PHANEROS: VISIBILITY-BASED FRAMEWORK FOR MASSIVE PEER-TO-PEER VIRTUAL ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ŞAFAK BURAK ÇEVİKBAŞ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE/DOCTOR OF PHILOSOPHY IN
COMPUTER ENGINEERING

JUNE 2017

Approval of the thesis:

PHANEROS: VISIBILITY-BASED FRAMEWORK FOR MASSIVE PEER-TO-PEER VIRTUAL ENVIRONMENTS

submitted by **ŞAFAK BURAK ÇEVİKBAŞ** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Date: 22.06.2017

ABSTRACT

PHANEROS: VISIBILITY-BASED FRAMEWORK FOR MASSIVE PEER-TO-PEER VIRTUAL ENVIRONMENTS

Çevikbaş, Şafak Burak
Ph.D., Department of Computer Engineering

Supervisor: Prof. Dr. Veysi İşler

June 2017, 98 pages

Contemporary distributed virtual environments are growing out of terabytes of 3D content and hundreds of thousands of users. Server-client architectures have become inadequate for fulfilling the scalability requirements. The peer-to-peer architectures provide inherently scalable, cost-effective distributed solutions for distributed virtual environments. We present a fully distributed peer-to-peer framework, Phaneros, which is capable of providing necessary means to realize more efficient and more scalable massive distributed virtual environments. Using the presented visibility aware interest management, Phaneros performs better than existing overlays, achieving single hop update dissemination while having lower bandwidth requirements. The provided visibility aware 3D streaming scheme distributes 3D content more efficiently without creating any significant load on the server. Our test results show significant improvements over existing frameworks.

Keywords: Distributed Virtual Environments, Peer-to-Peer, 3D Content Streaming, Update Dissemination, Potentially Visible Sets

ÖZ

PHANEROS: KİTLESEL SANAL ORTAMLAR İÇİN EŞLER ARASI GÖRÜNÜRLÜK TEMELLİ ALTYAPI

Çevikbaş, Şafak Burak
Doktora., Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Veysi İşler

Haziran 2017, 98 sayfa

Günümüzde çağdaş dağıtık sanal ortamlar terabaytlarca 3B içerik ve yüz binlerce kullanıcıyı barındıran büyüklüklere ulaştılar. Sunucu-istemci mimarileri gerekli ölçeklenebilirliği sağlamada yetersiz hale geliyorlar. Eşler arası mimariler doğaları gereği ölçeklenebilir ve uygun maliyetli dağıtık çözümler sunarlar. Bu çalışmada, daha verimli ve daha ölçeklenebilir kitlesel dağıtık sanal ortamlara imkân veren, tamamen dağıtık çalışan ve eşler arası bir iş çerçevesi olan Phaneros'u sunuyoruz. Phaneros, görünürlük temelli ilgi alanı yönetimi sayesinde hem daha düşük bant genişliği gerektirerek hem de tek atlamalı güncelleme yayma yeteneği sunarak var olan diğer örtü ağlarından daha yüksek başarım sunar. Phaneros'un sunduğu 3B akar düzeni, 3B içeriği; kullandığı görünürlük iyileştirmesi sayesinde sunucular üzerinde kayda değer bir yük oluşturmadan ve var olan yapılardan daha verimli şekilde yayar. Test sonuçlarımıza göre Phaneros, var olan diğer iş çerçevelerine göre dikkate değer seviyede iyileşme sağlar.

Anahtar Kelimeler: Dağıtık Sanal Ortamlar, Eşler Arası, 3B İçerik Akışı, Güncelleme Dağıtımı, Potansiyel Görünür Kümeler

Büyük Atatürk’e…

ACKNOWLEDGEMENTS

The author of this study presents his gratefulness to:

… Prof. Dr. Veysi İŞLER for his innovative guidance, criticism and encouragement;

… Nazmi ÇEVİKBAŞ, Nurten ÇEVİKBAŞ and Pelin ÇEVİKBAŞ for their support, motivation, and patience;

… Gürkan KOLDAŞ for his guidance;

… Umut KAHRAMANKAPTAN for his enthusiasm.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A Distributed Virtual Environment (DVE) is a virtual environment or a virtual world where multiple users are connected to a network to share a common experience by interacting with each other and their surrounding environment. The concept emerged for military purposes in the 80s and evolved into today's interactive applications[1]. DVEs have become more prevalent in the last two decades starting with online games and later evolving into massively multiplayer online games (MMOGs). As of 2011, Second Life reached 16 million, and World of Warcraft reached 12 million registered users. [2]

A virtual world is composed of the environment, the users, the non-player characters, the collectible/usable objects and interactions of these. The environment is represented by 3D visual content with sounds, music, and other auxiliary material. Usually, the 3D content of the virtual world is distributed in physical installation media (DVD) or downloaded at once before installation/use of the application. In the statically pre-assembled 3D world, participants move and interact with each other and the environment. The interactions are encoded as update messages and disseminated to all interested participants through the underlying network. An update message contains all the information about the interaction that is necessary for the receiver to reflect the change caused by the interaction to its local copy of the world. The receiver might be a server which in turn redistribute the results of the interaction to its clients. With many participants, DVEs can generate large amounts of network traffic and computational load [2].

Because of the reasons that are going to be detailed later, most of the popular DVEs rely on client-server architectures [2][3][4]. In a server-client setting, the game state is maintained centrally on the server. Clients send their actions to the server where the actions are evaluated, and the resulting changes in the game state are disseminated back to the clients. When the number of users becomes massive, high-performance server clusters provide the required computational power. Servers carry the load by either instantiating limited sized instances of the virtual world or partitioning the larger world into smaller parts statically or dynamically [5]. The users, the objects, the functions or the virtual world may be subject to partitioning [1]. Centralized structure of the server-client architectures eases the management at the expense of lower scalability and higher costs. Operators have to maintain more than enough resources under normal circumstances to handle the loads that may occur when larger numbers of users become simultaneously online. Recently, cloud architectures enabled better resource planning by allowing on-demand allocation of resources. However, the scalability limitations are not avoided even with the cloud-based approaches. Required bandwidth and computational power continue causing significant costs to the provider when the number of users is high [6]. Peer-to-peer (P2P) studies explore solutions that lower server requirements by distributing the state management to peers instead of bringing it together on the central server(s).

Contemporary DVE applications are downloaded as one big package containing all of their content. However, the trend of the DVE applications and the demand of the users are in the direction of open worlds with terabytes of content. Downloading such large amounts before accessing the virtual world, even in a progressive manner, is not convenient for the user. Therefore, the content should be streamed on demand. Streaming 3D content has different requirements than streaming sequential multimedia content. Participants of a DVE do not follow a defined sequence for requesting chunks of 3D data [7]. Requests are made from multiple simultaneous users scattered in the virtual world moving erratically and interacting with each other unceasingly. Consequently, streaming 3D data from a single source for the consumption of multiple clients becomes inapplicable because of the unpredictable nature of the demand. Servers have to manage each user separately and consequently have to carry excessive bandwidth and CPU load. 3D streaming is not feasible with client-server architectures because of the sizes of the DVEs and the massive number of users. Peer-to-peer 3D streaming studies investigate the necessary facilities to transfer 3D content seamlessly between the peers on demand without the intervention of a server.

Virtual worlds have grown to massive sizes, comprising terabytes of 3D content [8] and serving millions of people, generally tens of thousands simultaneously [9]. Client-server architectures are inherently limited regarding scalability thus being inadequate for meeting the increasing demand. It has become impractical neither to distribute all 3D content at once nor to connect all participants to central servers simultaneously. Peer-to-peer architectures are inherently scalable and suitable for both update dissemination and 3D content streaming; hence, they are studied to overcome the drawbacks of the client-server architectures.

In this study, Phaneros, a fully distributed peer-to-peer framework for building massive DVEs is presented. Phaneros provides high-performance update dissemination and efficient 3D streaming by incorporating visibility based interest management with peer-to-peer overlay networks for DVEs. Utilization of visibility allows accurately identifying source peers for streaming and avoiding redundant direct connections for update dissemination.

Phaneros is a Greek word that has the meanings like visible, apparent, obvious, clear. Phaneros is a visibility based framework and has a simple structure. Because of these reasons, we believe that the word Phaneros accurately reflects the characteristics of the proposed framework.

The organization of this work is as follows: Chapter 2 covers related work about update dissemination and 3D content streaming. Chapter 3 describes the internals of Phaneros. Chapter 4 describes the implementation details and the tests cases. Chapter 5 presents the results obtained and our evaluation of the results. Chapter 6 closes the work with concluding comments and openings for further research.

CHAPTER 2

RELATED WORK

Phaneros is the output of a multi-disciplinary approach which comprises peer-to-peer (P2P) overlay networks, application layer multicasting, visibility culling and distributed virtual environments. Therefore, this chapter purposes providing a foundation for Phaneros by covering all four topics. The discussion starts with distributed virtual environments (DVE) and their design considerations. The chapter continues with information about server-client DVEs and P2P DVEs. VON and FLoD are discussed in detail as they together from the counterpart for Phaneros. Following these, P2P overlay networks are explained with focus on CHORD; then application layer multicasting is investigated with a focus on SCRIBE. Finally, visibility culling methods are explained briefly to provide an understanding of the concept.

2.1 DISTRIBUTED VIRTUAL ENVIRONMENTS

A Distributed Virtual Environment (DVE) is a system that connects multiple users into a shared virtual environment where the users are represented by avatars and they interact with each other and the surrounding environment by the actions that their avatars perform in the virtual environment.

For a successfully shared experience, there are some qualities to be considered while building DVEs. DVEs should be consistent, responsive, scalable, reliable, persistent and secure [1][9][10]. These characteristics shape how a DVE is designed and operated.

DVEs become attractive to the users and the researchers when the number of users rises. They become even more attractive when they are massive. **Scalability** is the capability of supporting a high number of users without compromising on the qualities that are available for a smaller number of users. With this definition, scalability is the result of all other qualities [6]. When some of the qualities explained below are not available at required levels for higher number of users, the system is not considered as scalable. A successfully designed DVE yields a linear relation between the number of users and the required computational capacity. A poor design, on the contrary, may cause an exponential increase of required resources with increasing number of users [10]. Scalability requirements of DVEs are also application specific.

Presumably, the most important characteristic of a DVE for the users is **consistency**. Consistency can be defined as the capability of representing the same world state to all users within an acceptable margin of latency [11]. Consistency is a defining factor of a shared experience. All participants should perceive the same state of the environment and the same events consistently [10]. Although the level of consistency depends on the application requirements and the technical limitations, for a DVE to be persuasive, all users are expected to perceive the same state with inconspicuous time disparities. The tolerable latency is determined by the application requirements. Strict consistency, eventual consistency or inconsistency may be appropriate for the application requirements [10]. Maintaining strict consistency at every node may require complex algorithms and may hinder responsiveness. Consistency is also crucial if the virtual environment is a competitive one. A high level of consistency is a precondition for fairness. When inconsistencies are not handled properly, they may cause visual divergence, causality violation and expectation violation [1]. Finally, it is hard to achieve strict consistency with network latency. When it is not possible to avoid inconsistencies, predictive methods like dead reckoning and bulk state synchronization mechanism are employed to correct at least the effects of inconsistencies [2]. Another type of inconsistency is the one caused by cheating. Cheating is discussed below the topics of security.

Responsiveness defines the delay between a user's actions and their observable consequences [9]. Shorter response time means higher responsiveness.

Responsiveness requirements are subject to application characteristics. Seconds may be tolerated for a turn-based game or a social application whereas hundred milliseconds may not be tolerated for a fast-paced action game. Although higher responsiveness is the desired quality, it acts as a trade-off for some of the other characteristics. An improvement on consistency, reliability or security probably introduces a barrier for responsiveness. Responsiveness of is also dependent on the number of nodes, complexity of the virtual environment and frequency of the events occurring.

For an application with hopefully hundreds of thousands of simultaneous users, **reliability** is a major concern [9]. The failure of the application makes so many users unhappy at the same time that the service provider may face a severe financial and reputation loss. Therefore, DVEs should be reliable and fault tolerant. Failure of a single node (server or client) should not be allowed to affect more than a limited group of users, or if possible, it should not affect anybody at all. Another related term is fault tolerance. If node failures cannot be avoided, the DVE system should be designed to endure possible node failures. When a distributed system gets larger, the chance of failure also increases [10]. Reliability requires replication of data and hand-over procedures in case of failure. The hand-over should be fast and deterministic.

Persistency is the continuation of the game state on the absence of some or all nodes. Neither a maintenance shutdown nor a joining or leaving of a user should interrupt the persistence of the virtual world. [6] When a user leaves the DVE and comes back after a while, a continuation of the virtual environment should be available. Persistency is also critical for maintenance and recovery from severe failures. The chosen persistency method is dependent on the architecture of the DVE. Central systems persist on servers whereas distributed systems provide more complex means of persistency by using dedicated servers, super peers and user computers [2].

Security is a challenging quality of DVEs. Multiple, potentially untrustworthy and competing users come together in a DVE. Each one has a potential to abuse the distributed nature of the environments and escape the fairness rules. Some means of authorization and necessary monitoring tools should be incorporated in DVEs [1]. Cheating occurs when a user creates updates that go against rules. Users may exploit bugs and errors that are prone to exist in complex systems. Cryptography, two-phase commits, verification computations and referees are available methods for cheating prevention in DVEs. Encrypted messages prevent sniffing and illegal message modifications. Two-phase commits ensure that all users are notified of the new game state at the same time. Verification computations ensure that the same state has been reached by independent parties. Authenticated referee nodes continuously audit the game state and the messages to ensure that there is no cheating going on. A classification of cheating and counter measure methods are available in [2].

The operation of a DVE is realized by two core functions; the audio-visual representation of the virtual environment and dissemination of the changes on the environment (updates) that are generated by the user actions. Both functions become design challenges when there are many simultaneous users.

3D content must be delivered to user computers for rendering. The delivery may be offline or online. Offline methods are not in the scope of this study. Downloading the 3D content is also considered as offline. In terms of streaming there is no difference between delivering a DVD or downloading a package of several gigabytes at once. Online delivery of 3D content is called 3D content streaming and has its specific challenges. The 3D content cannot be treated like sequential video content. Users move virtually random in the virtual world and continuously make requests for dispersed content. Prefetching and caching techniques are commonly used ways of efficiently streaming 3D content [7][12].

The other main function of a DVE is disseminating the user actions and events to interested parties as updates. Any coordinating node or a user whose area of interest covers the location of the change should receive information about a change in the environment. The update can be a position update, a user action, a local happening or a global state change.

Neither 3D content streaming and nor update dissemination is not feasible without area of interest (AOI) management. By definition, streaming is about delivering only the necessary content on demand. Determination of the part whose 3D content is required by a user is realized by AOI management methods. Delivering all updates to all clients gets exponentially more expensive with increasing number of users. Filtering out unnecessary update messages is also realized by AOI methods. AOI management methods are central to all scalable DVE systems [4]. The fundamental assumption of AOI management is that although the users are participating in the same virtual environment, they do not need full awareness of it. Each user only needs to be aware of a much smaller part, usually its spatial proximity [4].

A concept closely related to AOI management is partitioning. When the computational load is high, the load is separated into smaller more manageable parts by the use of a partitioning strategy. Like AOI methods, partitioning may use spatial proximity in the virtual world. The environment is partitioned dynamically or statically into regular or irregular regions, and responsibility of these regions are assigned to different computers. Another partitioning may be on functions. A computer may carry out the scoring function while another performs the physical computations and another handles the inventory management. Another level can be added by classifying objects and assigning ownership of each class to another computer. [1][5]

Following sections covers server-client and peer-to-peer architectures for DVEs with their contained management methods.

2.1.1 SERVER-CLIENT ARCHITECTURES

Server-client architectures are defined by their central management of the virtual environment, not by the number of computers assigned for the central computation. With this approach, clients send messages to the server as input for computation of the next state. The server receives messages from clients, computes new state and sends back the resulting state change as updates to clients. All of the state maintenance is centrally handled on the server. Basic server-client flow is illustrated in **Figure 1**.

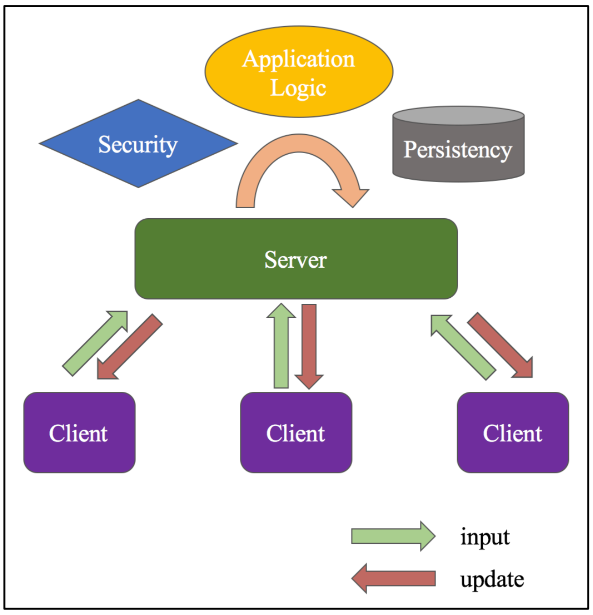

Figure 1 Basic server-client architecture: Clients send messages to the server as input for application logic. The server gets inputs through security, application logic and persistence then sends updates messages back to clients.

The explained flow introduces a theoretical limit on the propagation time. For a client to see consequences of its actions, a message should travel to the server and back to the client. For fast paced applications, this limitation is partially overcome by predicting the events and correcting any inconsistencies after receiving the update from the server.

It is simpler to provide consistency, security, and persistence with explained basic server-client architecture. There is a single computer that is responsible for everything about the virtual world. Application rules are securely applied by the server without any intervention from clients. There is a single source of state updates; therefore, all clients perceive the same state of the world consistently. Persistence is straightforward since everything is centrally maintained in a single place. However, reliability is a concern because the server is a single point of failure. When the server is not available, the application is not available at all. Scalability is limited by the resources on the server computer and responsiveness is relatively limited with the two-way communication between clients and the server.

Scalability and reliability issues of server-client architectures are solved with multi-server architectures by adding enough computational power for increasing number of users and replication [5][1]. A number of load balancing methods were developed for distributing the load among servers. These can be classified as zoning, mirroring, instancing and functional partitioning [1][13]. Zoning is based on dividing the world into smaller regions and assigning a different server for each region. The regions may be statically defined or dynamically formed according to the load on the regions and usage statistics. Inter-region interactions require communication between servers. Mirroring is assigning more than one server to the same region when one is not sufficient for the required traffic and computational load. The assigned servers have to communicate with each other to be in synchronization. Instancing is running multiple smaller instances of the world independently on different servers. Although instancing is limiting the users with a smaller virtual world regarding participants rather than providing true scalability, it is the most widely-used way of supporting a massive number of users. This trend is the result of the fact that in a virtual environment, users interact only a small portion of the other users at the same time, so it is possible to create the illusion of a larger world with instancing. As distinct from the others, instancing does not require inter-sever communication. Functional partitioning is assigning different servers for different aspects of the virtual world. However, it is not sufficient for scalability on itself because apart from the other aspects, the interaction between a massive number of users is the core aspect of a distributed virtual environment that needs scaling. Load balancing of multi-server architectures is illustrated in **Figure 2**.

Even with a multi-server approach, it is neither feasible nor necessary to inform all users about all changes in the virtual world. AOI methods are used for filtering unnecessary network traffic between the servers and the clients. Clients are only informed about their spatial proximity. No messages are sent to distant clients who are irrelevant for the update.

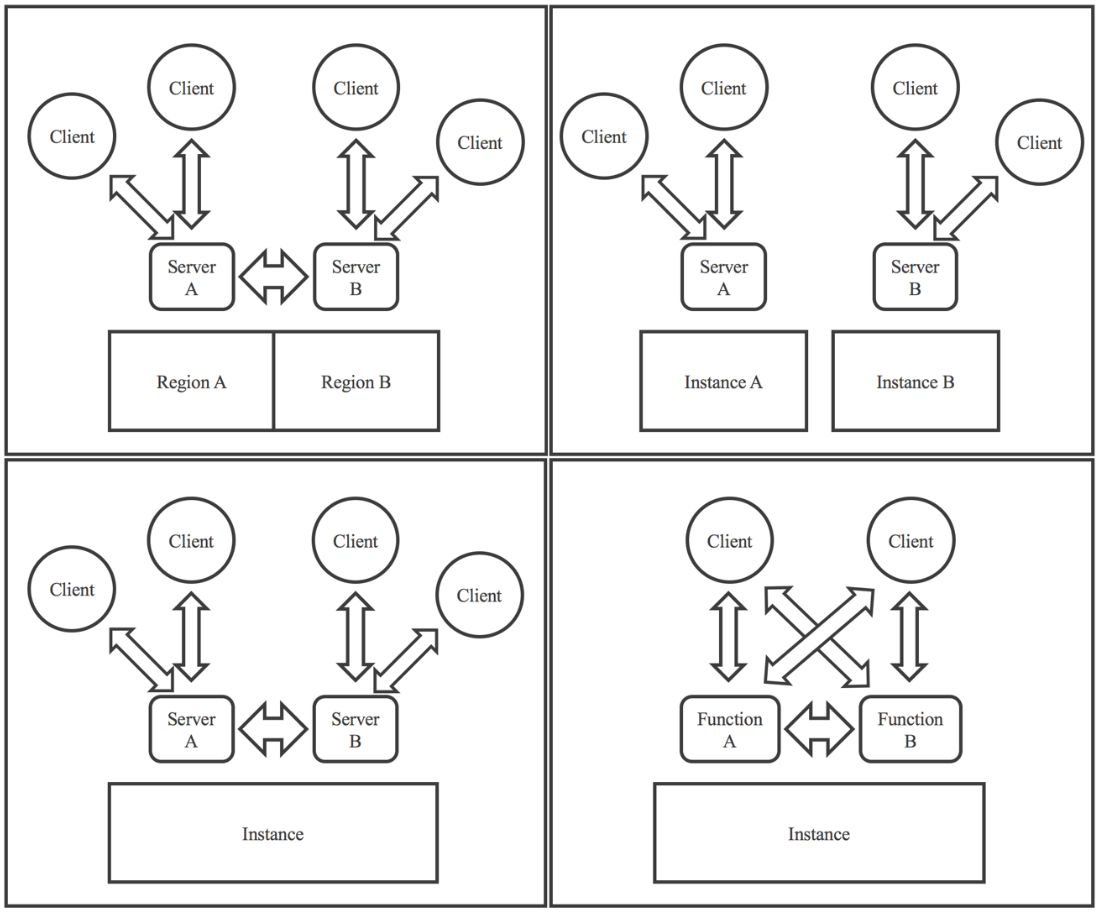

Figure 2 Multi-server load balancing: Zoning is shown top-left. Instancing is shown on top-right. Mirroring is shown on bottom-left. Functional partitioning is shown on bottom-right.

