# A Review on Networking and Multiplayer Computer Games

**Jouni Smed**

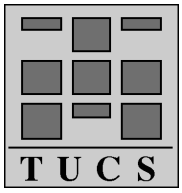
University of Turku, Department of Computer Science,
Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland

**Timo Kaukoranta**

University of Turku, Department of Computer Science,
Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland

**Harri Hakonen**

Oy L M Ericsson Ab, Telecom R&D,
Joukahaisenkatu 1, FIN-20520 Turku, Finland




**Turku Centre for Computer Science**
**TUCS Technical Report No 454**
**April 2002**
**ISBN 952-12-0983-6**
**ISSN 1239-1891**

**Abstract**

Networking forms an essential part of multiplayer computer games. In this paper, we review the techniques developed for improving networking in distributed interactive real-time applications. We present a survey of the relevant literature concentrating on the research done on military simulations, networked virtual environments, and multiplayer computer games. We also discuss on resource management, consistency and responsiveness, and networking on the application level.

**Keywords:** computer games, networked virtual environments, online entertainment, distributed interactive simulation

**TUCS Research Group**
Algorithmics laboratory

# 1 Introduction

Over the past twenty years three distinct classes of distributed interactive real-time applications have become prominent: (1) *military simulations* [62], (2) *networked virtual environments* (NVEs) [56], and (3) *multiplayer computer games* (MCGs) [59]. The focus of scientific research has shifted from the military simulations (the 1980s) through NVEs (the 1990s) currently to MCGs (see Figure 1). Moreover, the entertainment industry is investing seriously on MCGs, mobile gaming and online gaming in general.

The terminology encountered in the literature is diverse. For example, until recently NVEs were usually called distributed virtual environments (DVEs), which then gave way to collaborative virtual environments (CVEs) [42], [8]. We have adopted the term NVE, since it encompasses both DVE and CVE. Military terminology prefers the word 'simulation', because they can be more than NVEs (e.g., logistical simulations). The relationship between games and simulations is not straightforward. Granted, there are games that are simulations (e.g., football manager games) and games that are especially happening in a VE (e.g., flight simulators or first person shooters). However, as the games get more abstracted, they are less and less simulations (see Figure 2).

Networking is in dominating position when we consider the playability of a MCG. The *physical platform* induces resource limitations (e.g., bandwidth and latency) that reflect the underlying infrastructure (e.g., cabling and hardware). Normally, there is not much we can do the physical platform—except perhaps invest on new hardware. The *logical platform* builds upon the physical platform, and the choices made in the logical platform play a pivotal role in the design of a MCG. It provides architecture for communication, data, and control. Communication architecture defines the logical connections between the nodes in a network. For example, in peer-to-peer architecture a set of equal nodes are interconnected, whereas in client/server architecture one node acts as a server and all communication between nodes is handled through it. Data and control architecture defines, how information is stored and updated in the nodes. For example, in a centralized architecture one node holds the data, whereas in a replicated architecture each node has its own replica.

This paper concentrates on the features of logical platform. We cast a look back and present a review of the research work done in the past twenty years. This paper tries to sum up the story so far in the sense that it includes both a tutorial to techniques in Section 2 and a survey of literature in Section 3. Also, we discuss resource management, consistency and responsiveness, and networking on the application level in Section 4. The concluding remarks

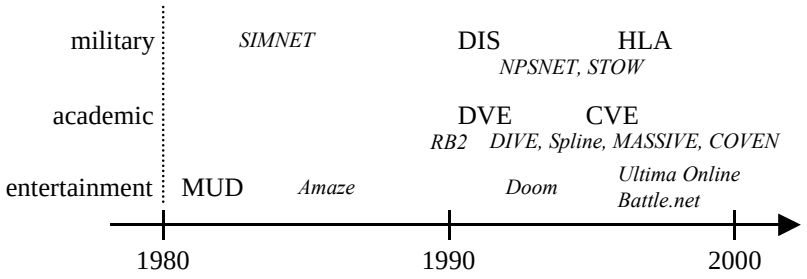



Figure 1: History of distributed interactive real-time applications.

