and begins to increase significantly after a loss rate of 60%.

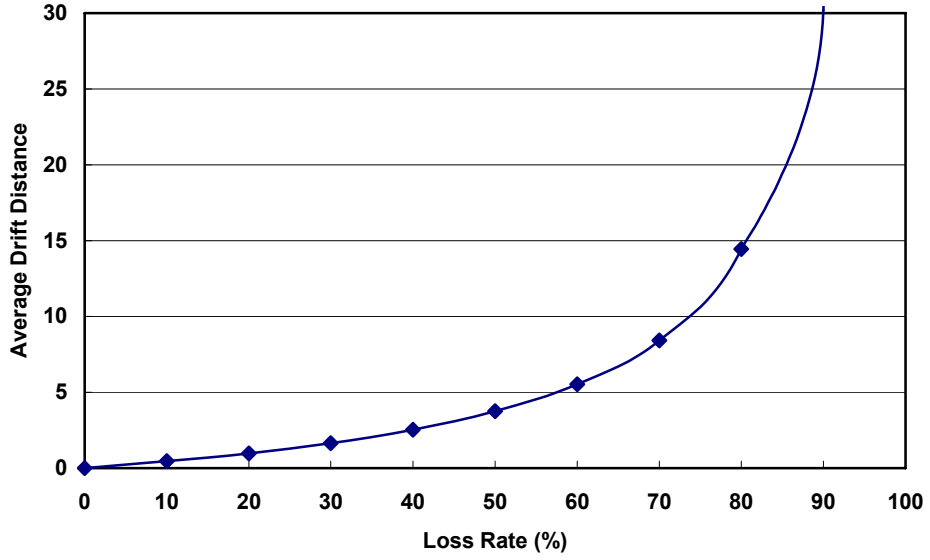


Figure 36: Effect of loss rate on average drift distance

Figure 37 shows the effect of loss rate on average neighbor size for both connected-neighbors and AOI-neighbors. Here we again see that neighbors are maintained in a relatively stable fashion until a loss rate of 60%, when the neighbor size begins to drop dramatically.

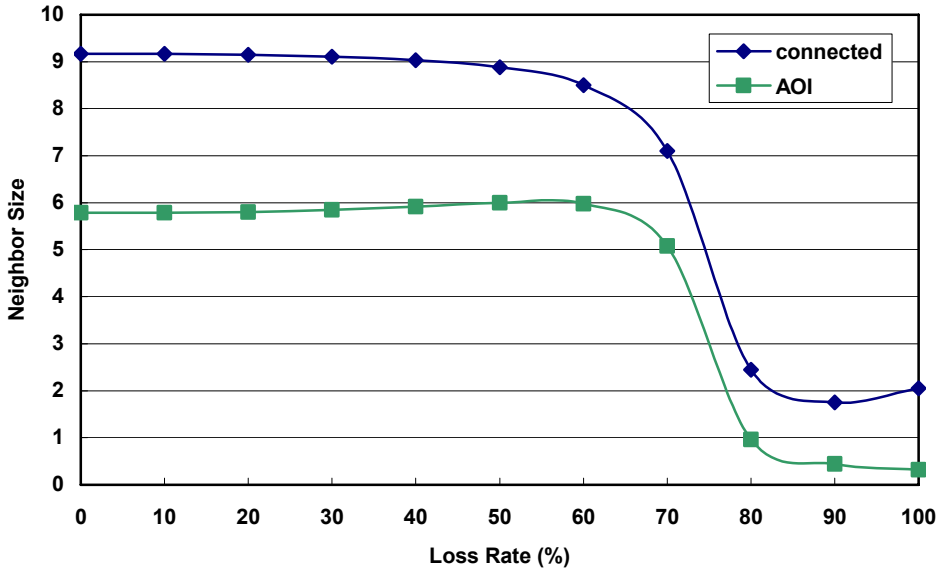


Figure 37: Effect of loss rate on average neighbor size

In Figure 38, we see the number of steps to recover from inconsistency grows slowly for loss rate below 50%, then increases sharply after the loss rate of 60%. This result again confirms that a critical point exists between the loss rate of 50% and 60%.

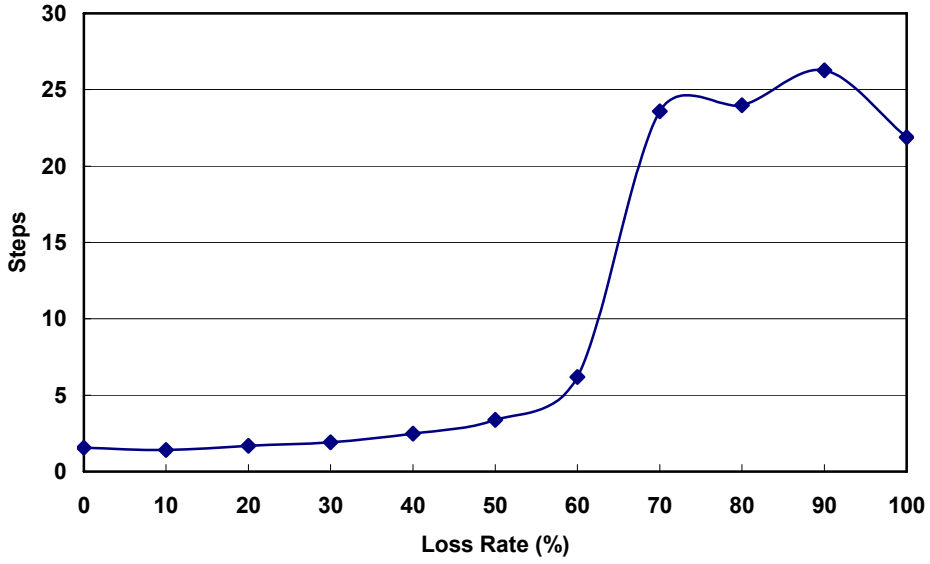


Figure 38: Effect of loss rate on average recovery steps

In summary, it can be seen that VON maintains relatively stable behavior for loss rate up to 50%, and only starts to degrade after a loss rate of 60%. This is good indication that VON is reliable and may endure severe packet loss.

We suspect that at the critical point of loss rate between 50% and 60%, some overlay partition might be occurring. However, confirmation and solution to this scenario is left for future research (see section 5.3.2 for discussion).

One remaining question is: how reasonable is our choice to allow loss in only the MOVE and NODE messages? In other words, is guaranteed delivery of other types of messages suitable in real systems, as guaranteed delivery could degrade system performance (i.e. responsiveness) by introducing too much delay?

To answer this question, we also measure the percentage breakdown of total transmission by message types (see Table 7). Here we see that all MOVE-related messages (i.e. MOVE, MOVE_B, and MOVE_BD) and NODE messages together occupy roughly 95% of all transmission. Therefore, sending other types of messages reliably (for example, via TCP) will not degrade system performance.