Server-client architectures do not provide any facility for 3D content streaming. By definition of the architecture, like update dissemination, content delivery is realized centrally by streaming all of the content from servers. Interest management methods used for update dissemination are also applicable for 3D content streaming. Clients fetch required portions of the virtual world from the servers that are responsible for

3D streaming and the related region. Multi-resolution modeling, compression, delta encoding, prefetching and caching are methods of streaming optimizations. They are not unique to server-client architectures. The mentioned methods define how the content is delivered effectively. Streaming the 3D content from other sources other than the servers is not in the scope of these methods.

2.1.2 PEER-TO-PEER ARCHITECTURES

A peer-to-peer (P2P) network is a distributed overlay network where participating peers operate independently and asynchronously from other peers without the existence of a central coordination unit. P2P architectures are inherently scalable and reliable. Peers make available their resources like bandwidth, CPU time and storage to the use of other peers on the network to share the computational load instead of central servers [1].

Peer-to-peer approaches are classified according to the incorporated interest management schemes, layout structures, communication protocols and load balancing methods [14]. Spatial partitioning is used for organizing the layout network and managing the area of interest in load balanced structures. Voronoi diagrams are a common spatial partitioning method used for both purposes [15]. When used for managing the layout, Voronoi diagrams have the disadvantage of bounding the layout management is to the positions of the nodes on the virtual world. Optimizations that reduce the number of connections by avoiding some connections cannot be applied because all connections are also required for the operation of the layout. Communications protocols defines how the update messages are transmitted between the peers. Direct connections (Mutual Notification), application layer multicasting, super peers and hybrids of these are communication protocols used by P2P DVEs. Static and dynamic load balancing strategies are both utilized by P2P DVEs [16]. Layouts networks used by P2P DVEs are generally structured layouts. The determinism and reliability provided by structured layout networks is necessary for P2P DVEs [14][15].

2.1.2.1 ADVANTAGES OF PEER-TO-PEER DVES

One of the advantages of P2P DVEs is the reliability that is inherently provided by peer-to-peer architectures. Although reliability comes with some additional concerns like replicating the data that is relevant for the nodes other than the failing nodes; the expected behavior of a P2P system is being available even when there are failing nodes. P2P networks are capable of reorganizing itself and recovering network connectivity when there are failed nodes.

The other advantage of P2P DVEs is their extreme scalability [4][13][14] [17]. The computational capacity of a peer-to-peer DVE is proportional to the number of peers that participate in it. Each new peer brings more capacity with the extra computational cost is generates. The additional capacity is generally assumed to be larger than the extra burden. P2P DVEs also provides flexibility alongside scalability. The alterable communication schemes allow implementation of dynamic strategies for managing the virtual world.

The third advantage of P2P DVEs is the lower running costs for the provider. With a typical server-client architecture, most of the computational power is provided by the servers which are operated by the provider. On the other hand, with a P2P architecture peers carry the processing load and share the network traffic instead of a single data center [13][18][19].

The final advantage of P2P DVEs over centrally managed architectures is that the update dissemination with server-client takes at least two hops whereas it is possible to achieve single hop update dissemination with P2P architecture. P2P architectures allow peers to communicate with each other directly. With servers, update messages should first reach to the servers before being disseminated to other peers. [9]

2.1.2.2 CHALLENGES OF PEER-TO-PEER DVES

Besides providing scalability and reliability, P2P architectures have their drawbacks. Distributed state management is more complex than the central management on servers. When object state management is handled by peers, limited resources on a single peer may become a bottleneck on the overlay network. Consistency and security considerations also introduce new challenges. Validation between the states maintained by independent peers may be necessary. Maintaining the persistence of the environment requires specialized replication strategies [1]. Failure of a node should not make unavailable the object states managed by the failing node. P2P approaches that do not allow peers to establish direct connections, latency and network saturation may become a performance bottleneck. Gilmore [6] even argues that it may not be feasible at all to realize P2P DVEs.

Consistency is harder to achieve with P2P DVE than server-client DVEs. Since the game state is managed distributedly; concurrent and conflicting updates can be processed at independent nodes and may cause inconsistencies between states maintained at different nodes [2]. In case of conflicting states during peer interactions, some of the peers or dedicated servers can be assigned as arbiters [20]. For the inconsistencies that are not caused by interactions, the prevention methods mentioned in section 2.1 are also applicable for peer-to-peer DVEs.

Persistency defines where and how the primary copies (root objects) of objects are stored. Traditionally root objects are stored on servers of server-client architectures. Same approach is applicable for peer-to-peer architectures by using dedicated persistency servers. If scalability is not a concern, each peer can maintain a copy of the global state in a fully distributed manner. A scalable alternative is delegating the persistency related jobs to super peers by assigning a sub-region of the world to a super peer. Super peers approach introduces additional challenges when inter-region events happen. Overlay storage is another option for persistency. Persistency is provided by distributing ownership of the objects according to their IDs and making use of existing DHT technology and its replication strategies. Hybrid solutions that use super peers and overlay storage together as backups for each other are also available. Another strategy for distributing object ownerships to peers is relying on the distance between objects and the peers on the virtual world. [6]

Security is the strongest drawback of peer-to-peer architectures. Solving security issues without centralized control in a peer-to-peer environment is very hard [17]. Distributed maintenance of the world state makes it really hard to have full control over the virtual world [2]. Aside from security issues related to users of the DVEs, peer-to-peer systems are also vulnerable to external attacks like DDOS [21]. A hybrid solution is proposed for overcoming security issues of P2P structures [14].

2.1.2.3 VON AND FLoD

The known approach for efficient and scalable update dissemination on peer-to-peer DVEs is employing Voronoi diagrams for managing the overlay network. Hu's frequently referenced work, Voronoi Overlay Network (VON) [9], forms the basis for the later Voronoi-based work. VON utilizes direct connections between peers who are in each other's area of interest (AOI). The Connections are established according to Voronoi diagrams locally maintained by peers. VON represents the peers as sites of Voronoi diagrams. Each peer maintains a Voronoi diagram that contains itself and the other peers in its spatial proximity in the virtual world. Peers then classify their neighbor peers according to their positions on these Voronoi diagrams and establish direct connections according to the resulting classification. The AOI of a peer is defined as the circle around the peer with a radius that is dynamically adjusted according to the number of peers contained in it. As the number of peers in proximity increases, the range decreases to limit the number of connections.

All peers inside a peer's AOI are called its AOI neighbors. AOI neighbors whose Voronoi regions immediately surround a given peer's Voronoi region are called the enclosing neighbors of the peer. AOI neighbors with some of their enclosing neighbors lying outside the given peer's AOI are called the boundary neighbors of the peer. Peers can be both enclosing and boundary neighbors for a given peer at the same time. Note that the enclosing neighbors may lie outside the AOI. **Figure 3** illustrates boundary, enclosing, and AOI neighbors.

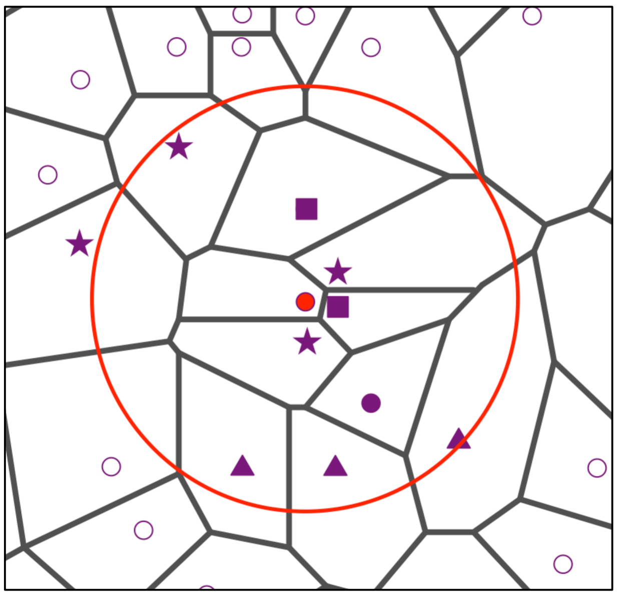

Figure 3 VON classification of peers on Voronoi diagram: For the red circle with the AOI of the large red ring, squares denote enclosing neighbors, triangles indicate boundary neighbors, stars denote neighbors that are both enclosing and boundary. Filled circles denote ordinary AOI neighbors, and empty circles denote non-AOI neighbors.

Peers maintain direct connections with their AOI neighbors and enclosing neighbors. When a peer moves in the virtual world, it notifies all connected peers about the change by sending updates. Since peers have direct connections to all other peers that are interested in the updates, updates are delivered with a single hop, whereas a client-server network can achieve two hops at best.

Boundary neighbors of a peer function as gateways to the outer world beyond the peer's AOI. When a peer changes its position, and notifies its boundary neighbors about its new position, the boundary neighbors check their enclosing neighbors to see whether they are now in the AOI of the moving peer. If a boundary neighbor discovers a new AOI neighbor for the moving peer, it sends a notification about the new neighbor and the moving peer establishes a new connection with its new AOI neighbor. The operation of VON is summarized in **Figure 4**.

```
peer n wants to join:
    n gets an id from the gateway
    n sends join request to random peer m
    n connects to the peers in the received neighbor peers list

peer n receives join request for peer m:
    if n's Voronoi region contains m then
        n sends list of m's neighbors to m
    else
        n forwards request to its neighbor that is closer to m
    end if

peer n moves:
    n updates its Voronoi diagram
    n sends update to all of its connected neighbor peers

peer n receives update from peer m:
    n updates its Voronoi diagram
    if n is boundary neighbor of m then
        for x in AOI neighbors of n do
            if x becomes AOI neighbor of m then
                n notifies m for x

peer n receives notification of its new AOI neighbor m:
    n connects to m
    n updates its Voronoi diagram

peer n updates its Voronoi diagram:
    for x in connected neighbors of n do
        if x is not AOI neighbor then
            n closes connection with x
```

Figure 4 Summary of VON's Operation

The edge-to-edge spatial continuity of the non-intersecting Voronoi regions is required for the utilization of Voronoi diagrams for overlay management. Therefore, VON does not take into account the visibility while determining the AOIs. VON peers maintain connections with their enclosing neighbors and AOI neighbors even when they are not visible. When enclosing neighbors are outside the AOI, the connections to them become redundant. Moreover, since the AOI itself is visibility unaware, some of the connections created by AOI neighbors may be redundant. An overlay that is visibility aware may overcome these drawbacks. **Figure 5** illustrates the redundant connections with invisible enclosing neighbors and invisible AOI neighbors.

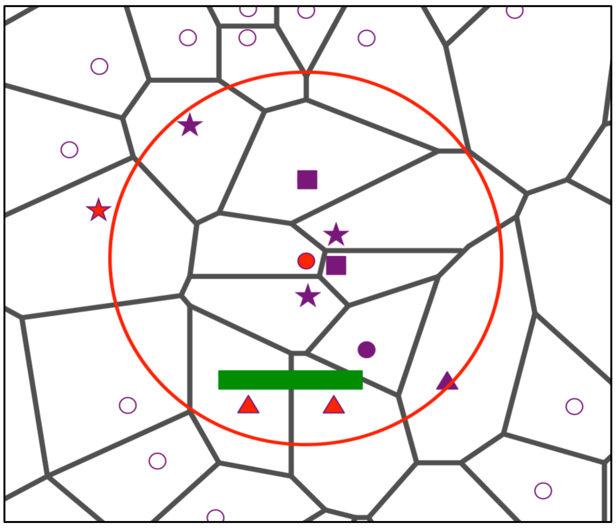

Figure 5 Redundant connections created by VON: When the green obstacle is present, peers marked with red triangles are invisible, but they have active connections since they are in the AOI. Peer marked with the red star is out of the AOI, but it has an active connection since it is an enclosing neighbor.

FLoD [8][22] is a VON based peer-to-peer 3D streaming framework. Peers on a DVE with enough density have intersecting AOIs. Exploiting this property, FLoD sends requests for content to other peers nearby before falling back on the server. The world is divided into fixed-size cells to index the content for efficient retrieval. Scene descriptions for the cells are computed offline and distributed to peers. When a peer moves, the set of missing scene parts is determined according to the inclusion tests performed on the cells against the AOI. Then, the moving peer queries its VON neighbors for the missing data. The data is requested from a randomly selected peer who responds positively to the query if there are any. In the absence of a positively responding peer, the data is requested from the server. The operation of FLoD is summarized in **Figure 6**.

```
peer n moves:
    for c in AOI cells of n do
        if c is not available then
            select a random connected peer m
            query m for c
            if response received from m then
                fetch c from m
            else fetch c from server

peer n receives query for c from m:
    if c is available then
        send response to m
```

Figure 6 Summary of FLoD's Operation

Since FLoD relies on VON and employs trivial circular AOI visibility, it lacks an effective source selection policy and accurate visibility determination. Missing data is requested only from the neighbor sites on the Voronoi diagram. It is quite likely that a peer out of the requester's AOI has the requested data in its cache, especially when the missing data is located near the boundary of the requester's AOI. Querying only the peers in the requester's AOI cannot benefit from intersecting AOIs adequately. Peers in the vicinity of 2 times the maximum radius of AOI share an intersection region with the requester, and these peers are ignored by the query policy of FLoD. Inadequate use of intersecting AOIs is depicted in **Figure 7**. Nonetheless, the principles used by FLoD are concrete and shared by Phaneros. Phaneros makes use of intersecting AOIs and two-phase requests in a more robust way by employing PVS.

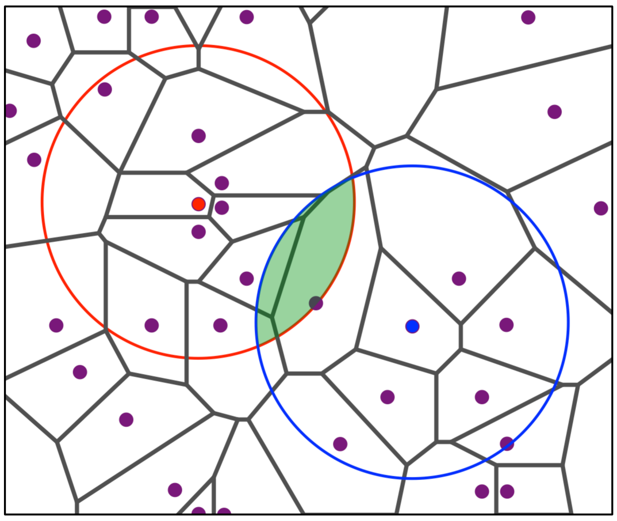

Figure 7 Intersecting AOI region that is not considered by FLoD: The green shaded region is the intersection of red and blue AOIs. When querying for content, although the blue peer is likely to have the content in the green region, the red peer does not query the blue peer because the blue peer stands out of the red AOI.

2.1.2.4 OTHER WORK

Genovali [23] proposes a hierarchical layout management scheme on top of Voronoi tessellation with the focus on load balance among peers according to their available bandwidths. Peers enlarge their Voronoi regions by absorbing other peers with low bandwidths, hiding them from the rest of the network and acting as proxies for them. Absorbed (hidden) peers communicate with other peers through the super peer that absorbed them. The hierarchical overlay provides load-balanced passive object management. However, the hierarchical multi-hop routing fails to provide efficient update dissemination when compared with the single hop alternative provided by VON. Also, the load on the super peers tends to be exponential because they have to cover the AOIs of the hidden peers they have absorbed.

Almashor et al. [16] follow a similar path by introducing a third dimension to Voronoi diagrams taking into account the available resources on peers for load balancing. The third non-spatial dimension represents the load of the peers. To form the layout network, peers stay connected to all their primary and secondary neighbors regardless of their positions in the world. According to the position on the z-axis of the Voronoi diagram, peers take the role of super peers. The proposed method successfully manages to shift the burden from servers to peers and load balance it according to available resources on peers. The autonomous load balancing scheme throws in servers when resources of peers are not sufficient. This way, the method remains effective when the flocking behavior of the users create hotspots. The method provides means of dissemination via the overlay network of peers, super peer, and servers. However, the method lacks a defined optimization for the communication between peers.

Ghaffari et al. [24] propose a greedy geometric routing method which maintains two Voronoi diagrams simultaneously, updating each in turn according to the positions of the peers in the virtual world while at the same using the other diagram for communication. Instead of handling joins/leave operations, the overlay network relies on updating itself continuously. The greedy routing achieves 13 hops for end-to-end on the average, but the provided optimizations lessen the number of hops to 3 hops at the expense of increased complexity caused by the potential time synchronization, the introduction of the extra Voronoi diagram and the optimizations. The proposed method form an overlay network which is capable of delivering messages according to positions and keeping itself operative on peer joins/leaves. However, the method is away from achieving single-hop update dissemination of VON.

Steed and Angus takes a different stance by utilizing visibility in their work. They use the Potentially Visible Sets (PVS) in their study to construct Frontier Sets [25]. PVS is a from-region visibility culling method where the world is divided into view cells, and the set of visible objects is computed for each view cell. The set of visible objects cover all objects that are visible from any point in the view cell. Therefore, the visibility defined by PVS is valid for a viewer as long as it is in the corresponding view cell [26]. Steed and Angus define a Frontier as a pair of view cells that are mutually invisible to each other and a Frontier Set as a pair of view cell sets such that no cell in one set is visible to any in the other and vice-versa. The method culls unnecessary communication between peers according to the Frontiers. No messages are exchanged between peers as long as they stay in Frontiers of each other. The defined invisibility culling requires recomputation of the Frontiers. When a peer leaves the Frontier where it resides in, the computation should be repeated for both the leaving peer and the other peer of the pair. The study relies on the rarity of the Frontier leaves to compensate for the complexity of the computations. However, with a higher number of simultaneous peers and a larger environment, computation complexity becomes polynomially more significant. The method is not suitable for massive worlds with a massive number of participants in the way the authors describe it, but the study reveals the applicability of PVS on the peer-to-peer DVEs. Phaneros employs PVS notion for deciding on the visibility between view cells instead of the invisibility between them.

Knutsson [27] presents a comprehensive study for peer-to-peer DVEs covering object management and update dissemination, known as SimMud, which relies on the distributed hash table (DHT) provided by Pastry [28] and the application layer multicasting provided by Scribe [29] on top of Pastry. SimMud statically divides the world into regions and assigns unique Pastry keys to the defined regions to make use of Pastry's key management for identifying responsible peers for regions. Peers in a region form a multicast group and update each other about their states in a single-writer multi-reader manner (publish/subscribe) using Scribe. Peers are allowed to communicate with peers in other regions if the application requires. SimMud allows direct UDP messages between peers when mutual interaction happens between pairs. It solves non-player objects management and assignment of arbiters for conflicts by handing the management of the regions to the peers who are the owners of the keys assigned to the regions on Pastry. The overall architecture shows similarity with VON and Phaneros, proposed in this paper. However, it lacks a defined protocol of direct messaging, a consistent scheme for region definition and defined usage of visibility even though it mentions some form of visibility while establishing multicast groups.

SimMud successfully demonstrates the convenience of application layer multicasting for peer-to-peer DVEs and becomes one of the founding studies for Phaneros with VON and Frontier Sets.

Han [30] proposes a multicast interest management method with dynamic partitioning. The study defines two levels of communication as low frequency and high frequency. Low-level communication is carried out by super-peers assigned to peer groups. They are responsible for aggregating updates from group members and casting aggregated updates with low frequency for the use of other peers that are relatively distant, in neighboring regions. Groups are formed according to AOIs and shared interests. A second multicast channel is used for high-frequency update messages for relatively near peers. Multicast addresses are assigned to statically defined regions. The method optimizes the number of update messages by making use of sub-regions, view direction and visibility range to limit the communication with neighboring regions. Usage of multicast makes the method unsuitable for internet applications.

In their comparative study, Rueda et al. propose COVER as their chosen method for providing awareness for P2P DVEs [31]. COVER defines two types of peers: covered and uncovered. Covered peers are those whose AOI is entirely covered by AOIs of its first level neighbors. Similar to VON, a covered peer cannot be approached by another without the notification of its neighbors. Communication with the covering peers is sufficient for providing continuous awareness. Super peers are hierarchically utilized to provide awareness to uncovered peers. Super peers are assigned to statically defined regions on a quad-tree, and they act as servers for the corresponding regions. Super peers also provide inter-region communication for peers whose AOI intersects multiple regions.

Teler [7] eliminates the need of prior download of the entire scene by introducing visibility based streaming for remote walkthrough applications. The proposed centralized approach makes use of the online visibility calculations taking place on the server side according to the periodically transmitted client viewing and movement parameters. The server selects and transmits only the parts of the scene that are visible to the clients. Compression of the transmitted data, a level of detail (LOD), and a request prediction scheme on the server are employed to improve the performance of the approach.

Chang [32] defines on-demand updates as the root of the latency problems caused by distributed object platforms like CORBA, DCOM and RMI. The study suggests object replication as the solution and redefines update dissemination problem as the multicast synchronization between object replicas. The update messages are delivered to object replicas by the multicast channels allocated to each object. However, the number of multicast channels are limited compared to the number of objects. Consequently, an intelligent object grouping scheme is necessary to make efficient large-scale distributed network applications. Chang proposes an adaptive and incremental object grouping method with multiple levels of consistency for update dissemination from a server to its clients and compares it the spatial object grouping.

Jia et al. [33] proposes a prioritization framework for 3D content streaming on P2P DVEs. The proposed method utilizes multi-layer AOI management that is based on spatial proximity. The study proposes a pull-push hybrid source discovery protocol. At the first stage, nodes push their data availability information to newly discovered neighbors. Then, the node in need, requests pieces from the nodes who has declared availability of the pieces. If a node cannot gather sufficient data at the push stage, it initiates pull procedure as the second stage. Piece queries are made only to the nodes that are in proximity to increase the chance of intersecting areas of interest. The regions being queried are prioritized according to their distances to the querying node. Querying starts with the closer regions and continues towards the outer regions. A scoring function is also suggested for prioritizing the source nodes. Source nodes are ordered according to their CPU utilization, network latency and bandwidth allocation. According to this ordering, primary and secondary sources are identified and queried.