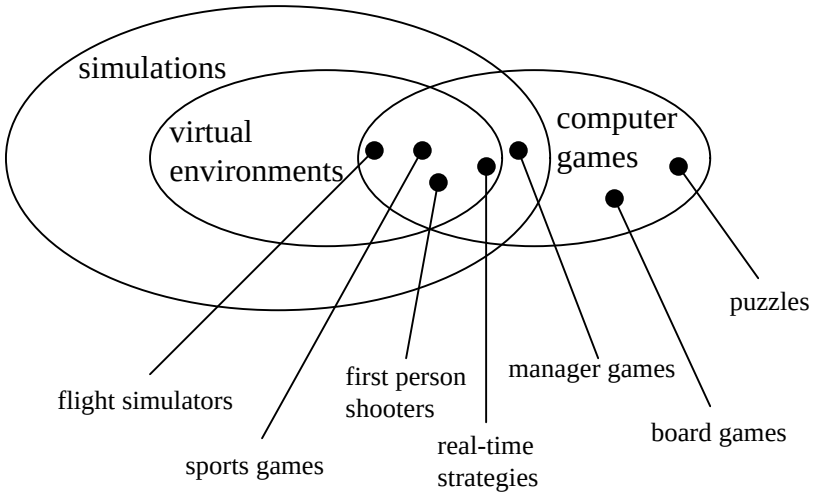



Figure 2: Relationship of simulations, virtual environments (VEs) and computer games. While VEs simulate (possibly real-world) environments, computer games do not necessarily belong to simulations or VEs.

appear in Section 5.

# 2 Techniques

Let us first reiterate the most common techniques to reduce bandwidth requirements of a distributed interactive real-time application (for more details, see [56]).

## 2.1 Packet Compression and Aggregation

The purpose of compression is to reduce the number of bits needed to represent particular information. Thus, the compression of network packets offers an intuitive approach to minimize network traffic. Compression techniques can be classified according to their ability to preserve information content. *Lossless techniques* preserve all information, and, therefore, the reconstructed data is exactly the same as the data before compression. As a rule of thumb, lossless compression techniques can shrink the size of data approximately down to half. To achieve a higher compression ratio, *lossy* compression techniques can be employed. The idea is to leave out less relevant information so that the distortion in the reconstructed data remains unnoticeable. This is a widely used technique, for example, in audio and image compression.

Because we are compressing network packets, it is also worth noticing how different compression techniques relate to data in packet format. *Internal compression* concentrates on the information content of one packet without references to other, previously transmitted packets. Therefore, it suits to the cases where unreliable network transmission protocols such as User Datagram Protocol (UDP) are used. On the other hand, *external compression* may utilize information that has been already transmitted and, therefore, can be assumed to be available to receivers. For example, we can transmit delta or transition information which is likely to require less bits than the absolute information. We can also give reference pointers to previously transmitted data if the same data occurs again. External compression can consider a large amount of data at a time, and, thus, it can better observe redundancy in the information flow. Consequently, it allows better compression rations than internal compression. However, because of the references to the previous packets, external compression requires a reliable transmission protocol.

*Packet aggregation* reduces bandwidth requirements by merging several packets and transmitting their content in one larger packet. Thus, the overhead caused by packet headers is smaller. Bandwidth savings can be considerable depending on the size of data in the original packets, the size of

packet headers, and the number of merged packets. For example, UDP/IP and TCP/IP packet headers take 28 and 40 bytes, respectively.

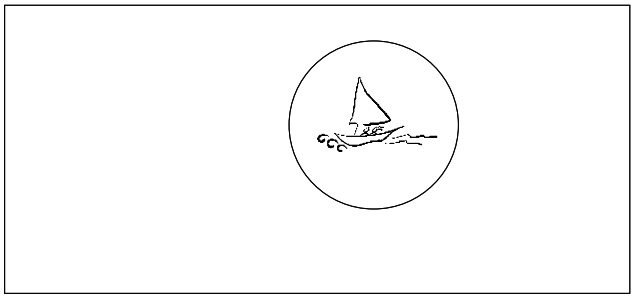
There are two basic approach to determine the number of merged packets: timeout-based approach and quorum-based approach. In *timeout-based approach*, all packets that are initiated before a fixed time period are merged. This approach guarantees an upper bound on the delay caused by aggregation. Now, bandwidth savings depend on the packet initiation rate, and, in the worst case, no savings are gained because no packets, or only one, are initiated during the period. In the *quorum-based approach*, a fixed number of packets is always merged. Because the transmission of the merged packet is delayed until enough packets are initiated, there is no guarantee for the transmission delay. Although bandwidth savings are predictable, long transmission delays can hinder the user's experience. The limitations of both approaches can be compensated by combining them. In this hybrid approach, packets are merged whenever one of the conditions fulfills, either time period expires or there are enough packets to merge.