Table 7: Percentage breakdown of transmission size by message types (for 150 nodes)

	Loss rate (%)								
Types	**0**	**10**	**30**	**50**	**70**	**90**	**Avg**	**Min**	**Max**
GREET	0.54	0.55	0.55	0.56	0.58	0.42	0.53	0.42	0.58
ID	0.00	0.00	0.00	0.00	0.01	0.02	0.01	0.00	0.02
QUERY	0.04	0.04	0.05	0.05	0.07	0.36	0.10	0.04	0.36
HELLO	2.79	2.82	2.84	2.87	2.94	1.89	2.69	1.89	2.94
EN	1.10	1.11	1.13	1.13	1.14	0.76	1.06	0.76	1.14
MOVE	6.51	6.50	6.74	7.37	7.58	2.51	6.20	2.51	7.58
MOVE_B	55.64	55.62	56.41	57.98	62.70	86.74	62.52	55.62	86.74
MOVE_BD	1.72	1.69	1.57	1.44	1.28	0.47	1.36	0.47	1.72
NODE	31.66	31.66	30.70	28.60	23.71	6.84	25.53	6.84	31.66
Sum	100.00	100.00	100.00	100.00	100.00	100.00	100.00		
MOVE+NODE	95.52	95.47	95.43	95.39	95.27	96.56	95.61		

5. Conclusion

5.1 A promising scalability solution

We have presented a promising solution for constructing scalable Networked Virtual Environment based on Voronoi diagram. The key idea of the design is for each node to maintain a Voronoi diagram of the neighboring nodes within its area of interest. Although demonstrating scalability in a real system is not practical for the current work, we have shown the scalability potential of Voronoi-based P2P with simulation results.

In the simulation, it is shown that there are upper bounds to message transmission (both in maximum and average message size) and the average number of neighbors maintained by a node. This indicates that the amount of bandwidth and processing requirement for each node is *bounded*, independent of the total number of nodes in the system. The bounded resource-consumption characteristic is evidence that the system is scalable. The proposed design offers several attractive qualities for creating large-scale virtual environments:

Scalable

VON restricts message traffic transmitted between host computers to only those that are relevant, achieving *near-ideal* interest management and message filtering. As the bandwidth and processing requirement for each node is independent of the total number of nodes in the system, and depends only on the number of neighboring nodes that it interacts with, the system is thus scalable.

Efficient

Not only message traffic is filtered down to near-ideal interest management, but because P2P topology is maintained with minimal overhead, redundant information exchange is greatly avoided. By utilizing direct connection between peers, messages also reach their relevant targets with the least amount of latency.

Robust

By making the architecture fully-distributed, there is no single point of failure that exists in client-server architecture. Also, due to the fact that each node maintains direct link with those that it interacts with, recovery from node failure can be done quickly with little overhead.

Simple

The algorithm for keeping existing neighbors and discovering new neighbors are facilitated with Voronoi diagram, with only a few rules to follow. Compared to other P2P schemes where each node may need to maintain various types of list or table, or sophisticated partitioning and merging mechanism for regions, VON's design is rather simple. It is also simpler than server-cluster architecture where various partitioning, load-balancing, and entity-migration mechanisms need to be considered.

Affordable

As the design is fully-distributed and does not place heavy responsibility onto any one node, the whole system may be constructed with just one light-weight gateway server, making it affordable to small NVE developers, even individuals.

5.2 Potential applications

VON's scalability, efficiency, and robustness make it a natural foundation for constructing certain types of applications. We discuss some of the possibilities below:

Massively Multiplayer Online Games (MMOG)

Although we have not developed adequate mechanism to address persistency and security issues, which are crucial for commercial NVEs, VON can still be useful to MMOG by relieving the server of user position maintenance. If there is no serious security consideration for user positions, then position updates (which usually take a large portion of network traffics in MMOG) can be maintained by using VON.

Large-scale military simulation

VON's design makes it suitable for constructing truly large-scale virtual environments affordably. Its scalability and low-cost characteristics match the design goals for SIMNET (refer back to section 2.1), which is to build a large number of low-cost simulators. As security is generally not an issue for military simulators (all simulators are under the military's internal controls), military applications of VON might actually be easier to realize than commercial ones.

3D Web

There are now many websites that offer 3D virtual navigation for training, product demonstration, or touring purposes. However, most of these navigations only allow single-user experience. Part of the reason is that there has not been an easy and affordable way to add multi-user capability to such environments, and all current solutions require costly server-side investments. The simplicity and affordability of VON could allow multi-user capability be added to these 3D environments.

Scientific simulations

Although we have mostly discussed VON in the context of virtual environment applications, viewed in a more general way, VON is in fact a mechanism to interconnect many dispersedly located nodes placed in some logical dimensions that require frequent synchronizations. Therefore, for certain scientific simulations that have a spatial orientation and require frequent state synchronizations, VON might be a possible platform to run such simulations on a large scale.

5.3 Future research

Work remains to be done to make VON more comprehensive to support the full functional requirements of NVE systems. We will discuss a few major topics for future research in this section.

5.3.1 Reliability measurements

We have discussed VON's reliability in terms of how fast it recovers from topology inconsistency and some simulated effects of packet loss. However, a more complete and realistic analysis of reliability should also consider latency and the issue of node failures. Experiments conducted in actual network environments will also shed lights on VON's real performance. Additionally, the effects of different movement velocities on topology consistency should also be studied.

5.3.2 Overlay partition problem

All P2P overlay networks face the problem of splitting into separate partitions (e.g. the overlay becomes not fully-connected). We have shown earlier that it would not happen under normal operation (if the number of failing node is small). However, if a large number of nodes fail simultaneously, or if packet loss rate is high, it is still possible for the overlay to be broken into separate parts. Kawahara *et al*. suggested in their paper that each node may keep information for some distant nodes, so that contacts can be made to restore the overlay in case of massive failures. They also introduced the concept of *random introduction* [Kawahara 04], which is for each node to check back with the server periodically to re-discover missing nodes. One important research direction therefore is a distributed recovery mechanism that provides high reliability against massive node failures.

5.3.3 Distributed event consistency

Event consistency has not yet been considered in the present design, therefore it is not guaranteed in the simulation. Some may consider it difficult to maintain event consistency without using a centralized authority (e.g. client-server architecture). However, we argue that it is not only achievable, but might even be done more efficiently than centralized approach.