Diaconu and Keller [34] proposes a distributed server system (Kiwano) which separates dynamic content (avatars and mutable objects) and static content from each other and manages the interactions between avatars by utilizing Voronoi diagrams.

Kiwano implements the interactions between avatars as the communication between super nodes. Each super node is responsible for a neighbourhood of nodes which can be dynamically adapted to the spatial distribution of the avatars. Avatars are assumed to be neighbors only when they are in a defined range of distance in terms of hops on the Voronoi diagram maintained continuously according to the positions of the avatars. The bounded number of nearest avatars are considered as area of interest. Therefore, interactions between avatars are allowed only when they are in each other neighbourhood. The size of the neighbourhood is bounded according to statistical data to achieve scalability.

Liu and Rungta [35] propose a caching approach for server-client virtual environments which reduces the bandwidth consumption by avoiding retransmission of the content. The proposed approach employs level-of-detail, delta transmission with multi-aura visibility. Objects are tested against co-centric axis-aligned bounding boxes with increasing sizes. Outer boxes represent the lower resolution visibility and inner boxes represent the higher resolution visibility.

Abdulazeez et al. [36] propose a hybrid architecture that aims scalability without compromising responsiveness. The proposed architecture is composed of multiple servers that manage the game state with super-peers and super-peer clones that manage sub-networks which are formed based on spatial proximity on the virtual world. The study demonstrates possible advantages of distributed architectures compared to traditional server-client architectures.

Hu and Chen [37] proposes spatial publish/subscribe messaging (VSO) on top of Voronoi-based overlay networks. VSO uses greedy spatial forwarding as its routing scheme. Peers are assigned as matchers for subscription areas whose centers lie inside their Voronoi regions. Matchers are responsible for greedy-forwarding publications to related subscribers. Voronoi regions of matchers are dynamically resized for load balancing. The study is interesting since it is built on top of VON, the reference study of Phaneros. A plausible question would be whether it is possible to implement Phaneros on top of VON and take advantage of both methods. Hu's work shows that even it is theoretically possible to do so, implementation of the scheme would be challenging. Phaneros relies on statically defined (or at least distributedly and deterministically identifiable) visibility regions as subscription channels. The dynamic nature of VSO makes is very hard to use it as the spatial publish/subscribe required by Phaneros.

2.2 PEER-TO-PEER NETWORKS

Peer-to-peer networks were made popular by file sharing applications in the late 90s [38]. The idea of a self-organizing, decentralized network without the need for servers, fascinated millions of users who want to share files freely without control. A P2P network is defined as a distributed overlay network of peers that are both consumers and suppliers of services and resource [39]. A **peer** is any networked entity that provides and/or consumes one or more services. Peers operate independently and asynchronously from other peers.

Peer-to-peer overlay networks may be structured or unstructured. Structured overlays strictly control the topology of the network for faster retrieval whereas unstructured overlays work on loose protocols and flooding. The efficiency of structured overlays comes at the expense of higher overhead and less flexibility. On the other hand, unstructured layouts may cause peer overloads that affect the performance of the whole network. [38]

Structured P2P overlays are structured on Distributed Hash Tables (DHT). DHTs deterministically distribute key-value pairs to peers according to their unique keys. A routing policy and lookup service are defined with join/leave procedures to provide required services. DHT-based systems provide $O(logN)$ query complexity in terms of network hops where N is the number of peers in the network. CAN, Tapestry, Chord [40], Pastry and Kademlia are examples of such systems. [38]

Like many other application domains, DVEs benefit from multicasting. Multicasting is the facility of delivering a message from a single sender to multiple recipients at once. This single producer multi-consumer messaging scheme is also called publish/subscribe messaging. When multicasting implemented at IP level, it is called IP multicasting. IP multicasting is performant and easy to use, however it is not suitable for the internet. Flooding nature and limited IP range make it inapplicable for internet applications. Application Layer Multicasting (ALM) is providing the same functionality as an application service. Instead of physical network nodes, ALM is implemented on application nodes (peers) and their connections. Since they use the connections between application nodes, multiple copies of messages are inevitably generated, and they perform worse than IP multicasting. However, ALMs are applicable for internet domain and with the utilization of multicast trees, they provide improved efficiency. [41]

Many examples of P2P overlays and ALMs are available [38][41]. Chord and Scribe are used by Phaneros as the underlying P2P overlay network and ALM scheme. Phaneros implements a spatial publish/subscribe messaging protocol on top of these two. It is not mandatory for Phaneros to use Chord and Scribe as long as the same spatial publish/subscribe messaging protocol can be implemented with the alternatives. The precondition for the Phaneros protocol is that the peers should be able to publish and subscribe to predefined static regions on the virtual world.

Following sections describes Chord and Scribe in detail provide insight into P2P overlays and ALMs.

2.2.1 CHORD

Chord is a straightforward and efficient distributed look-up protocol [40]. As an example of DHT systems, Chord's fundamental capability is fast key to node mapping for retrieving data stored as key-value pairs at nodes. Chord uses Chord IDs for uniquely identifying both its nodes and the keys of the look-up service. Chord IDs are generated by a consistent hash function, namely SHA-1.

Chord defines the function *successor(k)* as the node with the smallest ID that is larger than the key *k*. This function provides an ordering for the nodes. The nodes on the Chord network from a circle where each node is aware of its successor. Also, Chord assigns ownership of the keys to nodes using the *successor* function. The key-value pair *(k, v)* is kept on node *n* where *successor(k) = n*. Informally stating, a key is located on the node whose ID is the successor of the key. With this setting, a query for a key can simply be forwarded on the circular list of nodes until it reaches the successor node of the key. Basic Chord operation is illustrated in **Figure 8**.

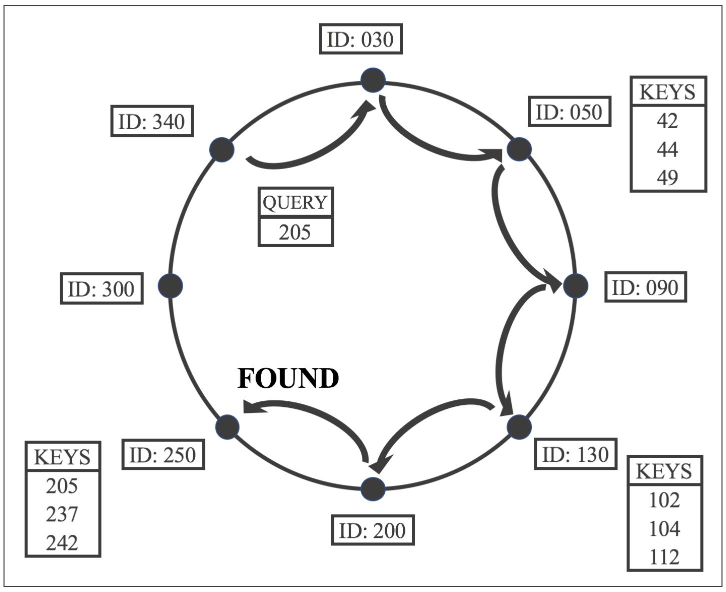

Figure 8 Basic operation of Chord: Each node is aware of only its successor. Node 340 queries for key 205. Each node forwards the query to its successor. The query makes *O(logN)* hops.

It is seen that the basic look-up guarantees locating the right node but it has *O(N)* complexity where *N* is the number of nodes on the network. Chord nodes maintain additional connections as shortcuts to lower this complexity. These connections are called fingers. On a network of $N = 2^m$ nodes, each node maintains *(m – 1)* fingers where the i^{th} finger is its $(1+2^i)^{th}$ successor. Instead of simply forwarding the query to its successor, a node chooses the finger which has the largest ID that is smaller than the queried key. In other words, nodes forward a query to the finger that is closest to the key on the circle. With this improved scheme, Chord lookups take *½O(logN)* time on average. Our tests show that on a network of 1.000 peers, a key is routed to its destination in less than five hops on average. Chord lookup with fingers is illustrated in **Figure 9**.

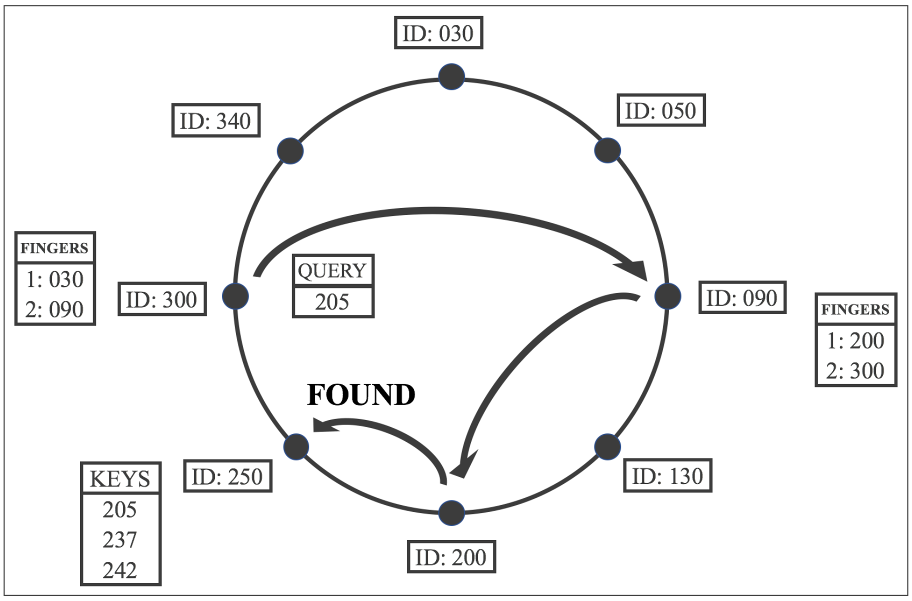

Figure 9 Operation of Chord with fingers: Nodes have finger tables. Node 300 queries for key 205. Instead of its successor, it forwards the query to its second finger, Node 090. Node 090 forwards the query to its first finger, Node 200. Node 200 forwards the query to its successor. Query reaches its destination in *O(logN)* hops.

In the case of node failure, Chord is capable of continuing its operation. Nodes that have the failing node among their fingers have other fingers or at least their successors as alternative paths for forwarding queries. The other reliability concern is not losing data when a node fails. This issue is overcome by replicating key-value pairs at successors of the owner node according to desired replication level. When a node fails, its successor becomes the owner of its keys by definition without loss of data.

Another key feature of Chord is that it can handle frequent leaves/joins efficiently. A join is realized by finding the successor of the joining node, connecting the node to the successor and notifying nodes whose fingers might need updating. Those nodes to be notified can be found by using the existing look-up service. The update procedure has $O(log^2N)$ complexity. Once a node is connected to the Chord circle, the keys of its predecessor are transferred to the new node if it is necessary. Leaves are implemented by using the connection with the predecessor of a node. The keys of the leaving node are transferred to its successor, and the join procedure is started for the predecessor of the leaving node.

2.2.2 SCRIBE

Scribe is an application layer event notification infrastructure which provides topic-based publish/subscribe multicasting. Scribe nodes create topics and publish messages on topics at runtime in a fully distributed manner. Multicasting is performed efficiently since Scribe uses multicast trees that are maintained on the fly. Subscribe messages are forwarded on Chord's internal routing, leaving marks on the nodes they pass through to form the reverse multicast trees. Since the topics are keys of the underlying DHT provided by Chord, reliability provided by Chord is inherently available to Scribe [29]. The original implementation of Scribe uses Pastry as the underlying overlay network.

Scribe is implemented as four main functions: subscribe, unsubscribe, publish and create. These functions are implemented in a fully distributed manner by utilizing the functions provided by the underlying DHT.

Each Scribe topic is a key on the DHT. The owner node of the key is responsible for maintaining its value and acting as the root node of the multicast tree corresponding to the topic. Since we use Chord as the underlying overlay, topics are owned by their successors. In other words, the node with the smallest id that is larger than the topics key is the owner.

When a node sends a subscribe request on a topic, the request is delivered to the owner node. Each node on the way to the owner keeps the previous node as one of its leaves on the multicast tree. This way each subscription request leaves a path marked behind for efficient delivery. Create function can be realized trivially when a subscribe request is reached to the owner for the first time. The operation of Scribe publish operation and formation of the multicast tree is illustrated in **Figure 10**.

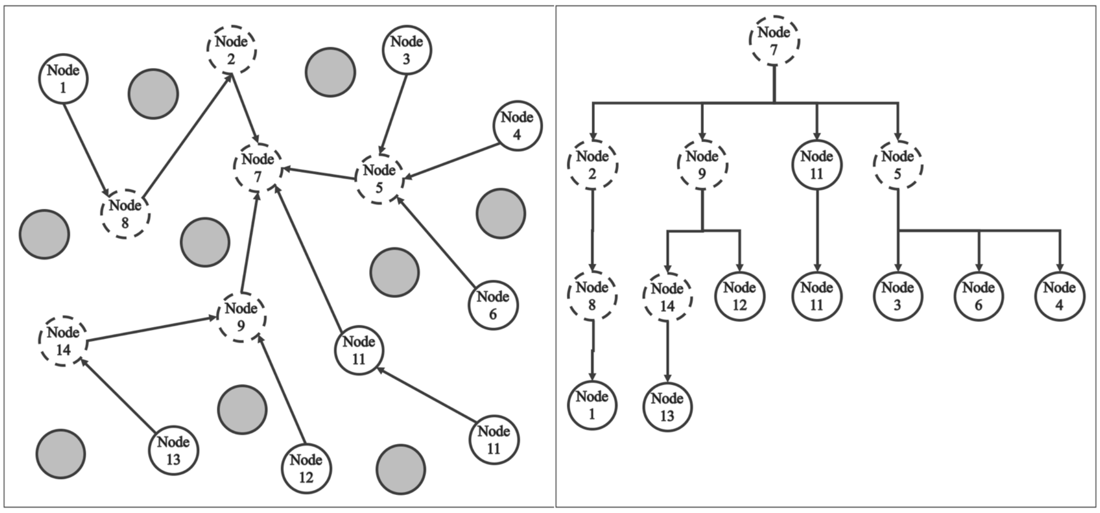

Figure 10 Scribe Subscription Operation: Node 7 is the owner node of the topic. Subscriptions and the paths followed by them are shown on the left. Solid nodes are subscribers. Dashed nodes are not subscribers, but they are in the paths of subscriptions. The resulting multicast tree is shown on the right.

The multicast trees are used as the second step of a publish operation. The first step is setting the value of a key on the DHT. A publisher sends a publish request on a topic as a value change for the corresponding key. The request is delivered the owner node of the topic by the overlay. Upon receiving the publish request the owner, who is also the root node of the multicast tree, casts the publish message to its leaf nodes. Each node on the tree re-casts the publish message to its leaves until all subscribers receive the message. The publish operation is illustrated in **Figure 11**.

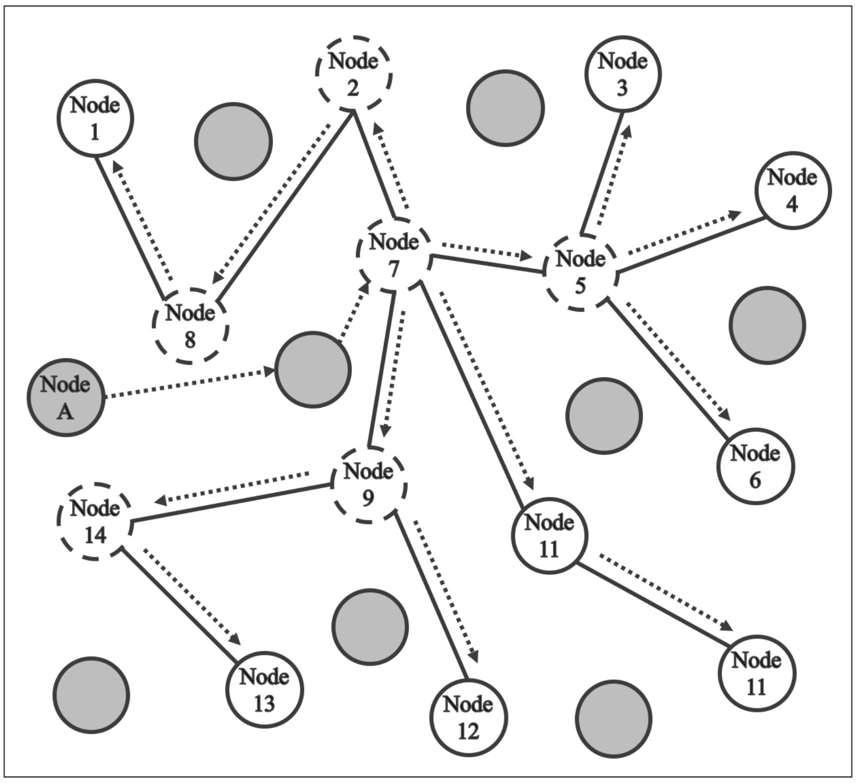

Figure 11 Scribe Publish Propagation: Node A publishes on the topic owned by node 7. The publish message reaches node 7 and is propagated through the multicast tree. The multicast tree rooted at node 7 is shown with solid lines. The propagation is shown with dotted arrows.

2.3 VISIBILITY CULLING

Visibility culling is a fundamental optimization in the field of computer graphics. From the beginning of the field, visibility culling has been used for filtering out invisible geometry before entering the rendering pipeline [26]. Visibility is a fundamental aspect of human perception; therefore, it is also employed by AOI management methods frequently. Some studies including this one, make use of visibility for filtering out unnecessary network traffic in virtual environments field.

Arguably the most frequently used visibility culling method is limiting the range of visibility. A viewer is assumed to be able to comprehend only a limited spatial proximity around it. Depending on the application requirements and available resources, a visibility range is dynamically or statically defined. Subjects farther than the range are filtered out.

Two other basic forms of visibility are view frustum culling and back face culling. View frustum culling filters out the objects that are out of the view frustum. The view frustum is the portion of the space that is visible at a moment according to the viewer's position and the looking direction. Back face culling filters out the face that are facing away from the viewer. When there is no transparency, rendering of back faces can be omitted. Besides these two, occlusion culling is available as the more advanced visibility culling method. Objects in a 3D environment occlude each other. If an object is occluded by other objects that are closer to the viewer, it can be omitted from rendering. Since the back face and view frustum culling methods are straightforwardly implemented, occlusion culling is subject to visibility studies. View-frustum culling, back-face culling, and occlusion culling methods are illustrated in **Figure 12**.

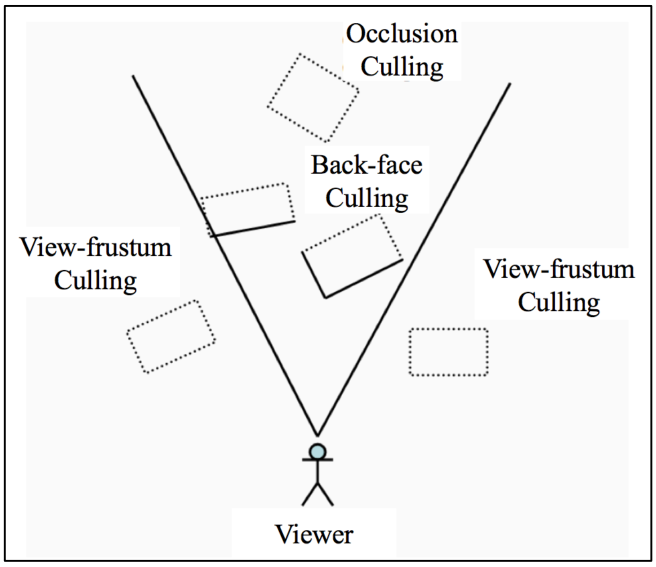

Figure 12 Visibility culling methods: Dashed lines are filtered out by the visibility culling methods denoted next to them.

Visibility culling methods are classified as online/offline and point-based/from-region. These two classifications overlap. Point-based methods are used as online culling methods, and from-region methods are used as offline culling methods. Point-based methods compute visibility for a single point of view. For each position of the viewer, visibility is re-computed. Re-computation may be avoided by iterative updates [42].

From-region visibility culling methods partition the world into regions that are called view cells. Then at the preprocessing stage, for each view cell, a potentially visible set of objects or primitives is computed. A potentially visible set conservatively contains, at least, all objects that are visible to a viewer from any point in the associated view cell. Rendering only the objects in the potentially visible set is sufficient for generating an accurate view of the world for the viewer in the expense of rendering some more than the exact set of visible objects [26]. Moreover, it is possible to compute conservative from-region visibility from a view cell by using point-based visibility. Wonka [43] shows that shrinking occluders by ε provides visibility that is valid as long as the viewer stays within the ε neighborhood of the point from where the visibility is computed. Using this property, a finite number of samples with ε distance between the samples is sufficient for computing conservative visibility. The accuracy of the method changes with the value of ε.

Phaneros is tested in an urban-like environment with a significant ratio of static occluders that are placed on the ground. Such environments are called 2.5D environments. The geometry of 2.5D environments allows efficient online computation of point-based visibility. Downs et al. propose an occlusion horizon method for efficient computation of visibility in such environments. The proposed method represents the scene as a set of convex vertical prisms. The line strip passing through the tops of the occluders is called the occlusion horizon. By using the binary tree representation of the horizon, efficient visibility computation can be performed online. Phaneros test simulator uses from-region visibility data produced by an implementation of the mentioned occlusion horizon method. The implementation is simplified by restricting viewer movement to be discrete. This way, the possible positions in a view cell is limited and the from-region visibility is computed with a brute-force implementation instead of using the occluder shrinking.

CHAPTER 3

THE PROPOSED FRAMEWORK: PHANEROS

In this study, we present a fully distributed peer-to-peer framework for DVEs. Phaneros is a fully distributed peer-to-peer DVE framework which provides two essential functions for DVEs: update dissemination and 3D streaming **Figure 13**. In a DVE actions of the participants and the changes in the world are transmitted as update messages. The process of delivering the updates to relevant peers is called update dissemination. Delivering 3D content to participants for rendering is called 3D streaming. By being peer-to-peer, Phaneros is able to distribute the computational load and bandwidth demand to multiple nodes. Even when it is distributed, the load is too much for peers to handle without further optimization.