## 2.2 Interest Management

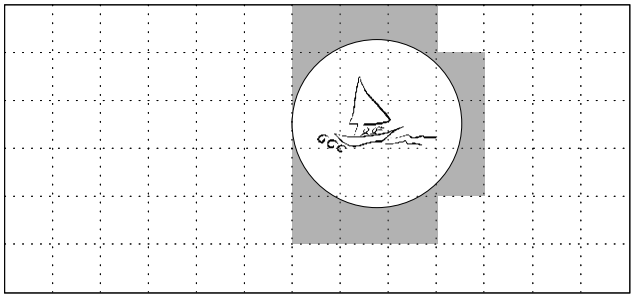
The entities usually produce update packets that are relevant only a minority of the nodes. Therefore, an obvious way to save bandwidth is to disseminate update packets only to those nodes who are interested in them. This *interest management* includes techniques that allow the nodes to express interest in only the subset of information that is relevant to them [8], [46]. An expression of data interest is called the *aura* or the *area of interest*, and it usually correlates with the sensing capabilities of the system being modeled (see Figure 3). Simply put, an aura is a subspace where interaction occurs. Thus, when two players' auras intersect, they can be aware of each others actions.

Interest management with auras is always symmetric: If the auras intersect, both parties receive messages from each other. However, aura can be divided further into a *focus* and a *nimbus*, which represent observer's perception and observed object's perceptivity [7], [31]. Thus, awareness requires that the player's focus intersects with another player's nimbus. By using foci and nimbi it possible to construct a finer-grade message filtering, since awareness needs not to be symmetric (see Figure 4). Auras, foci, and nimbi can be modified by *adapters* in order to customize player's interaction. For example, the VE can offer infrared binoculars and camouflage tools.

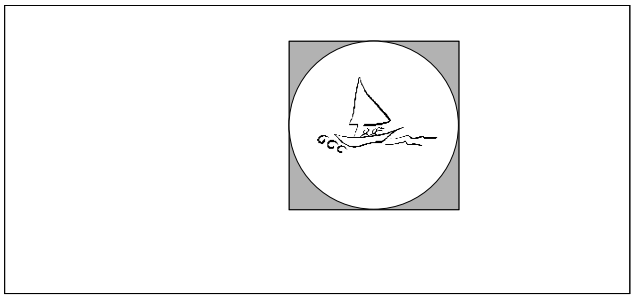
In a *area-of-interest filtering* scheme, the nodes transmit their state changes to *subscription managers*. The managers also receive subscriptions that express nodes' information interests (or foci). The manager then transmits



(a) By using formulae the aura can be expressed precisely, like the circle around the sailboat which indicates the observable range. However, the implementation can be complex and the required computation hard.



(b) The space can be divided into static, discrete cells. The sailboat is interested in the cells that intersect its aura. Cell-based filtering is easier to implement but it is less discriminating than formula-based. The cell grid can also be hexagonal.



(c) Extents approximate the actual aura with rectangles (i.e., it is a bounding box). The computation is simpler than by using formulae and the filtering better than by using cells.

Figure 3: Auras (or areas of interest) can be expressed using formulae, cells or extents.