We note that in any distributed system such as NVE, due to network latency, absolute event consistency simply does not exist (i.e. it takes time for message to travel from host to host, messages cannot be processed and executed at the *same time*, if time is defined as an objective wall-clock time). What centralized event processing

achieves is simply the preservation of *casual order consistency* and *time-space consistency* mentioned earlier (in section 1.5). Event consistency may be more appropriately called "*bounded inconsistency*". That is, although inconsistency exists, it is tolerated as long as it does not *diverge* and disturb user experience. What client-server architecture represents, with respect to consistency, are in fact two mechanisms:

1) A notion of absolute reality (e.g. a true version of reality).
2) A mechanism to correct inconsistency so it remains *bounded*.

Element 1) is simply the version of various game states maintained at the server, including the order of event occurrence (casual order consistency) and the actual values of various states (time-space consistency). In fact, because server's version is considered "the truth", it could even sacrifice a client's idea of its own position if it contradicts the server's version, even though strictly speaking, a client should know better what its own current position is. Client-server architecture simply chooses to ignore any client-side judgment in favor of the server's view of the world. Element 2) is done by sending out server states periodically to clients, so that all the clients follow the same event occurrences and states updates, more or less within the server's version.

We think that the notion of event consistency commonly referred in client-server architecture can be dissected into the above elements. Therefore, we may achieve event consistency for P2P systems if these two elements can also be defined. To create *bounded consistency* in VON, we offer some initial ideas:

Definition of the "true version":

In client-server architecture, the authoritative event order and states are defined as the version which the server maintains. However, in a fully-distributed system no central repository of states exists, so we necessarily need to resort to a model where each node keeps its own version of part of the "global reality". The global environment will exist more as a *consensus* between nodes than as synchronization of all the events (which is difficult to do efficiently, see the idea in Crosbie Fitch's article "*Cyberspace in the 21st Century*" [Fitch 01]).

One possibility is to allow each node to be the owner of all event states within its Voronoi region (note that the states include the node's own position and the various objects that might present in the region). Then, as the Voronoi region changes shape

due to position updates, ownership may be transferred to other nodes in a deterministic manner. However, at all times each node remains the authoritative owner of its own position. In such a scheme, instead of one machine maintaining the true version, every machine maintains a small part of the complete states of the world.

Recovery mechanism from inconsistency

Some previous works to achieve event consistency in a distributed simulation exist in the literature, namely, the concept of *critical causality* to preserve casual order consistency [Zhou 02], and *bucket synchronization* to preserve time-space consistency [Diot 99]. We will describe each of them below:

Critical Causality

Real-world events happen according to certain casual orders, and we become accustomed to such ordering. We will find simulation behavior "weird" if such ordering is violated. The concept of critical causality is to define events as "trigger-response" pairs, such that if an event is created as the result of a previous event, then the previous event is included in the message sent to other entities. This way, the receivers of the "response event" will also know the "triggering event" and be able to display it first, should it miss the trigger event. This *casual receive order delivery* will preserve the correct causality of trigger and response events, but not necessarily preserve *absolute ordering* over logical time. This way it could avoid the costly delay required for absolute event ordering and improve responsiveness of the simulation.

Bucket Synchronization

The idea of bucket synchronization is that simulation is divided into discrete *time buckets*. Each bucket has a clearly defined *start time* and *end time*, and is followed by the next bucket. As messages are received from other nodes, they are placed into respective buckets according to the time-stamp of the sender of the message. When the end time of a bucket is reached, event messages in the same bucket are sorted by time-stamp, and then processed together.

By adding a little delay to the event processing, synchronizations of all the relevant events can be achieved, given that event messages do arrive no later than a pre-specified *playout delay*. If a message is lost in transmission or delayed, then it will be ignored. Some dead reckoning mechanism might be used to estimate entity position in place of the missing message based on historical records.

In the MiMaze paper, experiments were carried out such that the playout delay is set to 100ms, and there are 25 buckets per second (each bucket is 40ms long). Users of the system reported smooth interactivity.

Event consistency is one aspect that has not being addressed in the present work. However, the above discussion should give some ideas as to how it could be achieved to an acceptable degree.

5.3.4 Persistency maintenance

Persistency is the next requirement after scalability for creating a realistic, immersive environment. It is also the corner stone for building true cyberspace. However, achieving persistency on P2P architecture is also an inherently difficult problem, as nodes in P2P overlay may fail at any time. Persistency therefore either has to be achieved with some centralized repository or with sufficient data redundancy across many nodes.

We expect that in the end, a decentralized solution to persistency would be required if scalability is to be supported, however, persistency cannot be achieved unless we can make the assumption that a large percentage of nodes in the virtual environment stay online most of the time.

Our current thought about potential persistency solution in VON is to have each node manage the object or event states within its own Voronoi region (as mentioned previously in section 5.3.3). This would allow transient states be maintained. To make states persistent, each node may send back a state update back to a central “game state server”. The server would maintain the persistency for all the states. The server does not interact with any node, but simply serves as a repository. Nodes and server may negotiate the frequency for update, so that server would not be overloaded.

This would be a centralized solution to the persistency problem. For decentralized solution, we might look into distributed database research such as OceanStore [Kubiatowicz 00] for inspirations.

5.3.5 P2P-based 3D streaming

To make NVE truly accessible and affordable, it is best if various types of NVE can be visited as easily as browsing the web, with a universal client program that could access various NVEs set up by companies or individuals. However, this would require large amount of 3D content data, such as 3D models, textures, animations be downloaded in real-time. A type of 3D streaming technique [Sahm 04] may thus be required alongside with message exchange and filtering techniques.

We consider 3D streaming a crucial part in promoting accessibility to NVEs, and it would create even greater impact if it is constructed on top of a matured P2P architecture. It is still early to discuss about P2P-based 3D streaming, as both P2P and 3D streaming are relatively new concepts, yet we do feel that their importance and applicability will become evident as infrastructure matures and P2P applications become integral to daily life.

List of References

[AC] Asheron's Call. [Online]. Available: http://ac.turbinegames.com

[Aurenhammer 91] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, pp. 345-405, 1991.

[Butterfly 03] Butterfly.net Inc., *The Butterfly Grid*, 2003. [Online]. Available: http://www.butterfly.net/platform

[Clarke 00] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Proc. ICSI Workshop Design Issues in Anonymity and Unobservability*, Berkeley, CA, June 2000, pp. 46-66.

[Diot 99] C. Diot, L. Gautier, "A distributed architecture for multiplayer interactive applications on the Internet," *IEEE Network*, vol. 13, pp. 6-15, 1999.

[Dwyer 86] R. A. Dwyer, "A simple divide-and-conquer algorithm for computing Delaunay triangulations in $O(n \log \log n)$ expected time," in *Proc. 2nd Annual Symp. Computational Geometry*, Yorktown Heights, NY, 1986, pp. 276-284.