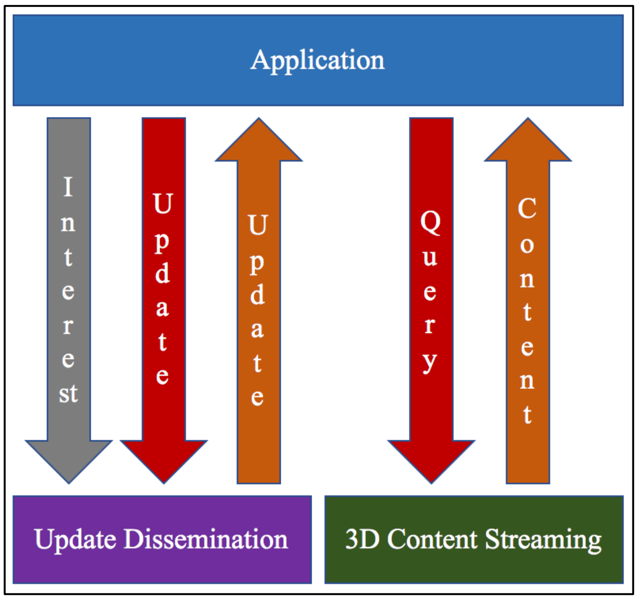

Figure 13 Phaneros Functional Overview

Update dissemination on P2P networks is optimized by filtering out unnecessary communication among peers. The challenge of designing a performant update dissemination scheme is keeping the overlay network operational while improving its performance. The connectivity cannot be ceased for the sake of performance improvements. The eliminated messages should not be part of the maintenance of the overlay network.

On a P2P environment, 3D streaming is optimized by selecting right sources for the content. A peer is a suitable source when it has the required content and available resources to deliver it. A successful DVE framework should be capable of choosing right peers as the sources for delivery. Phaneros is built on top of application layer multicasting (ALM) and from-region visibility.

From-region visibility allows accurate and efficient AOI management with the help of ALM. Visibility-based update dissemination filters out unnecessary traffic efficiently and visibility-based 3D streaming is capable of choosing right peers as sources.

The message types used Phaneros can be classified into three groups. The first group of contains the high-frequency update messages and the costly content messages. The required performance for the first group is hard to supply with multi-hop P2P communication. Because of this limitation, update messages and content messages are transmitted through the direct connections between peers. The second group of messages contains messages that are used for maintaining the mentioned direct connections. The direct connections are managed in the scope of AOI management. These AOI management messages are transmitted with significantly lower frequency, and they have significantly smaller sizes compared to messages in the first group. The small low-frequency interest management messages are delivered by the visibility aware application layer multicasting, which runs on top of a common peer-to-peer overlay network. The third group of messages is used for maintaining the overlay network. In Phaneros' case, these messages are the internal Chord messages. Phaneros uses existing technology for overlay network management and ALM. Scribe [29] is the employed application layer multicast scheme working on top of Chord [40], which is the underlying peer-to-peer overlay network. **Figure 14** illustrates the overview of Phaneros communication architecture.

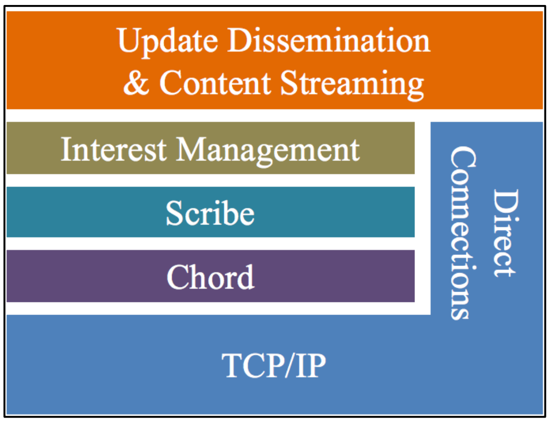

Figure 14 Phaneros Communication Architecture: Update dissemination and content streaming use direct connections, which are established according to interest management working on top of application-layer multicasting.

3.1 AOI MANAGEMENT AND UPDATE DISSEMINATION

In a DVE, the AOI of a peer can be defined as the portion of the world it sees and the other peers standing in the same visible portion. Interest management of Phaneros relies on the concept of Potentially Visible Sets (PVS). In its original definition, PVS is defined as the set containing all visible primitives from the point of view. When used for from-region visibility, PVS is defined as the set of all primitives visible from any point in a given region. Phaneros uses an extended definition of PVS. If we extend PVS definition to include the information of visible view cells. If a primitive in a view-cell is visible, the containing view-cell is also considered visible. With the extended PVS, the AOI of a peer is defined as its PVS. The PVS is the set of view-cells which are visible from the view-cell where the peer stands. It is trivial to compute extended PVS by adding a list of view-cells alongside the list of visible primitives. A sample AOI defined in terms of view-cells is illustrated in **Figure 15**.

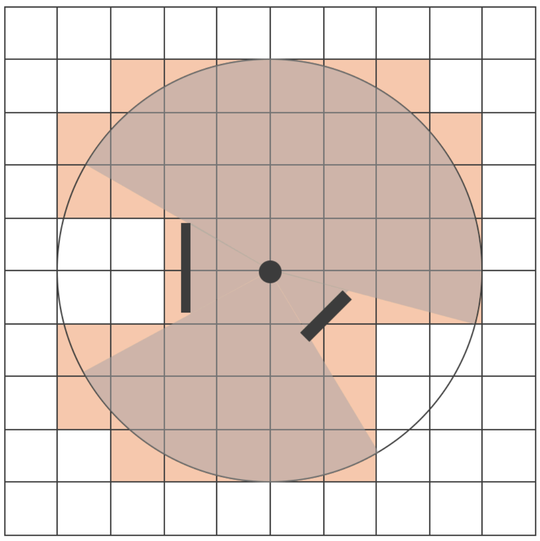

Figure 15 AOI and Corresponding View-Cells: For a viewer standing in the center, two occluders and the visibility range defines the visible area shaded with gray inside the large circle. View-cells corresponding to the actual visible area is highlighted with orange. The inclusion of partially covered view-cells depends on the application requirements. The illustration shows a conservative approach where partially covered view-cells are considered visible.

For interest management, Phaneros uses a straightforward publish/scribe messaging scheme. For each view-cell, a topic is made available for subscription. By definition, all the subscribers of a view-cell topic receive messages published in that view-cell's topic. Peers subscribe to topics of view-cells that are in their AOI. In other words, peers subscribe to messages published in their AOI.

Phaneros uses two message types for establishing its interest management function: cell-enter notification and cell-exit notification. When a peer leaves a view cell and enters another, it publishes a cell-exit notification to the cell it leaves and a cell-enter notification to the cell it enters. When a peer receives a cell-enter notification, it establishes a connection with the peer sending the notification. When a peer receives a cell-exit notification, it closes the connection with the peer leaving the cell.

Therefore, peers stay directly connected only with the peers in their AOIs. Notification scheme used by Phaneros is explained on **Figure 16**.

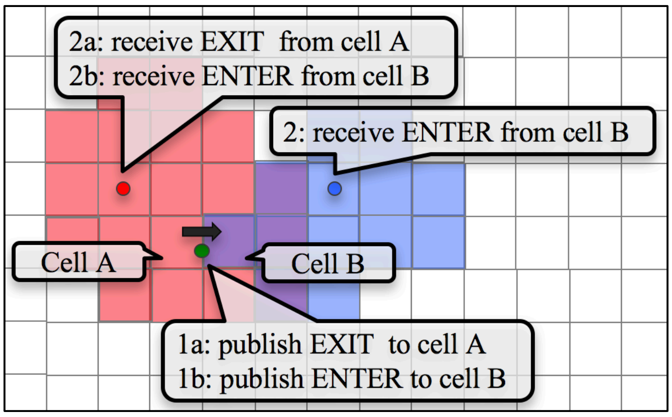
2a: receive EXIT from cell A
2b: receive ENTER from cell B
2: receive ENTER from cell B
Cell A
Cell B
1a: publish EXIT to cell A
1b: publish ENTER to cell B

Figure 16 Phaneros Notification Scheme: Red shaded cells are the PVS of the red peer. Blue shaded cells are the PVS of the blue peer. The intersection of the PVS is the purple region. Green peer moves from cell A to cell B. Red peer is subscribed to cell A and cell B. Blue cell is subscribed to cell B. Green peer publishes cell-exit to cell A, and cell-enter to cell B. Red peer receives cell-exit notification through cell A, and cell-enter notification through cell B. Blue peer receives cell-enter through cell B.

When a peer moves, it sends update messages to all its active direct connections. All peers that have the moving peer in their AOIs receive the update messages. The direct connections are used bidirectionally. Both ends of the connection send update messages. With the described scheme, Phaneros update dissemination is capable of delivering update messages to the interested peers with a single hop. Compared to two hops provided by the server-client approach, single hop is the only possible improvement. Update dissemination is illustrated in **Figure 17**.

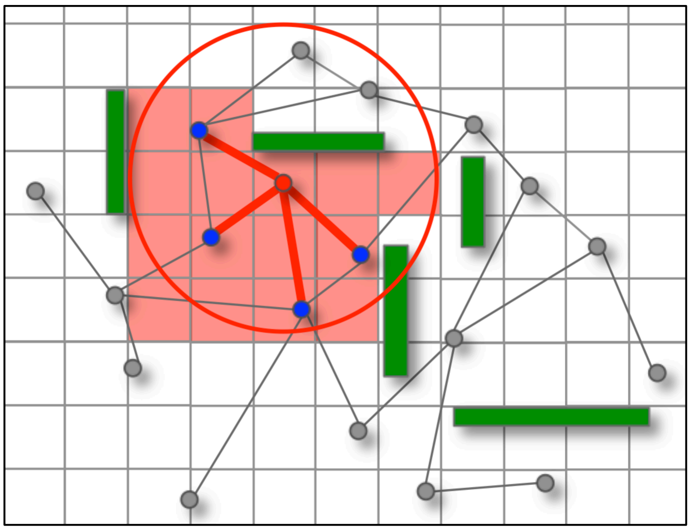

Figure 17 Phaneros Update Dissemination: Red shaded cells are the PVS of the red peer. Blue peers are visible to the red peer since they reside in the PVS. The red ring indicates the range of visibility. Green obstacles limit visibility and define the layout topology. The direct connections between the red peer and the visible peers are shown as red lines. In this example, the red peer has connections with four peers in its PVS although there are two more in the visibility range.

3.2 3D CONTENT STREAMING

Phaneros implements 3D content streaming using the same messaging scheme used for update dissemination and makes use of an important attribute of visibility. Visibility relation is symmetric in the sense we are using it. If we denote the set of view cells in the PVS of view cell *x* as PVS(*x*) then:

$x \in \mathrm{PVS}(y) \Leftrightarrow y \in \mathrm{PVS}(x)$, and

$\mathrm{PVS}(x) = \{y \mid x \in \mathrm{PVS}(y)\}$

Based on this definition, a policy for identifying sources to fetch content from can be constructed. The peers that are likely to have some content of view cell A in their caches are those peers residing in one of the view cells in PVS(A).

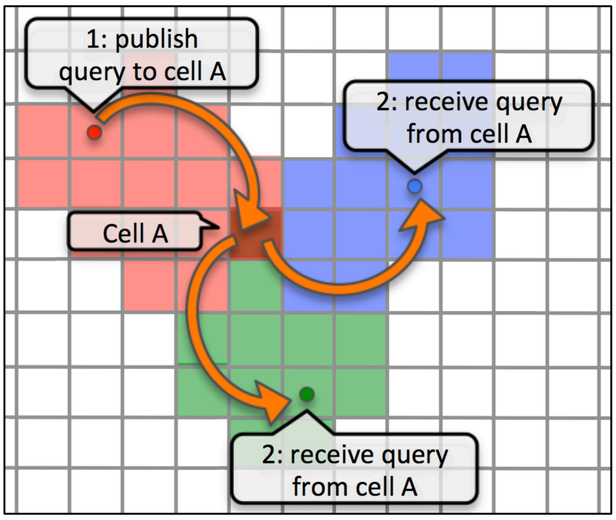
1: publish query to cell A
2: receive query from cell A
Cell A
2: receive query from cell A

Figure 18 Query Steps of Phaneros 3D Streaming: Red, Blue and Green peers have PVS of the matching colors. Cell A is the intersection of three PVS. Red peer needs the content in cell A and publishes a query to the topic of cell A. Blue peer and Green peer receives the query because they are subscribed to cell A's topic.

Phaneros 3D streaming works in two phases by first querying for the content then requesting the content from the fastest responder of the query. This approach chooses the peer with the lowest delay and the lowest load implicitly. The querying is implemented as a published query message. When a peer needs to query for content in view cell A, it publishes the query to the topics of the cells in PVS(A). Receiving subscribers respond through direct connections if they have the queried content in their caches. The querying peer selects the sender of the first received response as the source and requests the actual data by a request message that it sends through the direct connection with the responder. When no response is received for a query or a peer fails to deliver the content within a predefined period, the content is requested from the server. Since the publisher and the receivers already have connections no new connections are established for transferring the content package. Query steps are shown in **Figure 18**, and fetching steps are shown in **Figure 19**.

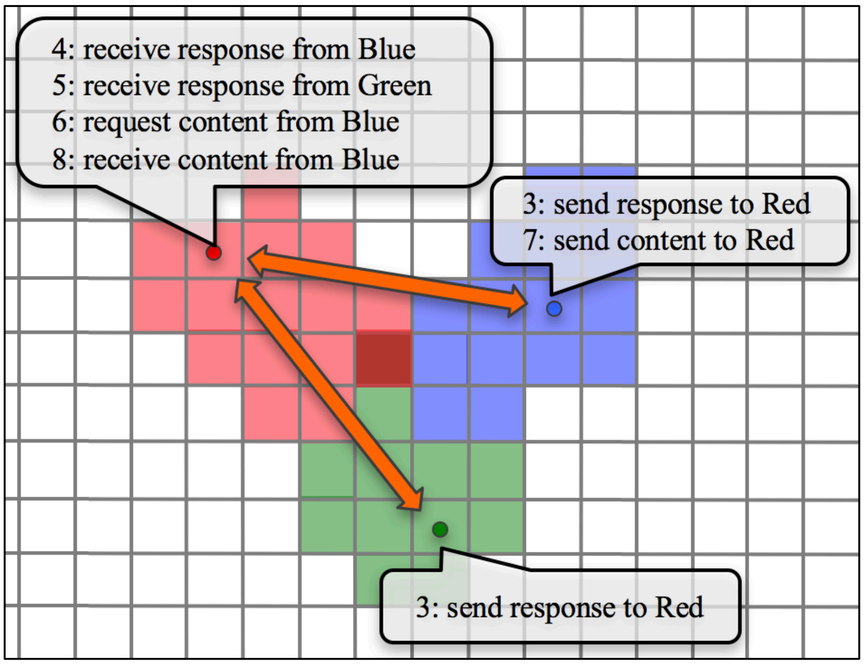
4: receive response from Blue
5: receive response from Green
6: request content from Blue
8: receive content from Blue
3: send response to Red
7: send content to Red
3: send response to Red

Figure 19 Fetching Steps of Phaneros 3D Streaming: Blue and Green peers send responses to Red peer via direct connections. Red peer first receives the response from Blue then the response from Green. Following the defined policy, Red peer requests the content from Blue peer and receive the content back via the direct connection.

In summary, Phaneros peers maintain subscriptions to the cells that are in their visibility based AOIs. When a cell transition occurs, the transiting peer publishes notifications to the cell it leaves and to the cell it enters. Update messages are disseminated through the direct connections that are maintained according to these notifications. When a peer does not have the 3D content of a cell in its AOI, it publishes a query for the content to the cells that are in the PVS of that cell. By the symmetry of visibility, the peers that receive the query are the peers that reside in the cells which the required content is visible from. Querying peer fetches the content from the fastest responding peer to select the most suitable peer as the source. The operation of a Phaneros peer is summarized in **Figure 20**.

```
peer p in cell c moves:
    p sends update to connected peers
    if p leaves its current cell c and enters d then
        p publishes cell-exit on topic of c
        p publishes cell-enter on topic of d
    p unsubscribes from PVS(c)
    p subscribes to PVS(d)
    p publishes queries for content of cells in PVS(d)

peer p receives query from q for c:
    if c is available in cache then
        p send response to q via direct connection

peer p receives response from q for its query for c:
    if q is the first responder then
        p fetches c from q via direct connection

peer p receives update from q:
    p updates its view

peer p receives cell-enter from q:
    p establishes connection with q
    p starts sending updates to q

peer p receives cell-exit from q:
    p disconnects from q
    p stops sending updates to q
```

Figure 20 Summary of Phaneros' Operation

3.3 RELIABILITY AND SECURITY

Phaneros relies on the reliability capabilities provided by Chord and Scribe. Chord internally replicates key-value pairs on successive nodes and maintains alternative successor connections as backup. In case of failure, the nature of the look-up process allows seamless recovery. Remember that Chord is a circular linked list of nodes that are ordered according to their associated Chord IDs. Each node is aware of its successor node and forwards look-ups when the look-up key is larger than its own identifier. Forwarding continues until the look-up reaches a node whose identifier is larger than the look-up's key. When a node fails, keys kept by the failing node are transferred to the next node on the circle. Since a node cannot forward to its failed successor, connections to the following successors are maintained as alternative forwarding paths. The number of backup connections are application specific. The mechanism explained here is responsible for keeping Chord network connected and operational but does not recover failed fingers. Failed fingers are eventually replaced by Chord's periodical update checks and updates fingers. [40]

Scribe provides another layer of reliability. Non-leaf nodes on the multicast tree send periodical heartbeat messages to their child nodes. When a child node does not receive heartbeat messages for a while, it suspects that its parent node has failed and send a subscription message to initiate repairing of the multicast tree. Scribe can also tolerate failing root nodes of the multicast trees. Access control lists kept by root nodes are replicated on its successors as Chord does with key-value pairs. When a root node fails, its immediate children detect the failure and initiate repairing by re-subscribing as explained above. [29]

The reliability provided by Chord and Scribe provides fast recovery for node failures but it does not compensate for the lost Phaneros messages that are published during recovery. Lost cell enter/exit messages do not cause failure for the whole system but, affects the consistency of the virtual world. Avatars may be invisible or suddenly appearing and disappearing with such messages loss. Fortunately, topics are distributedly managed. It is trivial to create duplicate topics and have multiple replica topics managed by dispersed peers. Implementation of such replication is trivial but the reliability is provided at the expense of extra bandwidth.

Security is a challenge for Phaneros application as it is for any other distributed system. Security is not in the scope of Phaneros. Phaneros aims at providing an efficient and scalable communication framework. Existing methods of security are available for the use of applications. Complex prevention methods of event ordering and state exposing might be employed as well as simpler reactive methods that use authoritative nodes for detecting and rolling back inconsistent updates.

3.4 AUXILIARY OPTIMIZATIONS

Phaneros is designed to be unobstructive by keeping it minimal. This study presents the capabilities of Phaneros in the purest sense in an abstracted way. Auxiliary optimizations of prefetching, level of detail, caching and dynamic AOI range management are kept outside of the scope and the implementation. For a fair comparison, VON/FLoD was also implemented without any auxiliary optimization. The test results available in this study reflects the power of the visibility-based area of interest management, spatial publish/subscribe messaging, single hop updates and 3D content streaming provided by Phaneros without any other application specific auxiliary optimizations. The mentioned optimizations are generally trivial to design as long as the underlying framework do not put barriers in front of them. In this sense Phaneros is more than ready for optimizations. Cell-based representation of the world and the AOI management schemes provides a strong foundation for further optimizations.

Level of detail methods require to know the location of the content relative to the viewer for prioritizing the pieces of the content and deciding the necessary level of detail. When there is from-region visibility information available, it is straightforward to incorporate a level-of-detail scheme because the peers are aware of the distance of the 3D content pieces before retrieving them. It is obvious that the order of retrieval can also be determined by using the location of the content. If necessary, a hierarchical LOD scheme can be constructed by merging view-cells into larger ones.

Caching strategies can easily be implemented on top of 3D content that is already packed in chunks with spatial metadata. Cache strategies are application specific. If user movement trends can be tracked, a proper caching strategy can be quite effective when paired with a successful prefetching scheme. **Prefetching** is about deciding which parts of the content will soon be needed and caching is about which parts of content should not be discarded because of the same concern.

Dynamic range of interest is a method of dynamic load management that is utilized by almost all studies that aim providing an infrastructure for DVEs. Like those mentioned above, Dynamic range of interest can also be applied to Phaneros applications easily. A dynamic distance filter which is adjusted according to the density of the environment can be implemented to cut down excess connections, update messages and content retrieval. The filter should check when cell entrance messages and update messages are received and when a cell transition occurs, just before the missing content is queried.

CHAPTER 4

IMPLEMENTATION AND TEST CASES

After experimenting with publicly available simulators, the need for a customizable, simple and efficient network simulator was arisen. The magnitude of tens or a couple of hundreds were not sufficient for comparing the performance of Phaneros. Test scenarios required 1000 peers to be simultaneously simulated. Chord and Scribe were needed to be implemented on the simulator. Existing simulators were not flexible enough for rapid implementation of these protocols. Either the implementations were too complex to be reliably modified, or the constraints could not be incorporated into the simulation. Besides these motivations, Phaneros was also planned to be open source and easily accessible. As a result of this evaluation, ActionSim was designed and implemented.

On top ActionSim, Phaneros and VON/FLoD are implemented. Phaneros implementation is built on top of Scribe implementation that is implemented on top of Chord implementation. VON and FLoD are directly implemented on ActionSim. Following sections explain how these implementations are made in the order of ActionSim, Chord, Scribe, Phaneros, VON and FLoD. The chapter continues with the design details of the virtual environment. The last section explains the test cases designed for the performance evaluation and comparison.

4.1 ACTIONSIM

ActionSim is a step-based continuous distributed application simulator. The fundamental entity of ActionSim is the simulation node. A simulation node can connect/disconnect with others nodes, do actions, send messages and receive messages. Computational budget, bandwidth allocation, and connection restrictions can be defined on the simulation nodes. Each simulation node has an inbox queue, an outbox queue, and an actions queue. In each cycle of the simulation, first all nodes process the messages in their inboxes, then they process the actions in their queues, and finally they deliver the messages in their outboxes to the corresponding recipients. At the completion of a simulation cycle, remaining computational budget and unused bandwidth is accumulated for the next cycle. An ActionSim message has three fields. These are the source node, the destination node, and the payload. The payload is the data to be carried, and it is set by the application node.

```
simulation.step:
        for n in nodes do
                n.process_inbox
        for n in nodes do
                n.process_actions
        for n in nodes do
                n.process_outbox

node.process_inbox:
        for m in inbox do
                application.onMessage(m)

node.process_actions:
        budget = budget + budget per cycle
        for a in actions do
                if budget > a.cost then
                        application.onAction(a)
                        budget = budget – a.cost
                else break

node.process_outbox:
        bandwidth = bandwidth + bandwidth per cycle
        for m in outbox do
                if bandwidth > m.cost then
                        bandwidth = bandwidth – m.cost
                        outbox.remove(m)
                        m.destination.inbox.add(m)
                else break
```

Figure 21 Flow of ActionSim

Application nodes implement application specific features by wrapping simulation node instances. Simulation node provides a simple interface for accessing the functionality defined in the previous paragraph. If registered for notification, an application instance is notified by the underlying simulation node about completed actions, received messages, established connections and dropped connections.