[eDonkey] eDonkey. [Online]. Available: http://www.edonkey2000.com

[EQ] Everquest. [Online]. Available: http://www.everquest.com

[Fitch 01] C. Fitch, "Cyberspace in the 21st century: Foundation II," [Online]. Available: http://www.gamasutra.com/features/20010226/fitch_01.htm

[Fortune 86] S. Fortune, "A sweepline algorithm for Voronoi diagram," in *Proc. 2nd Annual Symp. Computational Geometry*, Yorktown Heights, NY, 1986, pp. 313-322.

[Funkhouser 95] T. A. Funkhouser, "RING: A client-server system for multi-user virtual environments," in *Proc. 1995 Symp. Interactive 3D Graphics*. Apr. 1995. pp. 85-92.

[Gnutella] Gnutella. [Online]. Available: http://gnutella.wego.com

[Gowda 83] I. G. Gowda, D. G. Kirkpatrick, D. T. Lee, and A. Naamad, "Dynamic Voronoi diagrams," *IEEE Trans. Information Theory*, vol. 29, pp. 724-731, 1983.

[Guibas 85] L. Guibas and J. Stolfi, "Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams," *ACM Trans. Graphics*, vol. 4, pp. 74-123, April 1985.

[Kawahara 04] Y. Kawahara, T. Aoyama, and H. Morikawa, "A peer-to-peer message exchange scheme for large-scale networked virtual environments," *Telecommunication Systems*, vol. 25, pp. 353–370, 2004.

[Keller 02] J. Keller and G. Simon, "Towards a peer-to-peer shared virtual reality," in *Proc. 22nd Int. Conf. Distributed Computing Systems (Workshops)*, Vienna, Austria, Jul. 2002. pp. 695-700.

[Keller 03] J. Keller and G. Simon, "Solipsis: A massively multi-participant virtual world," in *Proc. Int. Conf. Parallel and Distributed Techniques and Applications (PDPTA 03),* Las Vegas, NV, 2003, pp. 262-268.

[Knutsson 04] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 96-107.

[Korpela 01] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky, "SETI@home-massively distributed computing for SETI," *IEEE Computing in Science & Engineering,* vol. 3, pp. 78-83, 2001.

[Kubiatowicz 00] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao, "OceanStore: an architecture for global-scale persistent storage," in *Proc. 9th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, 2000, pp. 190-201.

[Kung 01] H. T. Kung and C. H. Wu, "Hierarchical peer-to-peer networks," Inst. of Information Science, Academia Sinica, Taiwan, Technical Report IIS-TR-02-015, Apr. 2001.

[Liebeherr 02] J. Liebeherr, M. Nahas, and W. Si, "Application-layer multicast with delaunay triangulations," *IEEE J. Selected Areas in Communications*, vol. 20, pp. 1472-1488, Oct. 2002.

[Lineage] Lineage. [Online]. Available: http://www.lineage.com/

[Macedonia 95-1] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, "Exploiting reality with multicast groups," *IEEE Computer Graphics and Applications*, vol. 15, pp. 38 - 45, Sept. 1995.

[Macedonia 95-2] M. R. Macedonia, D. P. Brutzman, M. J. Zyda, D. R. Pratt, P. T. Barham, J. Falby, and J. Locke, "NPSNET: a multi-player 3D virtual environment over the Internet," in *Proc. 1995 Symp. Interactive 3D Graphics,* Monterey, CA, 1995, pp. 93-94.

[Miller 95] D. C. Miller and J. A. Thorpe, "SIMNET: The Advent of Simulator Networking," in *Proc. IEEE,* vol. 83, pp. 1114-1123, Aug. 1995

[Morse 00] K.L. Morse, L. Bic, and M. Dillencourt, "Interest management in large-scale virtual environments," *Presence*, vol. 9, pp. 52-68, 2000.

[Napster] Napster. [Online]. Available: http://www.napster.com

[NCSoft 04] NCSoft, Corp. "IR Report Mar. 2004." [Online]. Available: http://www.ncsoft.net

[Ratnasamy 01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM,* San Diego, CA, Aug. 2001, pp. 161-172.

[Rowstron 01] A. Rowstron and P. Druschel, "Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proc. 18th IFIP/ACM Int. Conf. Distributed Systems Platforms (Middleware)*, Nov. 2001

[Sahm 04] J. Sahm, I. Soetebier, and H. Birthelmer, "Efficient Representation and Streaming of 3D Scenes," *Computers and Graphics*, vol. 28, pp. 15-24, 2004.

[Schmidt 02] D. C. Schmidt and S. Huston, *C++ Network Programming: Mastering Complexity with ACE and Patterns*. Addison-Wesley Longman, 2002. [Online]. Available: http://www.cs.wustl.edu/~schmidt/ACE.html

[Singhal 96] S. K. Singhal and D. R. Cheriton, "Using projection aggregations to support scalability in distributed simulation," in *Proc. 16th Int. Conf. Distributed Computing Systems*, Hong Kong, May 1996, pp. 196-206.

[Singhal 99] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation,* ACM Press, New York, 1999.

[Skype] Skype. [Online]. Available: http://www.skype.com

[Stoica 03] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Trans. Networking,* vol. 11, pp. 17-32, Feb. 2003.

[Stytz 96] M. Stytz, "Distributed virtual environment," *IEEE Computer Graphics and Applications,* vol. 16, pp. 19-31, May 1996.

[UO] Ultima Online. [Online]. Available: http://www.uo.com

[Zhao 04] B.Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *IEEE J. Selected Areas in Communications*, vol. 22, pp. 41-53, 2004.

[Zhou 04] S. P. Zhou, W. T. Cai, B.-S. Lee, and S. J. Turner, "Time-space consistency in large-scale distributed virtual environments," *IEEE Trans. Modeling and Computer Simulation*, vol. 14, pp. 31-47, 2004.

[Zona 03] Zona Inc, "Terazona: Zona application frame work," 2003. [Online]. Available: www.zona.net/whitepaper/Zonawhitepaper.pdf

[Zyda 92] M. J. Zyda, D. R. Pratt, J. G. Monahan, and K. P. Wilson, "NPSNET: constructing a 3D virtual world," in *Proc. 1992 Symp. Interactive 3D Graphics*, Cambridge, MA, 1992, pp.147-15

Thesis revision: 3.8 (2005/04/04 17:30, 2005/10/11 10:22, 2006/03/08 17:19, 2006/07/01 22:28)

Scalable Peer-to-Peer Networked Virtual Environment

Shun-Yun Hu
Dept. of Computer Science and Information Engineering
Tamkang University
Tamsui, Taipei County 251, Taiwan
syhu.tw@yahoo.com.tw

Guan-Ming Liao
Institute of Physics
Academia Sinica
Taipei 11529, Taiwan
gm.liao@msa.hinet.net

ABSTRACT
We propose a fully-distributed peer-to-peer architecture to solve the scalability problem of Networked Virtual Environment in a simple and efficient manner. Our method exploits locality of user interest inherent to such systems and is based on the mathematical construct *Voronoi diagram*. Scalable, responsive, fault-tolerant NVE can thus be constructed and deployed in an affordable way.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design --- Distributed networks

General Terms: Algorithms, Design

Keywords
Networked Virtual Environment (NVE), peer-to-peer (P2P), massively multiplayer (MMP), Voronoi diagram, scalability, interest management

1. INTRODUCTION
Networked Virtual Environments (NVEs) [13] are computer-generated, synthetic worlds that allow simultaneous interactions of multiple participants. Since the early days of SIMNET, a U.S. Government project for large scale combat simulations, to the recent boom of Massively Multiplayer (MMP) Online Games (MMOG) [8], efforts to allow people to interact in realistic, immersive virtual environments have gone a long way. Science fiction works, such as Neal Stephenson's novel *Snow Crash* and the recent *Matrix* movie trilogy, serve as inspirations to many for the eventual creation of a 3D environment that is truly massive, persistent, realistic and immersive. With rapid technology developments, converging advances in CPU, 3D accelerator, and bandwidth may make the vision come true in the foreseeable future. However, a number of issues exist in the creation of a large-scale NVE, namely:

Consistency - For meaningful interactions to happen, each user's experiences in the virtual world must be more or less consistent. This includes maintaining states and keeping events synchronized.

Performance / Responsiveness - NVEs are simulations of the real world. Responsiveness therefore is important for immersion. However, performance requirements vary between applications.

SIGCOMM'04 Workshops, Aug. 30 & Sept. 3, 2004, Portland, OR, USA.

Security - Most NVEs allow people to engage competitively (e.g. combat or treasure hunt). User authentication and fairness against cheating therefore are required. In fact, this is often the most concerned issue by commercial NVE developers.

Scalability – Scalability usually concerns with the number of simultaneous users in NVE [13]. It is important in two respects: (1) Content possibility. Certain game plays are only realizable when many people participate, such as community and social oriented game play. (2) Service availability. Large-scale NVEs are similar to websites, where usage may increase dramatically and unexpectedly. Systems will break if they are not scalable.

Persistency - To create sophisticated contents, certain data, such as user profile and valuable virtual objects, must be persistently stored and accessed between user sessions.

Reliability / Fault-tolerance – User experience is negatively affected if a play session suddenly breaks down due to server failure. Reliability is thus important to make NVE a service with quality.

The first three issues exist for all multiplayer virtual environments. Remaining ones are additional criteria for MMP applications. We consider scalability the most important issue if we plan to build truly massive worlds and applications, which millions of people can enjoy. Current approaches to scalability mostly include setting up multiple servers or server-clusters. However, maintaining server resources is costly and has inherent design limitation, as we will discuss in the next section.

This paper proposes a fully-distributed peer-to-peer (P2P) architecture, which attempts to solve the scalability problem based on the mathematical construct *Voronoi diagram* [4]. The main contribution of the paper is the proposal of a very simple and resource-efficient solution to the difficult scalability problem. Our solution dramatically reduces server load and can be achieved with a single lightweight server. In fact, the server only serves as an access point that provides authentication, and is not required after login. This will allow scalable NVEs to be built affordably.

In order to simplify the problem, two assumptions are made: (1) user computers can be trusted (2) message exchange itself is sufficient for maintaining the states of the world (i.e. user position, user actions, and temporary objects). In other words, currently we do not take security and persistency issues into consideration.

We will analyze the scalability problem in Section 2, and will present our design in Section 3. Our current implementation is described in Section 4. Finally, Section 5 concludes the paper.

2. THE SCALABILITY PROBLEM

2.1 Theoretical Analysis
Scalability is a phenomenon observed in many natural and artificial systems. We see systems that accommodate components (or nodes) in a wide range of numbers as being "scalable".

There are two main characteristics for scalable systems:

- **Joinability:** components or nodes may be added to the system.
- **Maintainability**: system remains functional after various nodes enter or leave the system.

Existing resources in any given system is usually finite, and are consumed at end-point when a new node is added. For example, bandwidth of existing routers is consumed when adding new routers. A system is "joinable" only when the accepting node has enough spare resources. Likewise, maintainability is sustained only when resource is not depleted after the new node joins. Two more properties exist to counter resource depletion:

- **Resource-growing**: useful system resources (i.e. resource at the accepting nodes) increase with the addition of new nodes
- **Decentralized end-point resource consumption**: addition of a node does not consume some "centralized" resource.

Resource-growing is a general strategy found in almost all scalable systems (reducing consumption works to the same effect). In the Internet, although additional router consumes bandwidth, it also contributes new resource to accommodate new routers. Decentralized resource consumption, on the other hand, is not necessarily required. As long as resources are available at the accepting node, system can be "joinable" and "maintainable" even if it is done in a centralized fashion. For example, in a server-cluster, as long as server resources (bandwidth and processing capability) can be increased, then scalability is maintainable [1]. However, most massively scalable systems (such as Internet) exhibit decentralized resource consumption.

From the above discussion, we expect that to build a truly scalable NVE, one that may accommodate more users by orders of magnitude than existing systems, we need architectures that can grow its resource, and does not require centralized resource when additional users join.

2.2 Previous Work

Scalability for NVE generally concerns with whether the system can accommodate a large number of simultaneous users. Various approaches have been taken, and generally fall into either the "increase resource" or the "reduce consumption" categories:

Increase Resource. Using multiple servers for multiple worlds or using server-cluster to maintain a single world has become a popular approach, especially for commercial NVEs [1] [14]. For example, commercial MMOGs are set up with multiple servers for the same game, each serving a pre-determined maximum number of users. When a server is full, it simply denies additional connections. Total number of players can be very large. (For example, a record of 160,000 concurrent users was reported for *Lineage* in 2002 in Taiwan.) However, users between different servers may not interact, and some systems do not even share user profiles. Server-cluster [3], on the other hand, divides the virtual world into *regions* or *zones*, and supports what appears to users as a single coherent world. Since server-cluster offers desirable properties, it has become the trend for building large-scale NVEs. However, server-centered approach is costly for server-side bandwidth, hardware, and maintenance, which limits the number of potential NVE developers.