Application logic is realized in the callbacks of these notifications. When an application node needs to perform an action that is defined by the application domain, it registers the action to the simulation node and waits for its completion notification. Application instances are responsible for providing computational costs of actions and bandwidth consumption of application specific messages. An additional feature provided by ActionSim is the timer that works in the timeline of the simulation. With the help of the timers, an application node may register for a callback that will be called later in the simulation according to the simulation time. **Figure 21** presents an overview of the ActionSim.

Besides its simulation capabilities, ActionSim also provides a map generator module, two visibility calculation implementations, a rendering module, a logger, a statistics collection module. Rendering module provides top-down 2D rendering of the test environment. Peers, obstacles and view-cells are represented on the rendering. One of the peers is rendered in detail. **Figure 22** describes the rendering provided by ActionSim renderer.

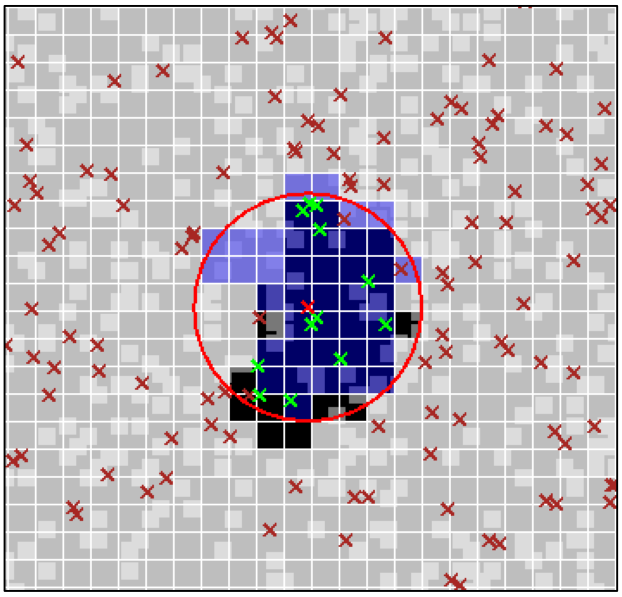

Figure 22 Rendering Sample: Blue shaded cells are the AOI of the peer in the center. The peer is connected to peers represented as green crosses. The red circle shows the visibility range. The cells without blue shade and the peers that are represented as red crosses are culled out by visibility. Dark shade represents the content available in the peer's cache.

Maps used by phaneros are grayscale PNG files. Value of a pixel represents the height at the corresponding coordinates. Map generation is done randomly according to the chosen occluder density. To be able to simulate with thousands of peers, actual map content is never replicated. Only a single instance of each map tile is created during the simulation and used by all peers.

Two visibility computation methods were implemented. First one is the trivial range based visibility. The other one an occlusion horizon method implementation. World metadata, map tiles and visibility information are packed as a single configuration package which represents a virtual world.

Leveled logging and sample-based statistics collection modules are also implemented in ActionSim. Logging facility allows tracing of the simulated method's internals. Statistics module provides a simple API for sampling at desired points of the simulation. Sample count, sum of all values, minimum value, maximum value and average value metrics are available for all monitored parameters.

ActionSim is suitable for developing and testing application level protocols like Phaneros, VON, Scribe and Chord. It is designed to be lightweight and customizable. The communication is abstracted as hops between nodes. With its current state, ActionSim provides bandwidth and CPU time constraints. Instead of simulating constraints, ActionSim focuses on simulating the effects of the constraints. If desired it is possible to incorporate additional constraints at the same abstraction level. For developing low level communication protocols with focus on simulated constraints, simulators like OMNET and NS-2 are suitable.

ActionSim do not provide ready-to-use latency and network failure simulation at its current state. An improvement on ActionSim would be implementing these two in an abstracted way to be available parametrically. Another point of improvement is the support for more complex scenarios. Phaneros tests were performed with random walking and hot spots and these two are sufficient for demonstrating capabilities of Phaneros. For developing and testing DVE applications and frameworks with interest on topics other than efficiency of layout, support for more complex scenarios will be necessary. Peer joins, peer leaves, repeating actions, goal based avatar movement and existence of competition are valuable features to be implemented.

4.2 CHORD

Chord nodes are implemented as wrappers around ActionSim nodes. Each Chord node is assigned a unique Chord ID. With its own ID, a Chord node keeps IDs of its successor, predecessor, and fingers. Chord IDs are implemented as the first four bytes of the SHA-1 keys created from the name of the nodes. Four bytes allows 2^{32} distinct values which are sufficient for the tests.

Besides the IDs, a Chord node internally maintains a list of queries that were forwarded to next node. An entry of the list contains the ID that is queried and the ID of the node that made the query. When a Successor Query message is received, the receiving node uses this list to identify the source of the query and forwards the response to it. Our implementation makes an optimization when there are multiple nodes waiting for the same response. The first response is forwarded to all nodes that made the query without waiting for distinct responses for each of them.

Chord protocol is implemented with five internal messages for maintaining the overlay and one message for carrying the payload over the overlay. The internal messages are Predecessor Notification, Predecessor Query, Predecessor Response, Successor Query and Successor Response. All internal messages have a common field that carries the Chord ID of the previous node. In fact, instead of a single previous node, the internal messages carry a list of all previous nodes with themselves for statistical data collection. The history is not used in protocol implementation except the last entry. Consequently, the size of the list is ignored in cost calculations. Besides the common field, Predecessor Response carries the Chord ID of the predecessor, and Successor Response carries the Chord ID whose successor is being queried and the Chord ID that is the queried successor. The carrier message is called Chord Message for easy identification by the external users.

```
node n receives Successor Query m:
    if m.key is in (n.id, n.successor) then
        n sends Successor Response to m.sender where
            response.key = m.key
            response.successor = n.id
    else n forwards m to closest preceeding finger of m.id
        n adds m.sender to waiting list

node n receives Successor Response m:
    if m.key = n.id then
        n sets n.successor = m.successor
    for x in waiting list do
        if x.key = m.key then
            n forwards m to x.sender
    for f in n.fingers do
        if f.id = m.key then
            n replaces f with m.successor

node n receives Predecessor Query m:
    n sends Predecessor Response to m.sender where
        response.predessor = n.id

node n receives Predecessor Response m:
    if m.predecessor is in (n.id, n.successor) then
        n sets n.successor = m.predecessor

node n receives Predecessor Notification m:
    if m.sender is in (n.predecessor, n.id) then
        n sets n.predecessor = m.sender

node n receives Chord Message m:
    if m.destination = n.id then
        n notifies application for m
    else if m.destination = n.successor then
        n forwards m to n.successor
    else n forwards m to closest preceeding finger of m.id
```

Figure 23 Summary of Chord Implementation

Two periodic maintenance function of Chord are stabilization and fingers fixing. The two functions distributedly check the validity of the network and fix it when it is necessary. The ActionSim timers are used for triggering these functions. Chord implementation is summarized in **Figure 23**. Stabilization and fingers fixing functions are shown in **Figure 24**.

```
when node n stabilizes:
        n sends Predecessor Query to n.successor

when node n fixes fingers:
        n sets n.fingerIndex = fingerIndex + 1
        if n.fingers[n.fingerIndex] is in (n.id, n.successor] then
                n sets n.fingers[n.fingerIndex] = n.successor
        else n send Successor Query to n.successor where
                query.key = n.fingers[n.fingerIndex]
```

Figure 24 Chord Stabilization and Finger Fixing

A Chord node provides a small interface for the use of upper layers. An upper layer entity can send messages with Chord IDs as their destinations, and it gets notified when a message arrives. A Chord node also provides an interruption point for upper layers. Before forwarding a message to its next destination, the Chord node notifies the upper layer and asks for approval to continue. This way, the upper layer may stop propagation of the message and can take actions according to the forwarded message.

4.3 SCRIBE

Scribe nodes are implemented as wrappers around Chord nodes. Scribe protocol uses three messages which are carried by the underlying Chord overlay. These messages are Publish, Subscribe and Unsubscribe. Scribe messages are derived from Chord Message. Publish message contains the key of the topic and the payload to be published. Subscribe and Unsubscribe contains the key of the topic.

Scribe constructs reverse multicast trees by recording the previous node of Subscribe messages. When a Subscribe message passes through a Scribe node, the sender of the message is recorded with the key of the subscription. When a Scribe node receives a Publish messages, it directly sends the message to all nodes that are recorded with the key of the publication.

```
node n recieves Publish Message m:
        if n.subscriptions contain m.topic then
                n notifies application for m
        for x in n.children[m.topic] do
                n sends m to x

node n recieves Subscribe Message m:
        n.children[m.topic].add(m.sender)

node n recieves Unsubscribe Message m:
        n.children[m.topic].remove(m.sender)

before node n forwards Subscribe Message m:
        n.children[m.topic].add(m.sender)
        n allows forwarding of m

before node n forwards Unsubscribe Message m:
        n.children[m.topic].remove(m.sender)
        n allows forwarding of m
```

Figure 25 Summary of Scribe Implementation

A Scribe node takes place in the publication of topics that are not in the interest of its upper layers. Therefore, it should not notify upper layers about the messages that they are not interested in. To do so, A Scribe node keeps the list of topics that are in the interest of upper layers. Therefore, a Scribe node is composed of a Chord node, a map for keeping children nodes on reverse multicast trees of topics and a list of topics that the upper layers are interested in. Scribe implementation is summarized in **Figure 25**.

The functionality provided by Scribe is accessed via an interface of three methods. A user may subscribe to a topic, unsubscribe from a topic and publish to a topic by using this interface.

4.4 PHANEROS

Phaneros implementation is built on top of the stack explained in the previous three sections as the topmost layer. Phaneros protocol consists of seven messages:

Cell Enter: Cell Enter message contains entering peer's ID, its position and the cell it enters. This message is published as Scribe message on topics assigned to cells.

Cell Exit: Cell Exit message contains exiting peer's ID and the cell it leaves. This message is published as Scribe message on topics assigned to cells.

Update: Update message contains moving peer's ID and its new position. This message is transmitted directly between nodes.

Tile Query: Tile query message contains querying peer's address for direct connection and the identification of the cell corresponding to the required tile. This message is published as Scribe message on topics assigned to cells.

Tile Available: Tile Available message contains identification of the cell corresponding to the required tile. This message is transmitted directly between nodes.

Tile Request: Tile Request message contains identification of the cell corresponding to the required tile. This message is transmitted directly between nodes.

Tile Envelope: Tile Envelope message carries the actual tile content. This message is transmitted directly between nodes.

```
peer p receives Tile Envelope message e:
    if e.tile is null then
        p fetches from server
    else p.cache.add(e.tile)

peer p receives Update message u:
    p.peerPositions[u.peerId] = u.position

peer p receives Tile Available message m:
    if not p.cache.contains(m.tile) then
        p sends Tile Request to m.sender

peer p receives Tile Request message r:
    if p.cache.contains(r.tile) then
        p sends Tile Envelope to r.sender

peer p receives Cell Enter message m:
    p connects to m.sender
    p.peerPositions[m.peerId] = m.position

peer p receives Cell Exit message m:
    p disconnects from m.sender
    p.peerPositions.remove(m.peerId)

peer p reveices Tile Query message q:
    if p.cache.contains(q.tile) then
        p sends Tile Available to q.sender

peer p in cell c moves:
    for n in p.connectedPeers do
        p sends update to n
    if p leaves c and enters d then
        p publishes Cell Exit on topic of c
        p publishes Cell Enter on topic of d
    for x in PVS(c) do
        if not x ∈ PVS(d) then
            p unsubscribes x
    for x in PVS(d) do
        if not x ∈ PVS(c) then
            p subscribes x
    for x in PVS(d) do
        if not x is available in cache then
            for y in PVS(x) do
                p publishes query
```

Figure 26 Summary of Phaneros Implementation

As explained in the third chapter, Phaneros uses publish/subscribe and direct messaging together. AOI management is maintained with Scribe messages and actual position data and 3D content are transmitted with direct messages. Phaneros node accesses interfaces provided by ActionSim node for sending and receiving direct messages and Scribe node for publishing and receiving ALM messages. Implementation of Phaneros is summarized in **Figure 26**.

4.5 VON

```
peer p moves:
    for x in p.aoiNeighbors ∪ p.voronoi.enclosing do
        p sends Update message to x
    p.peerPositions[p] = p.position
    p.voronoi.update(p.peerPositions[p])

peer p receives Update message u:
    p.peerPositions[u.sender] = u.position
    p.voronoi.update(p.peerPositions)
    if distance(p, u.position) > range then
        p.aoiNeighbors.remove(u.sender)
        if not p.voronoi.enclosing.contains(u.sender) then
            p disconnects from u.sender
    else p.aoiNeighbors.add(u.sender)
    if p is boundary neighbor for u.sender then
        for x in p.voronori.enclosingNeighbors do
            if distance(x, u.position) < range or
              x is enclosing for u.peer then
                suggestion.add(x)
        p sends suggestion to u.sender

peer p receives Connection Suggestion message s:
    for x in s.peers do
        p connects to x
        p.peerPositions[x] = x.position
        p.voronoi.update(p.peerPositions)
        if distance(p, x) < range then
            p.aoiNeighbors.add(x)
```

Figure 27 Summary of VON Implementation

A VON node contains a Voronoi diagram, a list of enclosing neighbors and a list of AIO neighbors. Each VON node is built around an ActionSim node. For update dissemination, VON uses two messages: Update message and Connection Suggestion message. Update message contains the moving peer's ID and its position. Connection Suggestion message is used for suggesting new connections and contains a list of VON nodes with their addresses. Implementation of VON is summarized in **Figure 27**.

4.6 FLoD

```
peer p moves in cell c:
    if p leaves c and p enters d then
        for x in PVS(d) do
            if not p.cache.contains(x) then
                for y in p.aoiNeighbors ∪ p.voronoi.enclosing do
                    p sends Tile Query to y for x

peer p reveices Tile Query message q:
    if p.cache.contains(q.tile) then
        p sends Tile Available to q.sender

peer p receives Tile Available message m:
    if not p.cache.contains(m.tile) then
        p sends Tile Request to m.sender

peer p receives Tile Request message r:
    if p.cache.contains(r.tile) then
        p sends Tile Envelope to r.sender

peer p receives Tile Envelope message e:
    if e.tile is null then
        p fetches from server
    else p.cache.add(e.tile)
```

Figure 28 Summary of FLoD Implementation

FLoD works in conjunction with VON implementation. It uses the lists of neighbor peers maintained by VON for querying and fetching 3D content. In addition to the VON messages, the FLoD implementation uses same four messages with Phaneros implementation for content streaming: Tile Query, Tile Available, Tile Request, Tile Envelope. Each message does exactly the same job that they do in the Phaneros implementation. Peers first send queries to all AOI neighbors and all enclosing neighbors. Peer that receive the queries respond with Tile Available messages. Querying peer sends a Tile Request and gets the content in the following Tile Envelope message. Implementation of FLoD is summarized in **Figure 28**.

4.7 TEST ENVIRONMENT

Phaneros tests were performed on a 2.5D pseudo-urban virtual world which is designed for fast execution and easy perception. The world consists of walkable areas and axis-aligned rectangular obstacles on the ground. The design of virtual world does not cause loss of generality. Obstacles in a 2.5D environment can be approximated with rectangular prisms. Convex vertical prisms (CVP) in [44] is an example of such usage. Obstacles are allowed to intersect with each other for allowing generation of complex shaped obstacles.

A 2.5D world can be represented as a 2D height map. The world map is generated as a PNG image by a randomized generator application. Intensities of the pixels on the image represent the height values of the map. The PNG representation simplifies efficient management of tiles. For efficiency, all nodes in the simulation access the very same tile instances. Tile are read-only and simulation runs on a single computer. Therefore, there is no need for replicating tile instances. During initialization, the PNG map file is read once and tiles are initialized as single instances of sub-images with necessary metadata. All simulation nodes use the same tile instances. This way the memory requirement of the simulation is greatly reduced. The best view for testing Phaneros is the top-down view. It provides all necessary visualization without significant performance load. The complete sample map from the renderer is shown in **Figure 29** with peers on it.

Two different offline visibility implementations were made. First one is the basic range visibility used by VON/FLoD and the other one is the PVS implementation used by Phaneros. PVS calculations were performed with a brute-force occlusion horizon method implementation. Results of visibility calculations are stored with map data on the disk. During initialization, the visibility data is read into a 2D array of PVSs corresponding to view-cells. This data is also read-only and a single instance is used by all nodes. Phaneros assumes the availability of the visibility data to all peers. Therefore, all peers are given the visibility data of the whole map. The size of visibility data is insignificant when compared to 3D content. Nevertheless, if it becomes be necessary to incorporate iterative distribution of offline visibility data for some specific applications, it is straightforward to represent from-region visibility in a hierarchical data structure and distribute it on demand.

All of the design decisions explained so far were made with performance consideration at the first place. The main motivation behind the design was to perform tests with crowded environments in easily accessible hardware. The effort yielded less than 10 minutes running time for 300 seconds of simulation with 1000 peers on a 16GB 2.3 GHz i7 laptop.

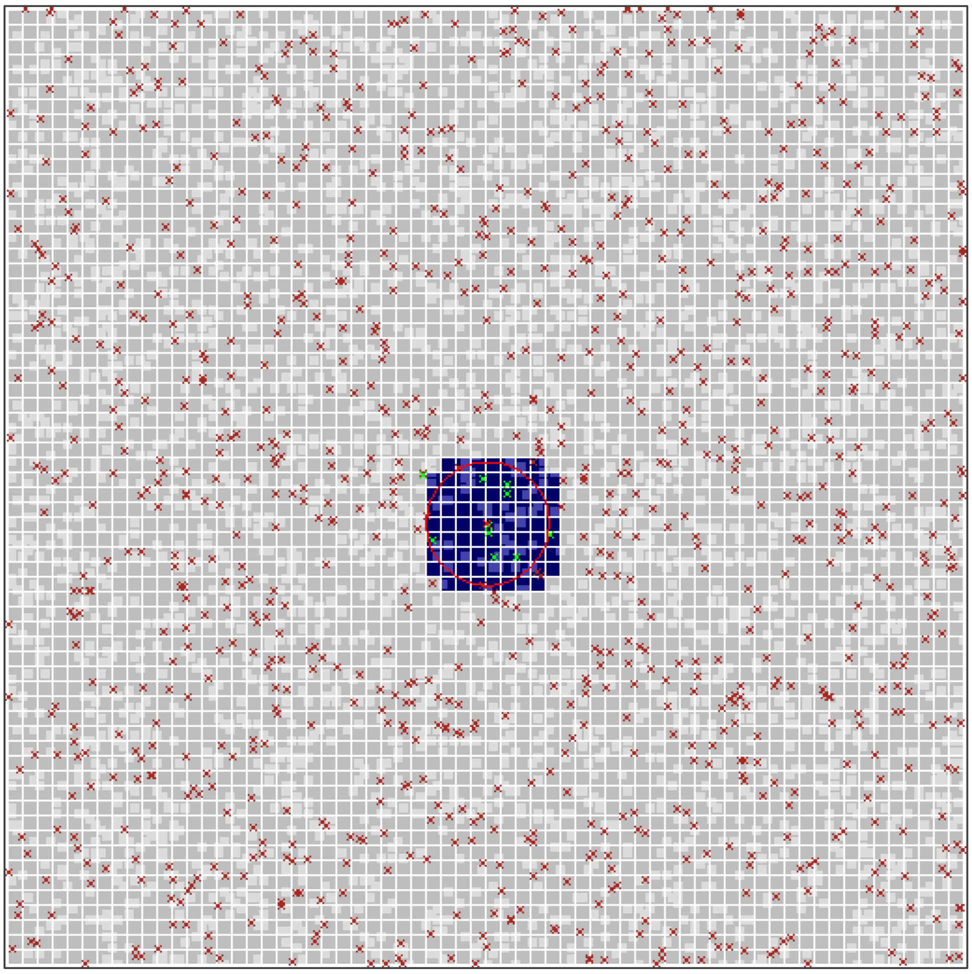

Figure 29 Sample Map Rendering with Peers: Complete map with 1000 peers, view-cells and view of a sample peer.

4.8 TEST CASES

Phaneros tests were performed on a randomized 1024 meters by 1024 meters wide world with 30% of its available area is filled with obstacles in various shapes and sizes. The world is divided into 4096 equal sized square cells that are 16 meters by 16 meters wide. Visibility was computed with 50 meters of range.

Connection delays and network failures are inescapable for any method and they are equally effective on the compared methods. Network latency and failures cannot be avoided. Phaneros is expected to tolerate failures to a certain degree and then fail on providing consistency if there are more failures. Therefore, no tests were performed to demonstrate the obvious. Effects of network latency and node failures and the recovery approaches are discussed in CHAPTER 4.

The size of the 3D content in each view cell is assumed to be 512 kilobytes. The Scribe messages used by Phaneros are assumed to be 1 kilobyte, much larger than the actual content. Internal Chord messages at the lower level are assumed to be 0,04 kilobytes, the size of TCP/IP overhead. With 1000 peers and 2048 kBps bandwidth, a peer receives 127 Scribe messages per second, which occupies 127 kilobytes per second bandwidth on the average. Our tests show that 6.5% of these messages received by a peer are not relevant for it and forwarded to other peers without further processing. Bandwidth consumed by internal Chord messages are insignificant since the message sizes are very small and the messages are actively used only when peers join or leave the overlay network.

Simulations were started at a stabilized state where all required content for their initial positions has been already fetched. The simulation was run for 300 seconds in each test case in terms of simulation time. For the cases involving Chord, the layout network is also stabilized before starting the simulation.

3D streaming was done with packages of 512 kilobytes, which are assumed to contain whole content in a single view cell, with no further fragmentation. Further fragmentation strategies, the level of detail management, model compression and prefetching strategies are not in the scope this study.

Table 1 List of Assumptions

Parameter	Assumption
World size	1024 m x 1024 m
Cell size	16 m x 16 m
Occluders density	30% of the total available area
3D content size	512 kB per cell
Visibility	Precomputed and available on all nodes
Network latency	Only affected by bandwidth
Network failures	None
Size of Chord messages	0,04 kB
Size of Scribe messages	1 kB

Offline visibility information is assumed to be precomputed and available on all nodes. Compared to the actual content, size of the visibility information is insignificant. For our test cases the size of the complete visibility information has the size of 1.5 megabytes. Nevertheless, the visibility information is quite manageable as chunks. For extreme situations, visibility data itself can be fetched in chunks like the 3D content. Assumptions of the test environment are summarized in **Table 1**.