Decrease Consumption. The central theme to this approach is *interest management* [12]. While other techniques to economize bandwidth exist, such as packet compression or aggregation [13], we consider interest management more relevant. Messages are generated by user actions and exchanged to maintain consistency [13]. However, if messages are sent to all other users, the amount of transmission and processing grows at $O(n^2)$, which is clearly not scalable. Real-world observation tells us that each individual only has a limited visibility or "sphere of interaction". In other words, our interest is localized [12]. Interest management therefore deals with relevant information filtering, to decrease unnecessary resource consumption while maintaining adequate interactivity. Early NVEs did not have interest management, and were set up by hosts broadcasting messages in the same LAN [13]. To provide interest management, later systems adopt the client-server model, where clients send messages to the server, which acts as interest manager and sends back filtered messages. Interest management can be based on various criteria. It can be distance-based (by geography), class-based (by object or user attributes), or some combination of both [12]. A commonly used concept is *Area of Interest* (AOI), which describes a circular or rectangular box centered on the user. Only messages generated within AOI are relevant to the user (Figure 1).

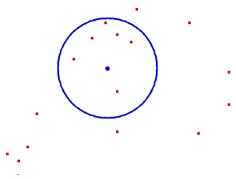


Figure 1: Each dot represents a user in the virtual world, and circle represents Area of Interest (AOI) of a particular user.

A common technique in interest management is to divide the world into various *regions*. Each user only receives messages (position update or interaction message) from relevant regions. This can be done by server-side message filtering, or via network-support such as multicast [10]. However, region size can be difficult to determine (Figure 2). If it is larger than AOI, irrelevant messages are still received; while if it is smaller than AOI, it becomes inefficient to maintain (e.g. subscribing to too many multicast address). Ideally, regions would dynamically adjust size and shape based on current user location. The real challenge then is to create *individualized* region that moves with the user.

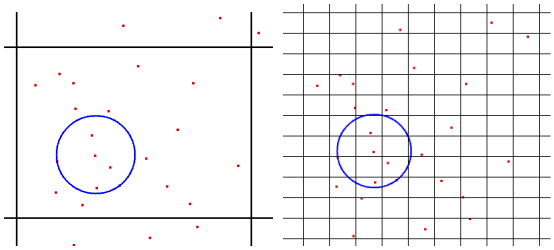


Figure 2: Difficulty in choosing region size. (L) AOI is smaller than region. (R) AOI is larger than region.

2.3 Promise and Challenge of P2P NVE

We consider the main challenge in scalability as: how to construct "resource-growing" and "decentralized consumption" into an architecture that also exhibits ideal interest management?

Peer-to-peer architecture naturally comes to mind, and appears to be an attractive alternative to client-server. Each participant contributes resource to maintain the system without consuming any

centralized resource. It matches with our criteria nicely. P2P architecture can be made very efficient if we only connect to relevant users (i.e. those within the AOI). (Keller and Simon describe this property as keeping *Local Awareness* [6].) A number of P2P overlay networks have been proposed in recent years: CAN, Chord, Pastry [8], and Hypercast [9], to name a few. However, these overlay networks mainly deal with setting up a distributed hash table (DHT) that maps keys to values and allows for content-lookup and retrieval (as in distributed file-sharing). While it is possible to build NVE on such overlay, as recently proposed by Knutsson *et al*. [8], there is overhead associated with using the overlay.

Common questions to all P2P networks are: correct topology maintenance and efficient content retrieval. Since a single node does not have knowledge of global topology or content location [7], these become difficult questions important to any P2P design. For topology maintenance, there are two issues to consider: whether it is *fully-connected* (described by Keller and Simon as the *Global Connectivity* property [6]) and whether all nodes have a *consistent view* of the topology. Unlike file-sharing P2P, where the desired content changes with user preference randomly and unpredictably, for P2P NVE the desired content is easier to identify -- messages generated by other users within the AOI. If only such messages are received, then message flow is managed optimally. So the "content discovery problem" for P2P network translates naturally to a *neighbor discovery problem* in P2P NVE.

Neighbor discovery is challenging, because there is a paradox in maintaining consistency in decentralized systems, as described by Makbily *et al*. [11]. At least three recently published papers offer different solutions: (1) Knutsson *et al*. describe P2P support for Massively Multiplayer Games by using Pastry and Scribe, a P2P overlay and its associated simulated multicast [8]. The virtual world is divided into regions of fixed-size. Each region is managed by a promoted node called *coordinator*, which serves as the root of a multicast tree. Users inside the same region subscribe to the address of the root node to receive updates from other users, so neighbors are discovered via the coordinator. Coordinators maintain links with each other, facilitating user transition to other regions. (2) Kawahara *et al*. describes a fully-distributed scheme where each user keeps track of a fixed number of nearest neighbors [5]. Nodes constantly exchange neighbor list with their own neighbors. After sorting through the list by distance, each node may learn of new nodes and update existing links. (3) Solipsis [6] is also a fully-distributed system, where each node attempts to link with all the nodes within its AOI. Neighboring nodes serve as the "watchmen" for any approaching foreign nodes. Neighbor discovery is done by notification from known neighbors.

However, each of them incurs some undesirable properties. In Knutsson *et al*., since fixed region size does not reflect true AOI, users cannot see across regions. If users decide to listen to more regions, as suggested in the paper, unnecessary messages beyond AOI will be received. A more serious problem is the performance penalty incurred by using P2P overlay. As the overlay does not consider AOI, messages may need to be relayed by other nodes (1 to 2 hops for most cases, but in some cases it goes beyond 50. Note too that this is "virtual hop", so more delays happen at the physical level). In short, the architecture does not fully utilize the power of direct connections. In the Kawahara *et al*. approach, direct links are maintained between neighbors, so hop-count is most efficient (e.g. one virtual hop). However, constant exchange of neighbor list incurs network overhead (if 10 nearest neighbors are kept, one exchange requires receiving updates of 10x10 nodes). The more serious problem is keeping the topology connected. Since only a finite number of nearest neighbors are maintained, groups of users may lose contact to each other if separated by a large distance. The underlying overlay can thus separate into isolated parts [5]. Solipsis also uses direct links among neighbors (there is no relay). Additionally, it requires that each node be inside a convex hull formed by its neighbors in 2D plane. This way the topology is guaranteed to be fully connected (i.e. *Global Connectivity* is kept). However, inconsistent topology may happen during normal operation (though rare), since an incoming node may be unknown to directly connected neighbors, proper neighbor discovery is not guaranteed (i.e. *Local Awareness* is not kept, see Figure 3).

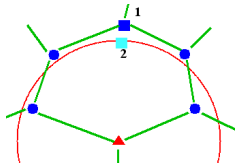


Figure 3: Undiscovered node in Solipsis. Lines are connections. Square node is not discovered as it moves from position 1 to 2. Topology is inconsistent though fully-connected.