Eleven test cases were constructed for different purposes. Each test case and their varying parameters are explained below. A list of test cases is presented in **Table 2**.

Primary purpose of the tests is comparing the performance of Phaneros to the performance of VON/FLoD. Therefore, first group of test cases aims comparing two methods under different conditions that are created by varying upload bandwidth allocation and the number of peers. Number of peers were chosen as 250 and 1000. On the defined area, casting 250 peers creates a level of density which is low enough to differentiate between Phaneros and VON/FLoD on sparse case and high enough to demonstrate performance of both. If the number if lower, there is no significant load for both of the methods. If the number is higher, some of weaknesses of VON/FLoD disappears. On the same area, casting 1000 peers creates a density which is high enough to differentiate between Phaneros and VON/FLoD and low enough to allow monitoring of the area. If the number is higher the simulation speed goes lower than convenient. If the number is lower operational characteristics of the methods are not demonstrated clearly. In a similar manner, bandwidths are chosen as 1024 kBps and 2048 kBps. These values are chosen to be both realistic and demonstrative. 1024 kBps is low enough to be restrictive and high enough to allow operation of the methods. 2048 is high enough to differentiate between levels of bandwidth restrictions and low enough to be realistic. First four test cases are in this group.

Table 2 List of Test Cases

ID	Upload (kBps)	Download (kBps)	Peers	Tile Size kB	Speed (m/s)	Hotspots	Visibility	Framework
1	2048	-	1000	512	2	0	Yes	Phaneros VON/FLoD
2	1024	-	1000	512	2	0	Yes	Phaneros VON/FLoD
3	2048	-	250	512	2	0	Yes	Phaneros VON/FLoD
4	1024	-	250	512	2	0	Yes	Phaneros VON/FLoD
5	2048	-	1000	512	2	4	Yes	Phaneros
6	1024	-	1000	512	2	4	Yes	Phaneros
7	2048	-	1000	512	1/2/4	0	Yes	Phaneros
8	1024	-	1000	512	1/2/4	0	Yes	Phaneros
9	2048	8.192 to 20.480	1000	512	2	0	Yes	Phaneros
10	2048	-	1000	512	2	0	No	Phaneros
11	2048	-	1000	256 to 2.560	2	0	Yes	Phaneros

Next group of test cases were constructed to analyze the behavior of Phaneros with existence of hotspots. Hotspot test setting introduces 4 hotspots which change their locations every 60 seconds. When hotspots change their locations, peers randomly choose their target hotspots and try to reach them. The pathfinding algorithm was implemented in a way that creates hot regions along the common paths followed by peers. The pattern generated by the path finding algorithm is illustrated on **Figure 30**. Fifth and sixth cases test effects of hotspots in the crowded environment with low and high bandwidths.

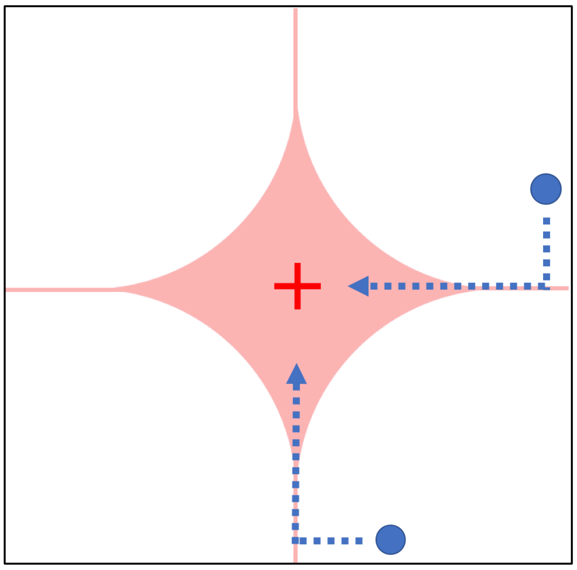

Figure 30 Hotspot Movement Pattern: The red cross in the middle is the hotspot. The pink star highlights the hot region formed by the movement pattern of the peers. Blue circles are two peers and the dotted arrows shows their movement according to the pathfinding algorithm.

Next group of test cases were constructed to analyze the behavior of Phaneros with the existence of peers with different movement speeds. Varying speed test setting introduces slower and faster peers. At the beginning of the simulations, peers randomly choose among 1 meter per second, 2 meters per second or 4 meters per second as their speeds. For all test cases, peers are spawned randomly on the walkable areas. Seventh and eighth cases test the effects of varying walking speeds in crowded environment with low and high bandwidths.

For the cases except the hotspots case, peers walk in randomly selected directions until they reach an obstacle. Upon reaching an obstacle, peers choose a new direction and continue walking.

Please note that the bandwidths mentioned so far are upload limits. Generally, user connections are asymmetric subscriptions with download speeds much higher than the upload speeds. Because of the asymmetry, the bottleneck for the methods is upload bandwidth. We designed the test cases so far with limited bandwidths and unlimited download bandwidths. However, the tests would not be complete without an analyzing the effects of download bandwidth limitation. Therefore, the ninth test case was constructed to analyze the effect of download bandwidth.

Another aspect of Phaneros that needs inspection is the performance of the sole communication scheme. Test cases so far covered the performance of the communication scheme together with the optimization gained by the visibility culling. Tenth test case was constructed to test the performance of Phaneros without the advantages of visibility.

Final test case is constructed to analyze the effect of 3D content size per tile. All other test cases so far assume the tile size to be 512 kilobytes. Eleventh test case tests Phaneros performance with increasing tile content sizes.

CHAPTER 5

TEST RESULTS AND DISCUSSION

Test results show that visibility culls out about 40% of the AOI compared to visibility unaware AOI. On the crowded case, the reduction in the AOI reflects as 17% reduction in AOI neighbors and 20% reduction in the number of simultaneous connections used for update messages. On the sparse case, the reduction in the number of simultaneous connections exceeds 50%. The number of boundary enclosing neighbors is much higher compared to AOI neighbors for VON in the sparse case. Therefore, the number of redundant connections avoided by Phaneros is higher. In accordance with the reduction in simultaneous connections, Phaneros reduces the number of update messages by 26% at least. Reduction exceeds 52% on the sparse case. The improvements on AOI management and number of update messages are illustrated in **Figure 31**.

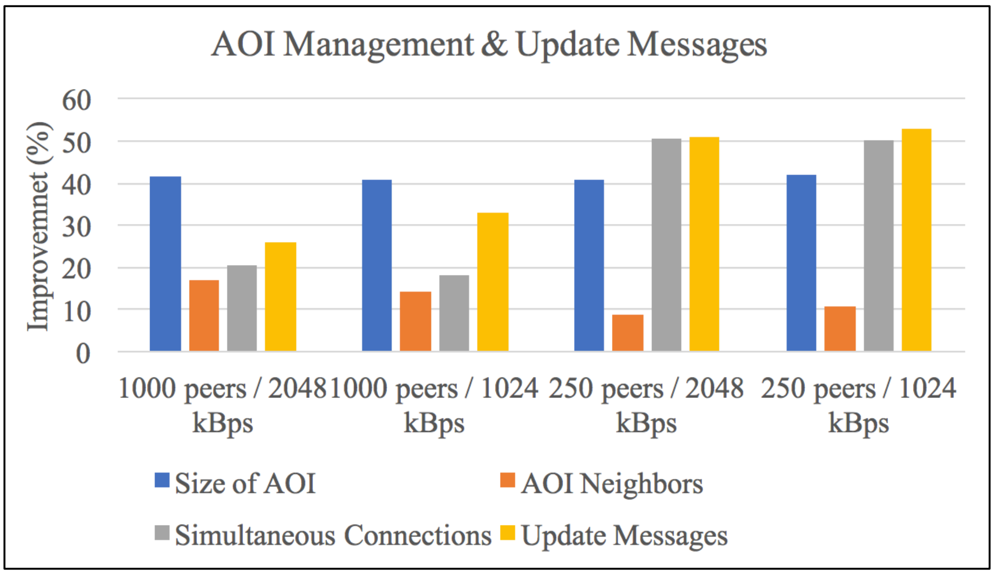

Figure 31 Improvements on AOI Management and Number of Update Messages

Visibility based 3D content streaming performs dominantly better than the visibility unaware alternative. Phaneros streams 100% of the content from the peers in the crowded case and 98.5% in the sparse case regardless of the bandwidth. FLoD achieves 84% in the crowded case and 97.5% in the sparse case but drops to 53% and 75% respectively when the bandwidth is halved. Therefore, Phaneros achieves more than 99% reduction when the number of peers is high. Moreover, utilizing visibility also reduces the number of fetches made. A Phaneros peer requests content for 5.6 view cells whereas a FLoD peer requests 8.7 on the average. The reduction achieved by Phaneros is 35%. Besides the source of the fetches, the delays are also important. Phaneros causes 1.2 to 1.3 second delay consistently whereas FLoD causes 0.5 to 5.1 second delay. The only case where FLoD has lower delay is the sparse case with high bandwidth allocation. Within the first 2 seconds following a view cell transition, Phaneros achieves to fetch all of the required content, whereas FLoD still has 4.6 to 12.5 cells missing depending on the bandwidth allocation. The improvements on 3D content fetching are illustrated in **Figure 32**.

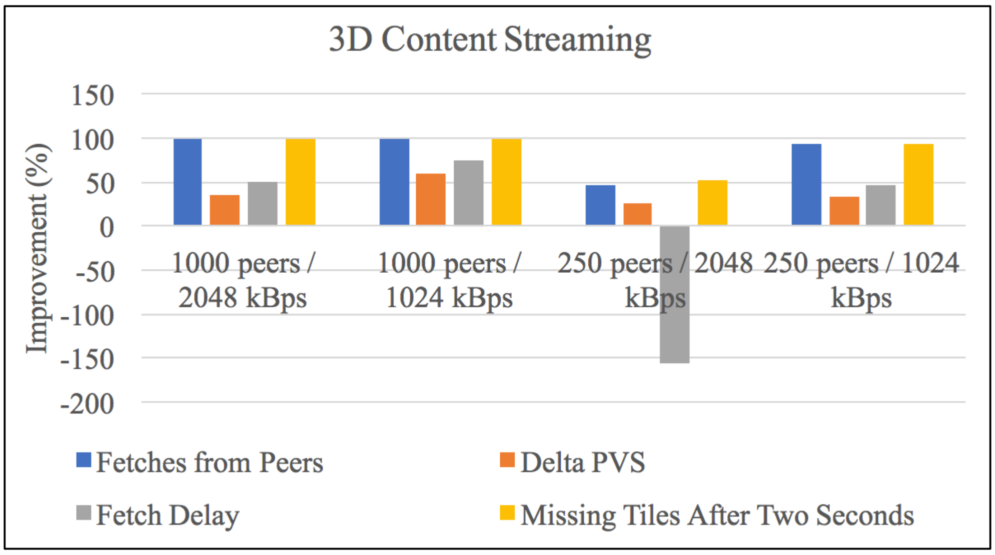

Figure 32 Improvements on 3D Content Streaming

Without instrumenting a prefetching method, a delay between cell transitions and retrieval of the content is inevitable since transmission of the content does not happen instantly. This result shows that if a prefetching method is utilized and it is capable of compensating for the two seconds of delay, Phaneros is capable of delivering flawless use experience. Please note that the result is valid for the test case and depends on the availability of extra bandwidth for prefetching. The actual performance requirements of the prefetching method are application specific.

The improvements obtained by Phaneros are dependent on how much of the world is culled out by visibility. The longer range of visibility and the higher rate of occlusion ensure higher improvements. **Figure 33** shows a sample scene from Phaneros simulation to illustrate the effects of visibility culling.

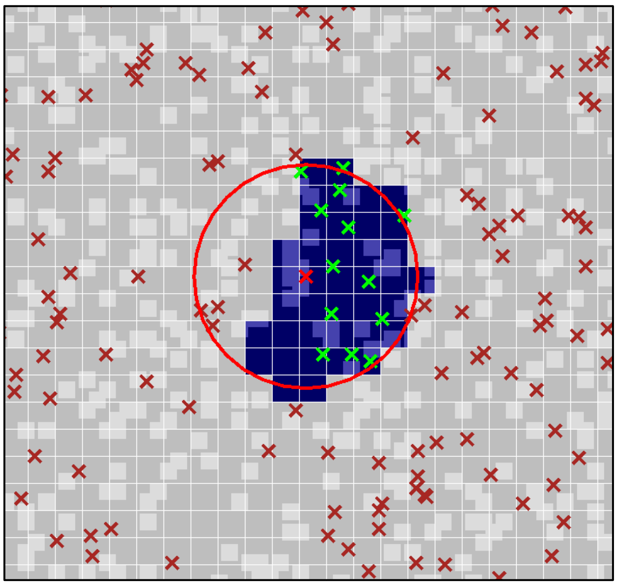

Figure 33 Phaneros Sample Scene: Darker shaded cells are the AOI of the peer on the center. The peer is connected to peers shown with green crosses. The red circle shows the visibility range. The light shaded cells and the dark red peers in the circle are culled

Two exceptional cases in the results might be interesting for the reader. It can be seen that FLoD unexpectedly generates lower average fetch delay than Phaneros when there are 250 peers and 2048 kBps bandwidth available. The average delay for Phaneros is

1.22 seconds whereas it is 0.48 seconds for FLoD. For fetching, the advantage of Phaneros over FLoD is that Phaneros chooses source peers according to the visibility while FLoD chooses source peers randomly among many connections. If any of the neighbor peers has the requested content, the lower number of peers increases the chance of finding the right peer quicker for FLoD. Moreover, since the bandwidth is high and the number of peers is low, the other advantage of Phaneros, which is fetching from the fastest responding (least loaded) peer, vanishes as there is plenty of bandwidth for the FLoD peer which is much more loaded on crowded case. The comparison results collected from tests are given in **Table 3**.

Table 3 Comparison Results: Results are collected from multiple runs of Phaneros (Phan.) and VON/FLoD for crowded and sparse cases with higher and lower bandwidth allocations. Improvement (Imp.) obtained by Phaneros is shown in percentage.

Number of Peers	**1000 Peers**						**250 Peers**					
Allocated Bandwidth per Peer	**2048 kB**			**1024 kB**			**2048 kB**			**1024 kB**		
Framework	**Phan.**	**VON FLoD**	**Imp. (%)**	**Phan.**	**VON FLoD**	**Imp. (%)**	**Phan.**	**VON FLoD**	**Imp. (%)**	**Phan.**	**VON FLoD**	**Imp. (%)**
Average size of AOIs in terms of view cells	42.08	71.79	**41.38**	42.28	71.31	**40.71**	42.38	71.47	**40.70**	42.08	72.33	**41.82**
Average number of peers inside the AOI of a peer (AOI Neighbors)	10.85	13.07	**16.98**	10.98	12.79	**14.16**	3.50	3.84	**8.83**	3.47	3.89	**10.86**
Average number of active connections a peer has simultaneously	10.85	13.63	**20.39**	10.98	13.42	**18.16**	3.50	7.06	**50.40**	3.47	6.98	**50.31**
Average number of update messages sent by a peer per second	21.12	28.58	**26.11**	21.39	31.91	**32.97**	6.80	13.81	**50.77**	6.74	14.28	**52.81**
Ratio of content packages fetched from server to content packages fetched from peers (%)	0.07	15.92	**99.58**	0.07	47.10	**99.85**	1.48	2.78	**46.86**	1.52	24.20	**93.70**
Average number of content packages requested after a cell transition	5.61	8.68	**35.36**	5.69	14.01	**59.39**	5.46	7.39	**26.11**	5.39	8.15	**33.89**
Average time passed between querying of a content package and its retrieval in milliseconds	1317.61	2642.61	**50.14**	1320.29	5116.65	**74.20**	1222.63	478.35	**-155.5**	1218.25	2281.48	**46.60**
Average number of missing content packages after 2 seconds following the view cell change	0.02	4.63	**99.57**	0.02	12.45	**99.84**	0.23	0.49	**52.80**	0.23	3.96	**94.19**

The other interesting point is also related to the first exceptional case. In the results, it is seen that the ratio of the content streamed from the servers decreases with decreasing number of peers for FLoD while it is the opposite for Phaneros. The explanation given for the first case is also applicable to this case. The lower number of peers allows FLoD to find a source peer quicker and availability of bandwidth allows it to deliver the content in time. As it is seen from the results, the tested bandwidths are sufficient for Phaneros in all cases. Therefore, Phaneros acts as expected and performs better when there are more peers to provide required content.

The introduction of hotspots affects the simulation as if the number of peers doubled. The increased density of peers doubled the average number of AOI peers, simultaneous connections and update messages. The average fetch delay stayed at the same level. Although the total number fetches made from server increased, the ratio of such fetches stayed below 0.1%. When the maximum numbers considered, the effects of hotspots become more visible. The maximum number of AOI peers increased from 30 to 85. Maximum fetch delay increased from 3.2 seconds to 5.5 seconds. When bandwidth lowered from 2048 kBps to 1024 kBps, average fetch delay increased slightly to from 1.3 seconds to 1.7 seconds but maximum fetch delay increased significantly from 5.5 seconds to 104 seconds.

When peers allowed to move at varying speeds, the average time spent by a peer in a cell decreased from 7.2 seconds to 5.8 seconds. Consequently, the ratio of fetches made from server doubled however, the ratio is still below 0.1%. When the bandwidth is lowered from 2048 kBps to 1024 kBps, the ratio of fetches from server double once more but again stayed under 0.1%.

The introduction of faster peers decreases the available time slot for fetching and the introduction of hotspots increases the required bandwidth. Phaneros does not create extra sensitivity for both cases. However, these factors should be considered when deciding application requirements and range of visibility.

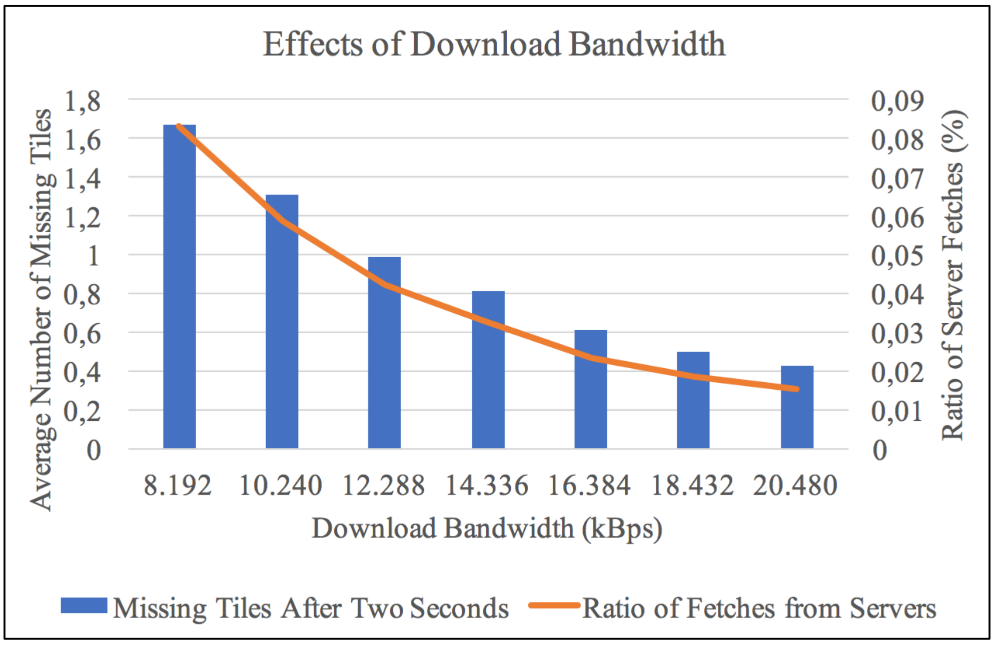

Figure 34 Effects of Download Bandwidth Limitation

Results obtained from download bandwidth tests are presented in **Figure 34**. It is seen that with increasing bandwidth, number of missing tiles after two seconds following a cell transition and the ratio of fetches made from servers decrease gradually. It can also be seen that the download bandwidth limitation increases the magnitude of its effect as the it gets lower.

The effects of increasing tile content size is depicted in **Figure 35**. The results show that up to a certain size (1280 kilobytes) Phaneros is not affected in the test configuration. However, larger sizes cause missing tiles of two seconds and the ratio of server fetches increase linearly. It can be concluded that as long as there is enough download bandwidth increasing tile content size does not affect the performance. As discussed earlier, tile content size is only a relative measure against available bandwidth regardless of the utilized method.

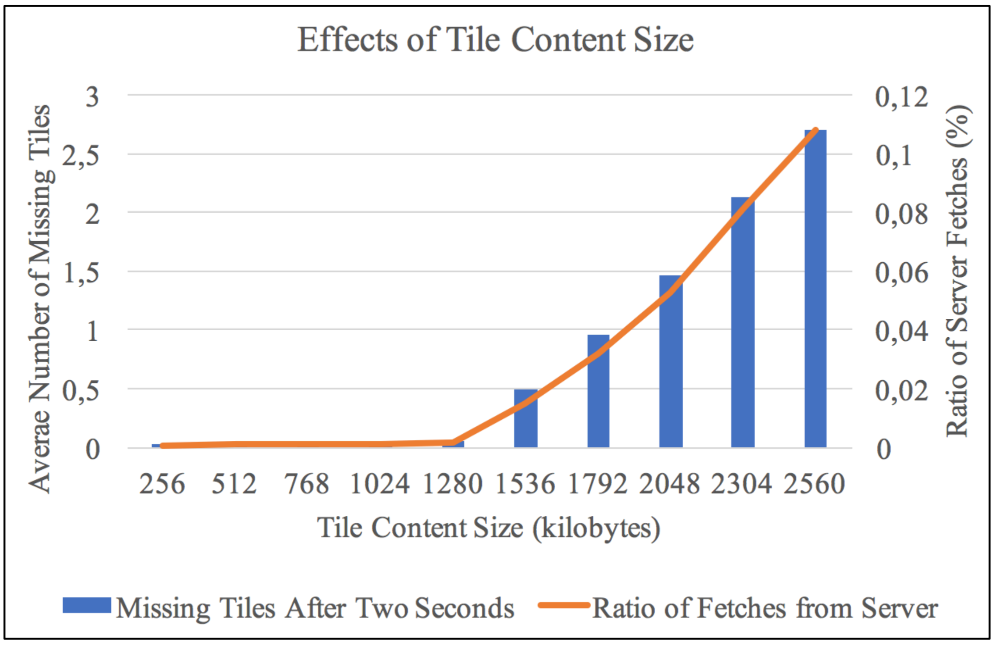

Figure 35 Effects of Tile Content Size

In the last test case where Phaneros is run without visibility optimization, the results show that Phaneros performs slightly better (5%) than VON in terms of the number of simultaneous connections and the number of update messages. The difference is caused by the enclosing nodes of VON which are out of the visibility range. This situation was discussed earlier on **Figure 5**.

The performance comparison and evaluation presented so far based on the selection of eight indicative measurement items among a larger set. The complete set of measurements is presented below. **Table 4** lists the measurements and their definition. When applicable, for each definition one or more of the five values are collected: number of samples taken, the sum of all samples, minimum sample, maximum sample and the average of the samples.

Table 4 Definition of Measurements

Measurement	Definition
AOI Neighbors	The number of peers that are simultaneously in the AOI of the peer. For Phaneros the number indicates the number of peers in the visible cells and for VON/FLoD the number indicates the number of peers in the visible range. The measure is an indicator for the accuracy of the utilized AOI determination method. The coarser is the method the higher is the number.
Cell Transitions	The number of cell transitions made. When a peer leaves a from one cell to another, this value is increased by one. The frameworks compared in this study introduces overhead when cell transitions occur. The measure is an indicator which shows that the cases impose equal loads to the compared methods. In other words, the movement patterns of the peers do not change between test.
Delta PVS	When a cell transition occurs, the set of cells visible from the old cell and the set of cells visible from the new cell are not the same. Usually, most of the sets intersect except some cells. Some cells become invisible and some cells become visible with the transition. Delta PVS is the number of cells that were not visible from the old cell and visible from the new cell. The measure is an indicator for the load caused by cell transitions. In general, it is expected that the smaller AOI results in Delta PVS.
Fetch Delay	The amount of time passed between querying for the content in a cell and the actual delivery of the content in seconds. The measure is a major indicator for the performance of the 3D content streaming function. Lower values indicate better performance.
Missing Tiles After Two Seconds	The number of cells whose content is not actually delivered to the requiring peer two seconds later the querying. The measure is a major indicator for the performance of the 3D content streaming function. Lower values indicate better performance.

Table 4 Continued

Measurement	Definition
PVS Size	The number of cells that are simultaneously in the AOI of the peer. For Phaneros the number indicates the number of visible cells in the PVS. For VON/FLoD the number indicates the number of cells in the visible range. The measure is an indicator for the accuracy of the utilized AOI determination method. The coarser is the method the higher is the number.
Query Hops	The number of peers that a tile query visited before reaching its destination. The measurement is an indicator for the overhead generated by the querying phase of Phaneros 3D content streaming function. The measurement is not applicable for VON/FLoD.
Server Fetch Because Of NULL Envelope	The number of fetches made from the server because of Time Envelope messages delivered with null content. This case occurs when a peer responds positively to a query and the requested content is discarded before its delivery. The measurement indicates that either the cache size is too small of the speed of the fetching process is too slow to cope with the demanded 3D content streaming performance.
Server Fetch Because Of Timeout	The number of fetches made from server because of missing tiles after two seconds. The measure in an indicator for underachieving 3D content streaming performance.
Server Fetch Because Of Urgent	The number of fetches made from the server because of tiles that a peer wants to go in before the tile is fetched. The measure in an indicator for underachieving 3D content streaming performance.
Simultaneous Connections	The number of peers that are simultaneously connected by the peer. The measure is an indicator of the efficiency of update dissemination function. The lower values mean better optimization in the number of simultaneous connections.

Table 4 Continued

Measurement	Definition
Subscriptions	The number of subscription calls made by peers. Only applicable for Phaneros. The measurement is an indicator for overlay maintenance overhead.
Suggestions	The number of suggestion messages sent by peers. Only applicable for VON/FLoD. The measurement is an indicator for overlay maintenance overhead.
Tiles from Peers	The number of tiles whose content is fetched from peers. The measurement, together with Tiles from Server measurement, is an indicator of 3D content streaming performance. The higher ratio of tiles from peers indicates better performance.
Tiles from Server	The number of tiles whose content is fetched from server. The measurement, together with Tiles from Peers measurement, is an indicator of 3D content streaming performance. Lower ratio of tiles from servers indicates better performance.
Updates Sent	The number of update messages sent by the peer. The measurement is an indicator for update dissemination efficiency. Lower values indicate better performance optimization.

Measurements collected with Phaneros are represented in **Table 5** and **Table 6** presents the measurements collected with VON/FLoD.

It can be seen that maximum number of AOI neighbors are similar for both while Phaneros has less neighbors on the average. This is expected since the worst case of visibility culling should produce same results with range-based visibility. The comparison of the values is presented in **Figure 36**.

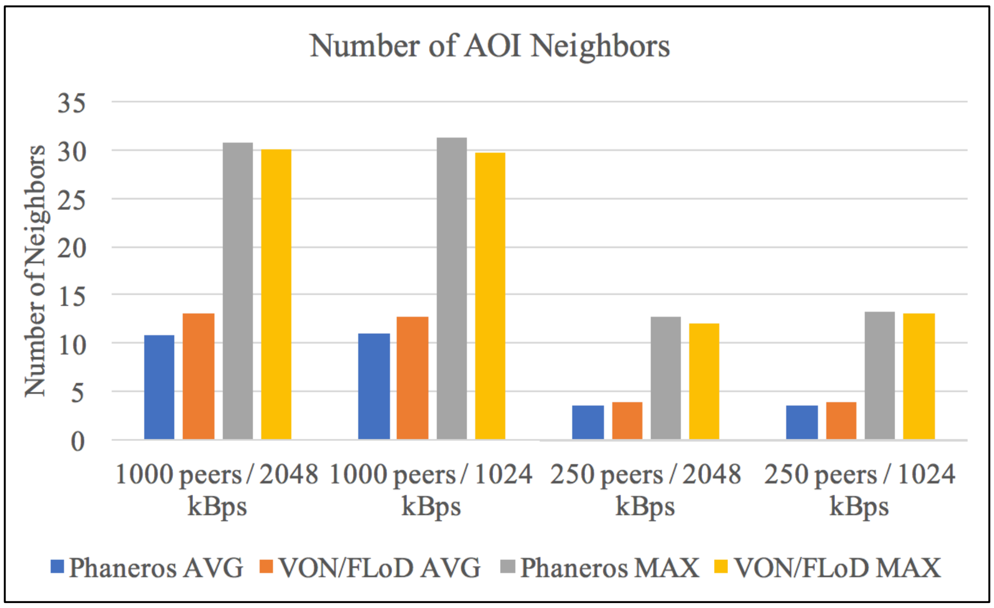

Figure 36 Comparison of AOI Neighbors

Both methods are tested with same maximum range of visibility. Number of cell transition and measurements of time spent in a cell is equal for both methods. It can be concluded that the tests were performed under fair conditions.

PVS size and Delta PVS measurements show parallelism as one might expect. Set of differences between larger PVS is larger than the set of differences between smaller PVS. Maximum Delta PVS and Average Delta PVS measurements reflects the difference between accuracy of visibility methods. The comparison of the results is presented in **Figure 37**.

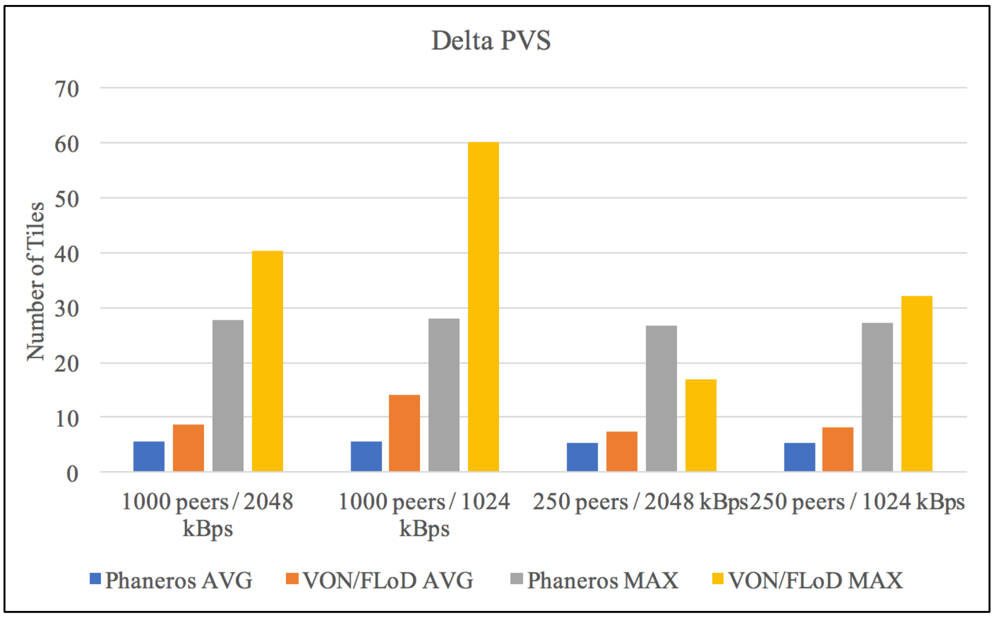

Figure 37 Comparison of Delta PVS

Fetch delay values reflect the better performance of Phaneros over VON/FLoD. The anomaly on the case with less peers and high bandwidth is discussed in CHAPTER 5. Another interesting result is that maximum fetch delay is significantly higher with than the average fetch delay for VON/FLoD whereas Phaneros deviation stays in much smaller margin. The comparison of the results is presented in **Figure 38**.

As discussed earlier in CHAPTER 5, Phaneros is much better at identifying candidate source peers and choosing the best among them. The number of missing tiles after two seconds of cell stay time is almost zero with Phaneros. Again, the maximum value shows that Phaneros is much more stable on the worst case than the reference study. The comparison of the results is presented in **Figure 39**.

Query hops measures the number of hops made by a query message before it reaches to its destination. Results show that the implementations of Chord and Scribe are accurate. Scribe messages reaches their destinations in *O(logN)* hops on the average and *2O(logN)* hop on the worst case.

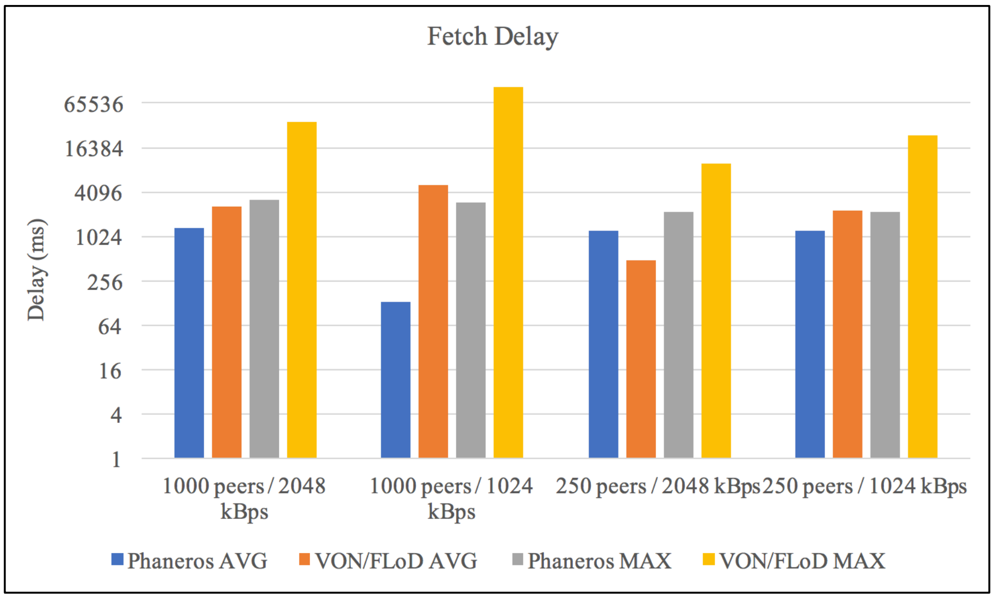

Figure 38 Comparison of Fetch Delay

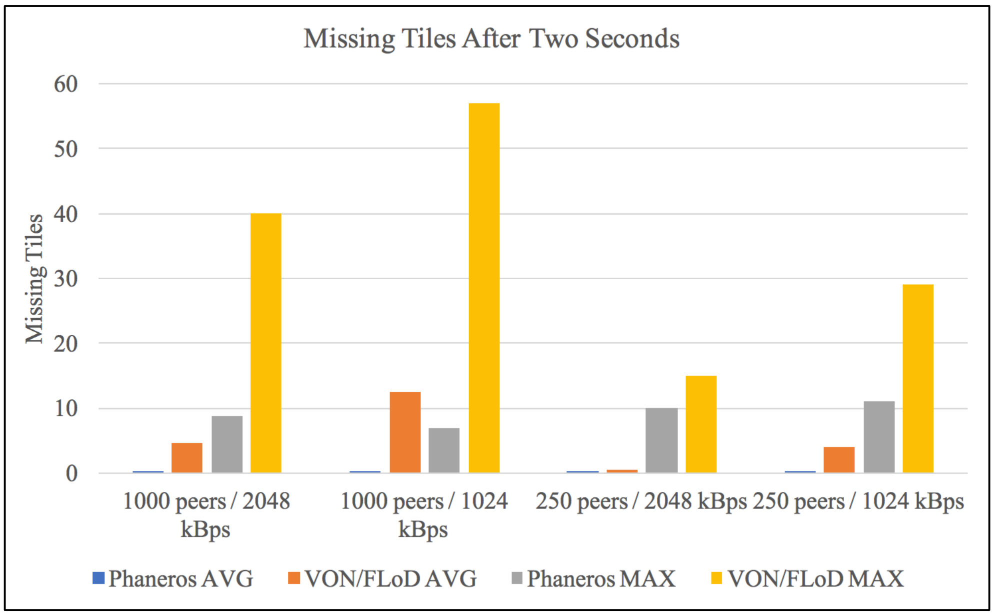

Figure 39 Comparison of Missing Tiles After Two Seconds

The number of suggestions shows the number of neighbor suggestions made by VON. This reflects the overhead of maintaining VON. Although a numeric comparison may not be meaningful, it is comparable to the overhead generated by multi-hop query messages sent by Phaneros.

The number of fetches made form peers and servers and the ratio between these measurements are discussed in detail before in CHAPTER 5. The comparison of the results is presented in **Figure 40**.

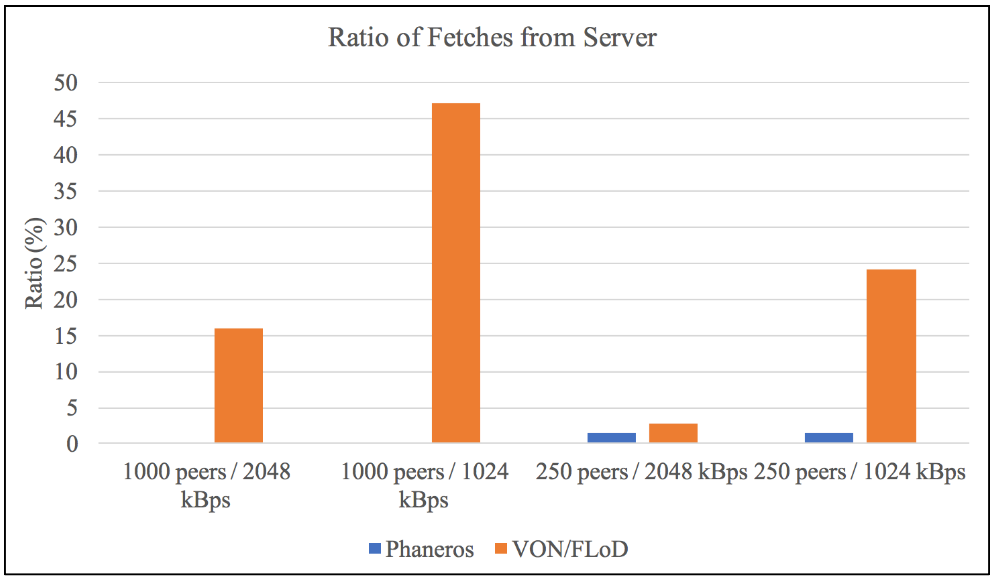

Figure 40 Comparison of Fetches from Server

The number of simultaneous connections and the number of update messages represents the optimization provided by Phaneros over VON/FLoD. Reasons of the difference is presented in CHAPTER 2 and the results are discussed in detail in CHAPTER 5.

Table 5 Phaneros Measurements

Measurement	**1000 peers 2048 kB**	**1000 peers 1024 kB**	**250 peers 2048 kB**	**250 peers 1024 kB**
AOI Neighbors AVG	10.85	10.98	3.50	3.47
AOI Neighbors MIN	0.00	0.00	0.00	0.00
AOI Neighbors MAX	30.67	31.33	12.67	13.33
Cell Transitions	42,026.33	41,845.67	10,569.33	10,498.67

Table 5 Continued

Measurement	1000 peers 2048 kB	1000 peers 1024 kB	250 peers 2048 kB	250 peers 1024 kB
Cell Stay AVG	7,208.22	7,235.89	7,162.07	7,208.34
Cell Stay MIN	500.00	500.00	500.00	500.00
Cell Stay MAX	70,666.67	62,166.67	60,333.33	62,833.33
Delta PVS AVG	5.61	5.69	5.46	5.39
Delta PVS MIN	0.00	0.00	0.00	0.00
Delta PVS MAX	27.67	28.00	26.67	27.33
Fetch Delay AVG	1,317.61	1,320.29	1,222.63	1,218.25
Fetch Delay MIN	100.00	100.00	100.00	100.00
Fetch Delay MAX	3,200.00	2,900.00	2,200.00	2,200.00
Missing Tiles After Second AVG	0.02	0.02	0.23	0.23
Missing Tiles After Second MIN	0.00	0.00	0.00	0.00
Missing Tiles After Second MAX	8.67	7.00	10.00	11.00
PVS Size AVG	42.08	42.28	42.38	42.08
PVS Size MIN	6.33	7.33	10.00	10.33
PVS Size MAX	58.00	58.00	57.67	57.67
Query Hops AVG	11.67	11.68	9.69	9.66
Query Hops MIN	0.00	0.00	0.00	0.00
Query Hops MAX	22.00	22.00	20.00	19.33
Server Fetches Because Of NULL Envelope	N/A	N/A	N/A	N/A
Server Fetches Because Of Timeout	848.00	915.33	2,156.67	2,150.67
Server Fetches Because Of Urgent	1.33	0.00	N/A	N/A
Simultaneous Connections AVG	10.85	10.98	3.50	3.47
Simultaneous Connections MIN	0.00	0.00	0.00	0.00
Simultaneous Connections MAX	30.67	31.33	12.67	13.33
Subscriptions	380,091.33	379,436.00	95,671.33	95,227.00
Suggestions	0.00	N/A	N/A	N/A
Tiles from Agents	1,276,071.00	1,297,889.00	143,601.67	138,999.67
Tiles from Server	849.33	915.33	2,156.33	2,150.67
Updates Sent	6,334,843.67	6,417,057.67	509,830.67	505,602.33

Table 6 VON/FLoD Measurements

Measurement	1000 peers 2048 kB	1000 peers 1024 kB	250 peers 2048 kB	250 peers 1024 kB
AOI Neighbors AVG	13.07	12.79	3.84	3.89
AOI Neighbors MIN	1.00	1.67	1.00	1.00
AOI Neighbors MAX	30.00	29.67	12.00	13.00
Cell Transitions	41,682.33	38,922.33	10,419.33	10,465.67
Cell Stay AVG	7,244.63	7,702.74	7,250.89	7,215.25
Cell Stay MIN	500.00	500.00	500.00	500.00
Cell Stay MAX	67,166.67	93,500.00	49,833.33	43,666.67
Delta PVS AVG	8.68	14.01	7.39	8.15
Delta PVS MIN	0.00	0.00	0.00	0.00
Delta PVS MAX	40.33	60.00	17.00	32.00
Fetch Delay AVG	2,642.61	5,116.65	478.35	2,281.48
Fetch Delay MIN	100.00	100.00	100.00	100.00
Fetch Delay MAX	36,433.33	110,533.33	10,133.33	23,800.00
Missing Tiles After Two Seconds AVG	4.63	12.45	0.49	3.96
Missing Tiles After Two Seconds MIN	0.00	0.00	0.00	0.00
Missing Tiles After Two Seconds MAX	40.00	57.00	15.00	29.00
PVS Size AVG	71.79	71.31	71.47	72.33
PVS Size MIN	24.00	24.00	24.00	24.00
PVS Size MAX	77.00	77.00	77.00	77.00
Query Hops	N/A	N/A	N/A	N/A
Server Fetch Because Of NULL Envelope	21,685.67	77,655.67	13.67	940.33
Server Fetch Because Of Timeout	174,789.33	439,267.00	4,577.00	37,480.67
Server Fetch Because Of Urgent	4,289.33	41,282.67	16.33	321.00
Simultaneous Connections AVG	13.63	13.42	7.06	6.98
Simultaneous Connections MIN	2.33	3.00	1.00	1.00
Simultaneous Connections MAX	30.33	30.33	14.67	14.67
Subscriptions	N/A	N/A	N/A	N/A
Suggestions AVG	3.53	3.28	3.09	3.04

Table 6 Continued

Measurement	1000 peers 2048 kB	1000 peers 1024 kB	250 peers 2048 kB	250 peers 1024 kB
Suggestions MIN	1.00	1.00	1.00	1.00
Suggestions MAX	11.67	11.33	8.33	8.33
Tiles from Agents	1,053,585.33	563,467.00	160,779.67	120,946.33
Tiles from Server	199,562.00	501,713.33	4,604.00	38,612.00
Updates Sent	8,573,219.33	9,572,702.33	1,035,523.00	1,071,343.00

Test results without hotspots and varying speeds, results with hotspots and results with varying speeds are compared proportionally to each other in **Figure 41**. All measurements from hotspot tests and varying speed tests are presented in **Table 7** and **Table 8** respectively.