3. VORONOI-BASED P2P NVE

3.1 Design

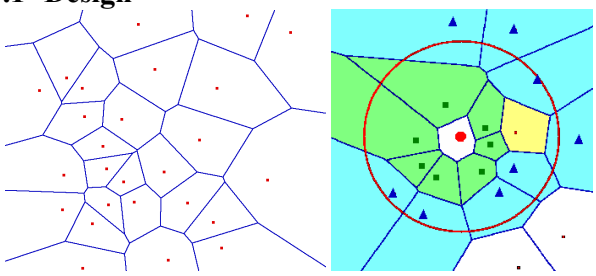


Figure 4: (L) Voronoi diagram. (R) Square (□): *enclosing neighbors*, triangle (Δ): *boundary neighbors*.

In this section, we will explain the design and analysis of our P2P approach, which is based on a well-studied mathematical construct *Voronoi diagram* [4]. Given *n* points on a plane (each point called a *site*), a Voronoi diagram is constructed by partitioning the plane into *n* non-overlapping *regions* that contain exactly one *site* in each *region*. A *region* contains all the points closest to the *region*'s *site* than to any other *site* (Figure 4L). The entire plane is therefore divided into arbitrary sizes in a deterministic way. Voronoi diagram can be used to find the *k*-nearest neighbors of a specific *site*. By using Voronoi, we may be able to identify *enclosing* and *boundary neighbors* for a given node. *Enclosing neighbors* are defined as *regions* that share a common edge with a given node's own *region*. *Boundary neighbors* are defined as *regions* that overlap with the node's AOI boundary (Figure 4R). Note that an *enclosing neighbor* can also be a *boundary neighbor*. These properties will help to solve the neighbor discovery problem described earlier.

The basic idea of our approach is to let each node construct and maintain a Voronoi diagram, based on the spatial coordinates of neighbors within the node's AOI. Each node keeps P2P connections with all neighbors that constitute the Voronoi. Connections are therefore based on spatial relationship in the NVE (not physical

network proximity). In our basic model, we assume that all AOIs are of the same radius, and are determined in an application-specific manner by the designer. Although a node only knows about a limited number of neighbors, it can learn of other new neighbors with the help of its *boundary neighbors*. Each peer serves as the "watchman" for one another in discovering approaching neighbors.

When the node moves, position updates are sent to *all* neighbors recorded in the Voronoi. If the receiver is a *boundary neighbor* (as determined by the sender), an overlap-check is performed. The receiver checks if the mover, with its new AOI, would enter into any of its *enclosing neighbors'* Voronoi region. The receiver only notifies the mover if a *new* overlap occurs (i.e. previously non-overlapped region becomes overlapped). This allows the moving node to get aware of potentially visible neighbors outside the AOI with minimal network overhead (only normal movement message is used). In case of a node leave or failure, its neighbors simply update their Voronoi after detection (through a loss of TCP connection or inactivity timeout). If the leaving user is considered a *boundary node*, queries are sent to discover any replacement (Figure 7).

3.2 Procedure

We will describe the basic procedures for joining, moving and leaving in the P2P NVE. The emphasis of these procedures is to maintain P2P topology consistency in a message-efficient manner.

Join

1. *Joining node* contacts the *gateway server* for a unique ID.
2. Join request is forwarded to *acceptor region* (defined as the region that contains the joiner's coordinates) via neighboring nodes with simple greedy forward (see Figure 5L).
3. *Acceptor node* sends back a complete list of its own neighbors.
4. *Joining node* contacts each neighbor on the list.
5. *Joining node* builds up a new Voronoi while other nodes update their Voronoi to accommodate the *joining node* (see Figure 5R).

Move

1. *Moving node* sends position coordinates to all neighbors (i.e. *boundary*, *enclosing*, and other neighbors). Messages for *boundary neighbors* are specifically marked.
2. *Boundary neighbor* will check if the *moving node*'s new AOI becomes overlapped with any of its *enclosing-neighbor's* Voronoi regions. If so then it sends a notification. (Figure 6L).
3. If a new neighbor is found, the *moving node* connects to it.
4. *Moving node* disconnects any *boundary neighbors* whose Voronoi region no longer overlaps with its AOI (Figure 6R).

Other actions (jump, chat, trade)

1. Send message to relevant neighbors recorded in the Voronoi. (For example, a private chat is directed only to neighbor(s) in the conversation, but an action such as "jump" is sent to all neighbors that can see the action, i.e., those in the Voronoi.)

Disconnect/Leave

1. *Leaving node* notifies with a list of its *enclosing neighbors.*
2. Neighboring nodes affected by the disconnection update their Voronoi. If the leaving node is seen as a *boundary neighbor*, then new *boundary neighbors* may be assigned.
3. For abnormal departure of *boundary neighbor*, a request for *enclosing neighbor* list is sent to known neighbors to ensure that topology remains consistent (see Figure 7).

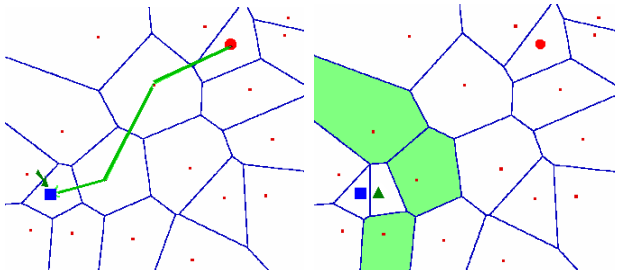


Figure 5: Join procedure. (L) Forward of join request. Circle is *gateway server*. Arrow indicates the *acceptor node.* (R) Triangle is the new node, shaded regions are neighbors affected by join. Note that the effect is localized.

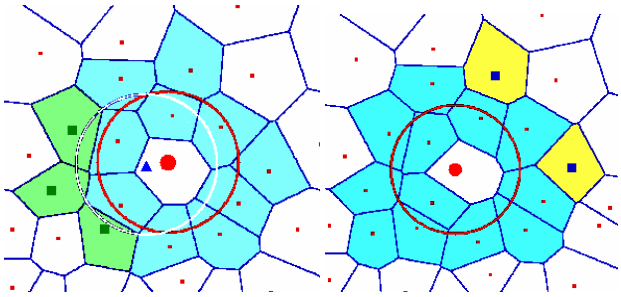


Figure 6: Move procedure. (L) Triangle indicates the intended new position. Squares are new neighbors about to be discovered. (R) After the move. Squares are the neighbors no longer overlap with AOI, therefore are disconnected.

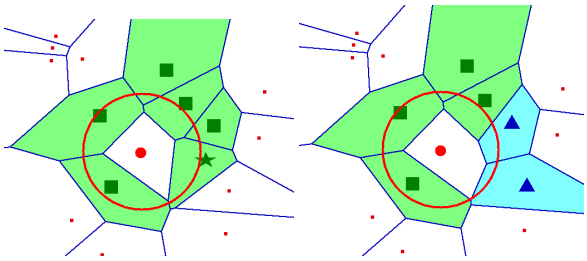


Figure 7: Leave procedure. (L) Before node leave, star is the leaving node. (R) After node leave, triangles (Δ) are the new boundary neighbors discovered with help of existing neighbors.

3.3 Analysis

A qualitative analysis of our current design is given below:

Consistency: In our design, as long as each node correctly keeps track of at least their *enclosing neighbors,* there is a guaranteed path between any two nodes; since discovery is covered in all directions, no node would be missed. P2P topology is therefore both *fully-connected* and *consistent* provided there is no network failure. Even if a small number of node fails, the network can still self-repair. This is an important improvement over existing approaches. Consistency in event synchronization is not currently guaranteed, because no "central authority" exists to decide the ordering.

Performance: Transmission hop-count between peers is optimally efficient as there is no relay. The low latency quality allows for responsive applications. When users are close to each other, *ideal interest management* is achieved (i.e. only messages within the AOI are received). However, if users are dispersed, connections with enclosing neighbor beyond AOI are needed to maintain the

topology (Figure 8L). Luckily, on average such neighbors are few, given Voronoi's characteristics (six on average [9], but *n*-1 neighbors in the worst-case, see Figure 8R).

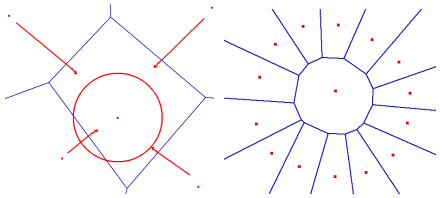


Figure 8: Potential issues with Voronoi. (L) Messages outside of AOI are still received to maintain P2P topology, but the amount is expected to be small. (R) Circular line-up of nodes.

There are two inherent disadvantages in the current design. (1) Each node must send duplicate messages to reach the neighbors, which requires more bandwidth than client-server (i.e. only one message is sent). (2) Since no message is processed centrally, aggregation or compression techniques cannot be leveraged.

Security: We assume all hosts can be trusted.

Scalability: The architecture matches the two criteria for scalable systems: resource-growing and decentralized consumption.

Persistency: We assume no persistent states in the current model.

Reliability: As long as each node maintains *some* reliable neighbors, the system should be able to self-repair inconsistency due to node failures. However, if a large number of nodes fail simultaneously, the P2P overlay may still be separated into mutually unaware parts. However, this is a general problem faced by all P2P networks that warrants future study.

4. IMPLEMENTATION

There are a number of existing algorithms for constructing and maintaining Voronoi diagrams [4]. The particular one we implement is Fortune's sweepline algorithm [2], which constructs Voronoi in O(n log n) time. Our design is currently implemented using High-Level Architecture Run-Time Infrastructure (HLA-RTI) interface, an IEEE standard originally proposed by the U.S. Department of Defense. We are investigating the suitability of the standard for MMOG applications. We name the current implementation Adaptive Scalable Cooperative Environment for Networked Dimensions (ASCEND). Our long-term goal is to develop an open source platform for NVE development.

5. CONCLUSION

We have presented a general picture of the scalability problem in NVE, and have analyzed requirements for potential solutions. A promising solution for the difficult neighbor discovery problem is also presented, by using Voronoi diagram. The general idea of our solution is to leverage knowledge of each peer about its neighbors to maintain the position states of all participants. Our solution is simple, efficient, and close to ideal interest management in NVE. Future works include variable-size AOI, persistency maintenance and security mechanism under P2P.

One of the most important concerns for commercial NVE is security, both for account information and game state authenticity (e.g. player's experience points, valuable virtual items). While accounts can be handled by a central server, user computers are always prone to hacking. This is the main reason why client-server is almost universally adopted. We feel that security might indeed be the main obstacle for commercial adoption of P2P, despite benefits in performance and cost. However, we believe that active research can be done to find "good enough" solutions. On the other hand, this "weakness" can be opportunity for other areas, such as education or social communities, where social interactions and collaborations are emphasized over competitions. One major feature of P2P NVE is its low cost, where scalability and performance is achieved affordably. For developers with limited budget, P2P NVE provides a promising alternative.

6. ACKNOWLEDGMENTS

We thank members of the WISE Lab in Tamkang Univ. and Joaquin Keller for valuable feedbacks, LSCP, Inst. of Physics, Academia Sinica for their facility, and Dr. Tzu-yang Chen for proofreading. We are very grateful of the helpful comments by the anonymous reviewers. We would also like to thank Prof. Wen-Bing Horng and Dr. Chin-Kun Hu for supporting this work. Special thanks to Prof. Jiung-yao Huang for introducing NVE to us.

7. REFERENCES

[1] Butterfly.net, Inc. The Butterfly Grid, 2003. www.butterfly.net/platform

[2] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmic* 2, Pages 153-174. 1987.

[3] T. A. Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments. In *Proc. of the 1995 Symposium on Interactive 3D Graphics*. pp. 85-92, Apr. 1995.

[4] L. Guibas, J. Stolfi. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. *ACM Trans. on Graphics*, Vol. 4, No. 2, Pages 74-123, April 1985.

[5] Y. Kawahara, T. Aoyama, H. Morikawa. A Peer-to-Peer Message Exchange Scheme for Large-Scale Networked Virtual Environments. *Telecommunication Systems* Vol. 25 Issue 3, Pages 353–370, 2004

[6] J. Keller, G. Simon. Solipsis: A Massively Multi-Participant Virtual World. In *Proc. of PDPTA 2003*. Pg. 262-268. 2003

[7] H. T. Kung, C. H. Wu. Hierarchical Peer-to-Peer Networks. Technical Report IIS-TR-02-015, Institute of Information Science, Academia Sinica, Apr., 2001.

[8] B. Knutsson, H. Lu, W. Xu, B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. In *INFOCOM*, Mar. 2004.

[9] J. Liebeherr and M. Nahas. Application-layer Multicast with Delaunay Triangulations. In *Proc. of IEEE GLOBECOM*, Nov. 2001.

[10] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, P. T. Barham. Exploiting Reality with Multicast Groups. *IEEE Computer Graphics and Applications*, Volume 15, Issue 5, Pages: 38 - 45, Sept. 1995.

[11] Y. Makbily, C. Gotsman, R. Bar-Yehuda. Geometric algorithms for message filtering in decentralized virtual environments. In *Proceedings of the 1999 Symposium on Interactive 3D graphics*. Pages: 39 - 46. 1999.

[12] K.L. Morse. Interest Management in Large-Scale Distributed Simulations. Tech. Report ICS-TR-96-27, UC Irvine, 1996.

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

[14] Zona Inc. Terazona: Zona application frame work, 2003. www.zona.net/whitepaper/Zonawhitepaper.pdf.