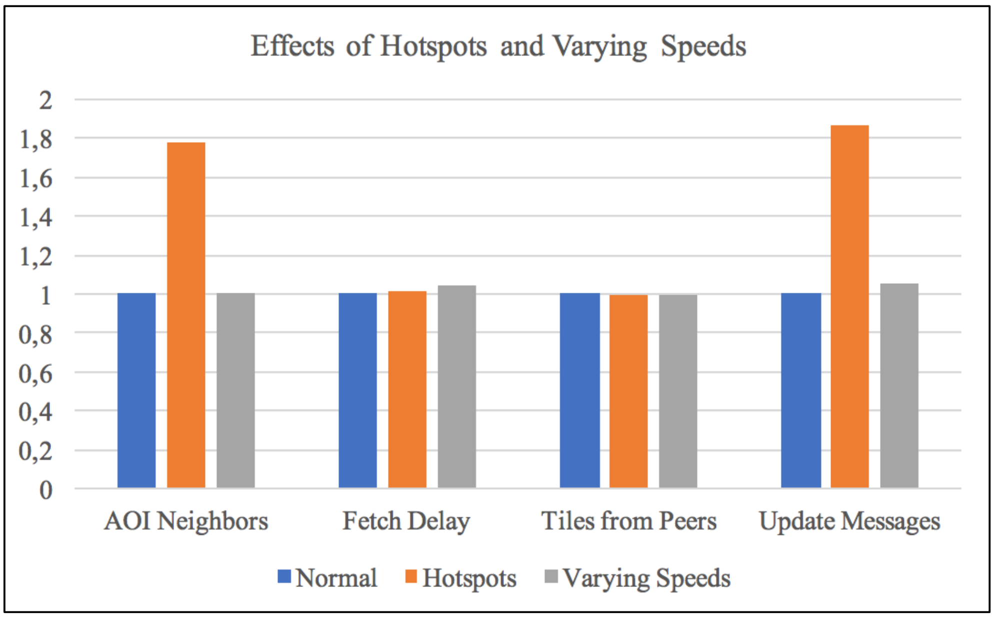

Figure 41 Effects of Hotspots and Varying Speeds

Table 7 Phaneros Measurements with Hotspots

Measurement	1000 peers 2048 kB	1000 peers 1024 kB
AOI Neighbors AVG	19.30	19.55
AOI Neighbors MIN	0.00	0.00
AOI Neighbors MAX	85.00	116.67
Cell Transitions	51,332.00	51,245.33
Cell Stay AVG	5,703.49	5,662.28
Cell Stay MIN	0.00	0.00
Cell Stay MAX	269,166.67	283,666.67
Delta PVS AVG	6.86	7.21
Delta PVS MIN	0.00	0.00
Delta PVS MAX	30.33	37.00
Fetch Delay AVG	1,337.88	1,748.32
Fetch Delay MIN	100.00	100.00
Fetch Delay MAX	5,533.33	103,833.33
Missing Tiles After Two Seconds AVG	0.48	2.15
Missing Tiles After Two Seconds MIN	0.00	0.00
Missing Tiles After Two Seconds MAX	14.00	36.33
PVS Size AVG	43.77	43.72
PVS Size MIN	13.00	10.33
PVS Size MAX	58.00	58.00
Query Hops AVG	11.69	11.62
Query Hops MIN	0.00	0.00
Query Hops MAX	22.00	22.00
Server Fetch Because Of NULL Envelope	16.33	9,454.67
Server Fetch Because Of Timeout	19,095.67	85,882.33
Server Fetch Because Of Urgent	66.00	5,404.00
Simultaneous Connections AVG	19.30	19.55
Simultaneous Connections MIN	0.00	0.00
Simultaneous Connections MAX	85.00	116.67
Subscriptions	515,831.67	512,108.67
Suggestions	N/A	N/A
Tiles from Agents	2,382,301.67	1,791,903.00
Tiles from Server	19,129.33	89,826.00
Updates Sent	11,788,521.33	11,830,753.00

Table 8 Phaneros Measurements with Varying Walking Speeds

Measurements	1000 peers 2048 kB	1000 peers 1024 kB
AOI Neighbors AVG	10.91	10.86
AOI Neighbors MIN	0.00	0.00
AOI Neighbors MAX	32.00	32.67
Cell Transitions	52,097.33	52,861.00
Cell Stay AVG	5,773.96	5,688.04
Cell Stay MIN	0.00	0.00
Cell Stay MAX	76,666.67	88,166.67
Delta PVS AVG	5.37	5.39
Delta PVS MIN	0.00	0.00
Delta PVS MAX	28.67	30.00
Fetch Delay AVG	1,379.60	1,390.42
Fetch Delay MIN	100.00	100.00
Fetch Delay MAX	2,800.00	8,966.67
Missing Tiles After Two Seconds AVG	0.06	0.12
Missing Tiles After Two Seconds MIN	0.00	0.00
Missing Tiles After Two Seconds MAX	10.00	18.00
PVS Size AVG	41.93	41.97
PVS Size MIN	6.00	5.00
PVS Size MAX	58.00	58.00
Query Hops AVG	11.68	11.67
Query Hops MIN	0.00	0.00
Query Hops MAX	22.00	22.00
Server Fetch Because Of NULL Envelope	N/A	10.33
Server Fetch Because Of Timeout	2,420.67	5,042.00
Server Fetch Because Of Urgent	4.33	6.67
Simultaneous Connections AVG	10.91	10.86
Simultaneous Connections MIN	0.00	0.00
Simultaneous Connections MAX	32.00	32.67
Subscriptions	462,582.00	471,004.67
Suggestions	N/A	N/A
Tiles from Agents	1,481,044.33	1,510,651.00
Tiles from Server	2,425.00	5,025.00
Updates Sent	6,651,330.33	6,621,513.67

Another set of measurements were made to evaluate the overhead of using Scribe messaging for a node. **Table 9** presents the average number of messages per second per peer and differentiates between the messages intended for the peer and the messages that are intended for other peers and forwarded.

Table 9 Overhead of Scribe Usage in Phaneros: Scribe Messages per Second per Peer. Delivered column shows the number of messages that are intended for the receiving peer. Total column shows the number all of messages including those are not intended for the receiving peer. These messages are forwarded to next peer without notifying the application.

Scribe Message	Delivered	Total	Overhead Percentage
All Messages	118.83	126.97	6.41%
Cell Enter/Exit Messages	29.39	30.74	4.38%
Subscription/Unsubscription Messages	0.36	3.24	88.86%
Tile Query/Available/Request Messages	89.08	92.99	4.20%

CHAPTER 6

CONCLUSION

This study presents a visibility-based, fully distributed, peer-to-peer (P2P) update dissemination and 3d content streaming framework for massive virtual environments. We name our framework as Phaneros. Phaneros utilizes from-region visibility for Area of Interest (AOI) management and layout management. The regions are defined as fixed-sized, regular grid of cells. Potentially Visible Set (PVS) definition is extended to contain visible cells instead of visible primitives. Each cell is associated with a PVS which contains all cells that are visible from it. PVSs are computed offline and related metadata is distributed to peers when they join the virtual environment.

Phaneros update dissemination layout is composed of two separate messaging schemes that work together. Direct connections between peers are used for transmitting update messages. Application layer multicasting (ALM) is used for transmitting AOI management messages. Direct connections are established according to the messages exchanged through ALM. The fundamental assumption of Phaneros about AOI management is that peers are interested in the portions of the world that are visible to them. In Phaneros terms, peers are interested in cells that are visible from the cell which they reside in. In other words, peers are interested in the cells that are in the PVS of the cell they reside in. An AOI management scheme is built on top of ALM based on this assumption. A topic is created for each cell on the ALM and with these topics, Phaneros AOI management works as a simple publish/subscribe scheme which has two rules: Peers subscribe to the topics that are assigned to the cells in their AOIs. Peers publish to the topics of the old cell and the new cell when they move from one cell to another. This scheme allows peers to know about the other peers in their AOIs and to establish direct connections with them.

Phaneros 3D content streaming uses the same PVS structure. Visibility from cells is a symmetric relation. When a cell is visible from another, the relation also holds in the opposite direction. Using this property of the visibility relation, the PVS of a cell can also be defined as the set of all cells that the cell is visible from. Consequently, the cells in the PVS are sources with high probability to have the content of the cell available in their caches. Phaneros makes use of this heuristic for 3D content streaming. When a peer requires the content of a cell, it publishes a query to the topic of the cell. Since all peers that can see the cell are already subscribed to its topic, peers with the best probability of having the missing content receive the query. Peers that receive the query responds if they have the required content in their caches. Querying peers then fetches the content from the fastest responding peer as the second phase of the 3D content streaming. Choosing the fastest responding peer implicitly chooses the best source because the fastest responding peer is the closer peer with the lower load.

We developed our in-house simulator ActionSim to meet our requirements for testing and comparing Phaneros. We implemented Chord, Scribe, Phaneros, VON and FLoD on top of ActionSim. We compared Phaneros with its well-known and successful counterpart. Our results verified our visibility-based heuristics.

Another aspect of Phaneros is its simplicity. It presents a straightforward approach for facilitating efficient update dissemination and 3D content streaming. This makes Phaneros an applicable framework for real world applications rather than a theoretical study. However, it is obvious that further optimizations will be required.

6.1 FUTURE WORK

Phaneros assumes the occluders to be static because of the offline from-region visibility culling method that it uses. With this approach, the subtraction of existing occluders at run time may affect the consistency. In this sense, Phaneros does not support dynamic environments with its presented state. For such dynamic environments, one of the following approaches may be utilized. If the dynamic part of the world is significantly smaller than the static part, the dynamic part can be excluded from the visibility calculations, surrendering some minor optimization. Another approach may be utilizing an aggressive visibility culling method and assuming the occluders to be smaller than they are. If the modifiable part of the content is limited by the shrinkage (like a layer to be peeled), consistency will be guaranteed. To fully support consistency in highly dynamic content, the utilized offline visibility method should be replaced with an online alternative or existing offline method should be improved with an iterative visibility update method. Independent of Phaneros, the visibility computation costs will vary according to the method utilized and the characteristics of the virtual world.

With its presented state, Phaneros does not include a prefetching scheme. Prefetching schemes require determination of the content that will be needed in the near future. A strategy for choosing cells to be prefetched can be developed with the use of PVS. It is safe to assume that a peer will move into one of the cells it sees. From this assumption, it is derived that PVSs of visible cells are good candidates to be prefetched. When there are sufficient resources, union of the PVSs of the cells in the PVS of the current cell of a peer composes the complete set of cells for the prefetching. When resources are limited, a prioritization strategy can be incorporated according to the movement direction or the statistical movement tendency of the peers [12][45].

First, phase of 3D content streaming provided by Phaneros has an opening for further development. When there are hotspots, it is quite expected that many connections to be established as the response to a content query. Query responses are sent through direct connections because of performance considerations. It may not be necessary to use direct connection when an effective prefetching scheme is utilized. When the prefetching is in action, there will be enough time for sending responses through P2P connections. However, when no prefetching is used, the responses have to be sent through direct connections. Around a hotspot, many peers simultaneously may receive a query and try to connect to the querying peer. This situation can be overcome by limiting the number of receivers for a topic. Another approach might be not responding to all queries. The queries to be responded may be selected randomly or according to another criterion based on load or time.

Another point for improvement might be the AOI management. As described, the AOI messages are delivered by P2P ALM. By the time a cell enter message delivered to an interested peer, the moving peer might have traveled a considerable distance and become visible to the viewer by popping in front of him suddenly. Although the aspects are different, this problem is similar to the popping of the 3D content without the existence of a prefetching method. Same prefetching principles can be applied to AOI management. A peer can publish cell enter messages before they actually enter the cell. The strategy for a pre-publishing scheme will depend on the movement direction and statistical movement tendency of the peers as in the prefetching strategies. With pre-publishing, the connection will be established but update messages will not be sent until the peer actually enters the cell. The only overhead introduced by pre-publishing will be limited to the connections that are made before actual need and are abandoned without any actual use.

REFERENCES

[1] L. Ricci and E. Carlini, “Distributed Virtual Environments: From client server to cloud and P2P architectures,” *2012 Int. Conf. High Perform. Comput. Simul.*, pp. 8–17, Jul. 2012.

[2] A. Yahyavi and B. Kemme, “Peer-to-peer architectures for massively multiplayer online games: A survey,” *ACM Comput. Surv.*, vol. 46, no. 1, 2013.

[3] X. Jiang and F. Safaei, “Churn performance study of structured peer-to-peer overlay in supporting massively multiplayer online role playing games (MMORPGs),” *2011 17th IEEE Int. Conf. Networks*, pp. 183–188, Dec. 2011.

[4] C. Carter, A. El Rhalibi, and M. Merabti, “A Survey of AoIM, Distribution and Communication in Peer-To-Peer Online Games,” *2012 21st Int. Conf. Comput. Commun. Networks*, pp. 1–5, Jul. 2012.

[5] B. Ng, A. Si, R. W. H. Lau, and F. W. B. Li, “A multi-server architecture for distributed virtual walkthrough,” *Proc. ACM Symp. Virtual Real. Softw. Technol. - VRST ’02*, p. 163, 2002.

[6] J. Gilmore and H. Engelbrecht, “A survey of state persistency in peer-to-peer massively multiplayer online games,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 5, pp. 818–834, 2012.

[7] E. Teler and D. Lischinski, “Streaming of complex 3D scenes for remote walkthroughs,” *Comput. Graph. Forum*, vol. 20, no. 3, p. C/17-C/25, 2001.

[8] S. Y. Hu, T. H. Huang, S. C. Chang, W. L. Sung, J. R. Jiang, and B. Y. Chen, “FLoD: A framework for peer-to-peer 3D streaming,” *Proc. - IEEE INFOCOM*, pp. 2047–2055, 2008.

[9] S. Hu, J. Chen, and T. Chen, “VON: a scalable peer-to-peer network for virtual environments,” *Network, IEEE*, no. August, pp. 22–31, 2006.

[10] A. Valadares, E. Gabrielova, and C. V. Lopes, “On Designing and Testing Distributed Virtual Environments,” 2015.

[11] H. Hu, R. W. H. Lau, H. Hu, and B. Wah, “On view consistency in multi-server distributed virtual environments,” *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 10, pp. 1428–1440, 2014.

[12] Ş. B. Çevikbaş, G. Koldaş, and V. Işler, “Prefetching optimization for distributed urban environments,” in *Proceedings of the 2008 International Conference on Cyberworlds, CW 2008*, 2008, pp. 291–297.

[13] M. Moraal, “Massive Multiplayer Online Game Architectures,” Radboud University of Nijmegen, 2006.

[14] S. A. Abdulazeez, A. El Rhalibi, M. Merabti, and D. Al-Jumeily, “Survey of solutions for Peer-to-Peer MMOGs,” in *2015 International Conference on Computing, Networking and Communications, ICNC 2015*, 2015, pp. 1106–1110.

[15] L. Fan, P. Trinder, and H. Taylor, “Design issues for peer-to-peer massively multiplayer online games,” *Int. J. Adv. Media Commun.*, 2010.

[16] M. Almashor, I. Khalil, Z. Tari, and A. Y. Zomaya, “Automatic and Autonomous Load Management in Peer-to-Peer Virtual Environments,” *IEEE*

J. Sel. Areas Commun., vol. 31, no. 9, pp. 310–324, Sep. 2013.

[17] E. Buyukkaya, M. Abdallah, and G. Simon, "A survey of peer-to-peer overlay approaches for networked virtual environments," *Peer-to-Peer Netw. Appl.*, Sep. 2013.

[18] L. Fan, P. Trinder, H. Taylor, and A. I. Management, "Design Issues for Peer-to-Peer Massively Multiplayer Online Games," in *The 2nd International Workshop on Massively Multiuser Virtual Environments at IEEE Virtual Reality 2009*, 2009, pp. 1–11.

[19] W. Cai *et al.*, "A survey on cloud gaming: Future of computer games," *IEEE Access*, vol. 4, pp. 7605–7620, 2016.

[20] B. Knutsson, H. L. H. Lu, W. X. W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," *Ieee Infocom 2004*, vol. 1, no. C, 2004.

[21] B. Shen, J. Guo, and P. Chen, "A Survey of P2P Virtual World Infrastructure," *2012 IEEE Ninth Int. Conf. E-bus. Eng.*, pp. 296–303, Sep. 2012.

[22] S. Y. Hu, J. R. Jiang, and B. Y. Chen, "Peer-to-peer 3D streaming," *IEEE Internet Comput.*, vol. 14, no. 2, pp. 54–61, 2010.

[23] M. Albano, L. Ricci, and L. Genovali, "Hierarchical p2p overlays for DVE: An Additively Weighted Voronoi based approach," *2009 Int. Conf. Ultra Mod. Telecommun. Work.*, pp. 1–8, 2009.

[24] M. Ghaffari and B. Hariri, "A Dynamic Networking Substrate for Distributed MMOGs," *IEEE Trans. Emerg. Top. Comput.*, vol. 3, no. 2, 2015.

[25] A. Steed and C. Angus, "Frontier sets: A partitioning scheme to enable scalable virtual environments," *Proc. EUROGRAPHICS 2004*, pp. 1–4, 2004.

[26] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand, "A survey of visibility for walkthrough applications," *IEEE Trans. Vis. Comput. Graph.*, vol. 9, no. 3, pp. 412–431, Jul. 2003.

[27] B. Knutsson and H. Lu, "Peer-to-Peer Support for Massively Multiplayer Games," *INFOCOM 2004*, pp. 96–107, 2004.

[28] A. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," *Middlew. 2001*, vol. 2218, no. November 2001, pp. 329–350, 2001.

[29] A. Rowstron and A. Kermarrec, "SCRIBE: The design of a large-scale event notification infrastructure," *Proc. Third Int. COST264 Work. Networked Gr. Commun.*, no. November 2001, 2001.

[30] S. Han, M. Lim, D. Lee, and S. J. Hyun, "A scalable interest management scheme for distributed virtual environments," *Comput. Animat. Virtual Worlds*, vol. 19, no. 2, pp. 129–149, 2008.

[31] S. Rueda, P. Morillo, and J. M. Ordu??a, "A comparative study of awareness methods for peer-to-peer distributed virtual environments," *Comput. Animat. Virtual Worlds*, vol. 19, no. 5, pp. 537–552, 2008.

[32] G. Popescu, C. Codella, P. O. Box, and Y. Heights, "Scalable and Efficient Update Dissemination Georgia Institute of Technology," in *ICDCS '02 Proceedings of the 22 nd International Conference on Distributed Computing Systems*, 2002, p. 143.

[33] J. Jia, M. Wang, W. Wang, and X. Hei, "On Prioritization Mechanisms for Large-Scale 3D Streaming in Distributed Virtual Environments," in *2016*

International Conference on Virtual Reality and Visualization (ICVRV), 2016, pp. 465–472.

[34] R. Diaconu and J. Keller, "Kiwano: Scaling virtual worlds," in *Proceedings - Winter Simulation Conference*, 2017, pp. 1836–1847.

[35] E. S. Liu and A. Rungta, "A Quantitative Analysis of Local Data Management for 3D Content Streaming," in *Proceedings of the 2016 Winter Simulation Conference*, 2016, no. Ldm, pp. 3155–3166.

[36] S. A. Abdulazeez, A. El Rhalibi, and D. Al-jumeily, "Evaluation of Scalability and Communication in MMOGs," in *13th IEEE Annual Consumer Communications & Networking Conference (CCNC) Evaluation*, 2016.

[37] S. Y. Hu and K. T. Chen, "VSO: Self-organizing spatial publish subscribe," *Proc. - 2011 5th IEEE Int. Conf. Self-Adaptive Self-Organizing Syst. SASO 2011*, pp. 21–30, 2011.

[38] E. Lua, J. Crowcroft, and M. Pias, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Commun. Surv. Tutorials*, pp. 72–93.

[39] S. Holzapfel, A. Wacker, T. Weis, and M. Wander, "An architecture for complex P2P systems," *2012 IEEE Consum. Commun. Netw. Conf.*, pp. 507–511, Jan. 2012.

[40] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *Conf. Appl. Technol. Archit. Protoc. Comput. Commun. (SIGCOMM '01)*, pp. 149–160, 2001.

[41] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *IEEE Commun. Surv. Tutorials*, vol. 9, no. 3, pp. 58–74, 2007.

[42] G. Koldas, V. Isler, and R. Lau, "Six Degrees of Freedom Incremental Occlusion Horizon Culling Method for Urban Environments," *Adv. Vis. Comput.*, pp. 792–803, 2007.

[43] P. Wonka, M. Wimmer, and D. Schmalstieg, "Visibility preprocessing with occluder fusion for urban walkthroughs," *Render. Tech. 2000*, pp. 71–82, 2000.

[44] L. Downs, T. Möller, and C. H. Séquin, "Occlusion horizons for driving through urban scenery," *Proc. 2001 Symp. Interact. 3D Graph. - SI3D '01*, pp. 121–124, 2001.

[45] Z. Zhou, K. Chen, and J. Zhang, "Efficient 3-D scene prefetching from learning user access patterns," *IEEE Trans. Multimed.*, vol. 17, no. 7, pp. 1081–1095, 2015.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name	: Çevikbaş, Şafak Burak
Nationality	: Turkish (TC)
Date and Place of Birth	: 4 July 1982, Ankara
Marital Status	: Married
Phone	: +90 505 436 96 14
Email	: safakburak@gmail.com

EDUCATION

Degree	**Institution**	**Year of Graduation**
MS	METU Computer Engineering	2008
BS	METU Computer Engineering	2004
High School	Gazi Anatolian High School, Ankara	2000

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2015-Present	T2 Yazılım	Senior
2013-2015	ATOS	Hands-on Project Manager
2010-2013	Simsoft A.Ş.	Hands-on Project Manager
2004-1010	Milsoft A.Ş.	Software Engineer

FOREIGN LANGUAGES

Advanced English, Preliminary German, Preliminary Spanish

PUBLICATIONS

1. JNSE 15, “Survey on Visibility and Data Distribution in Distributed Virtual Environments”, 2015, Yekta Kılıç, Gürkan Koldaş, Şafak Burak Çevikbaş
2. Thesis, “Visibility Based Prefetching with Simulated Annealing”, 2008, Şafak Burak Çevikbaş
3. CGW 08, “Prefetching Optimization for Distributed Urban Environments”, 2008, Şafak Burak Çevikbaş, Veysi İşler, Gürkan Koldaş
4. SAVTEK 06, “Dağıtık Simülasyon Uygulamalarında Gerçek Zamanlı Görselleştirme, Ufuk Tabanlı Görünürlük Tespit Yöntemi”, 2006, Şafak Burak Çevikbaş, Veysi İşler, Gürkan Koldaş
5. TEHOSS 06, “Echo Generation for Mobile Radar”, 2006, Şafak Burak Çevikbaş, Veysi İşler, Gürkan Koldaş