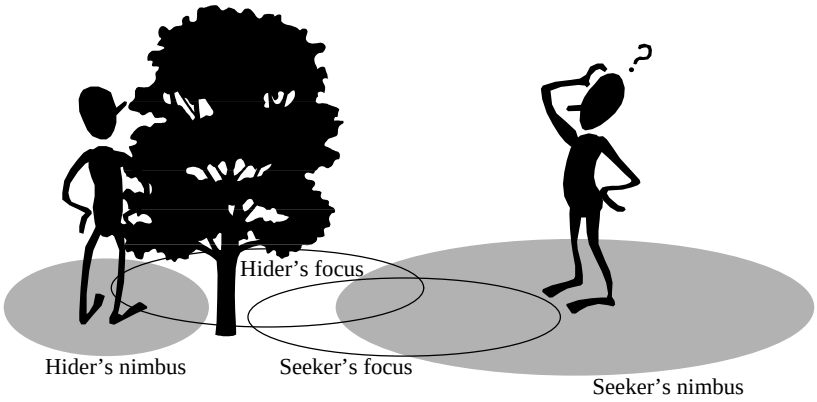



Figure 4: In hide-and-seek, the nimbus of the hiding person is smaller than the seeker's, and the seeker is not aware of the hider. Instead, the hider can observe the seeker, since the seeker's nimbus is larger and intersects the hider's focus.

to the node only the relevant information (i.e. that matches to the node's subscription). Area-of-interest filters can be called *intrinsic filters* because they use application specific data content of an update packet to determine which nodes need to receive it. This filtering provides fine-grained information delivery but packet processing may require a considerable amount of time. *Extrinsic filters* determine the receivers of a packet merely based on its network attributes (e.g., address). Extrinsic filters are faster to process than intrinsic filters, and even the network itself can provide them.

*Multicasting* is a network protocol technique that realizes this approach [21]. In multicasting, an application transmits packets to a *multicast group* identified by a *multicast address*. To receive packets from the multicast group, the node has to subscribe (or join) it. Multicasting is comparatively efficient network dissemination protocol. The challenge in the design of a multicast-based application is how to categorize all transmitted information into multicast groups. Each packet sent to a multicast group should be relevant to all subscribers. Although by using several multicast groups we can achieve a fine-grained information supply, the maintenance can be problematical (e.g., their addressed can collide with other applications in the Internet or network adapters can support a limited number of multicast subscriptions).

In *group-per-object allocation* strategy, each object has its own multicast group to which the object transmits its updates. Assigned servers keep a record of the multicast addresses so that the nodes can subscribe to the relevant groups. By using several multicast groups, one object can provide more fine-grained update dissemination and the subscribers can better approxi-

mate their foci.

In *group-per-region allocation* strategy, the virtual environment is divided into regions that have their own multicast groups. All objects within the region transmit updates to the corresponding multicast address. Typically objects subscribe to groups corresponding to their own region and the neighboring regions. As an object crosses a region boundary it has to receive information about currently relevant multicast groups from *directory servers*. Instead of categorizing objects based on their location in the VE, other attributes are possible (e.g., object's type or team).

## 2.3 Dead Reckoning

One approach to reduce bandwidth use is to send update packets less frequently. To maintain consistency (or at least a reasonable semblance) we have to somehow compensate the lack of information between the packet updates. Especially in the case of position information, *dead reckoning* [57] methods have proved to be successful. In dead reckoning, the missing information is computed with an approximation technique. Based on previously received information, the node predicts movement of a particular object. The predicted state of the object is used in the application until new information is received from the source node. Depending on the accuracy of the prediction technique, the approximated location can be some distance from its actual location. To avoid jerky movements when new location information is applied a convergence algorithm is used to smooth the transfer. Thereby, dead reckoning consists of two parts: a *prediction technique* and a *convergence technique*.

The most common prediction technique is to use derivative polynomials. In the case of positions, their natural interpretations are velocity, acceleration, and jerk. If we use zero-order derivative polynomials, only position information is transmitted and we achieve no gain. In the case of first-order derivative polynomials, the velocity of an object is transmitted in addition to the position. Velocity improves prediction accuracy noticeablely. To improve accuracy further, we can add acceleration to the transmitted terms. This second-order polynomial prediction is the most popular technique. However, higher derivatives increase the risk of inaccurate prediction: Because the prediction is more sensitive for high-order terms, a small inaccuracy in them may result in significant errors. Also, high-order derivative polynomials increase the computational burden of the prediction. Additional terms consume also our limited bandwidth resources.

To balance between bandwidth requirements and the risk of inaccurate predicition, hybrid systems can dynamically select either first-order or second-

order prediction. For example, if the entity's acceleration changes often, it is probable that wrong value is applied to the prediction at some point. Therefore, it might be safer to content with first-order prediction at that time.

Instead of transmitting higher polynomial terms, they can be approximated in the receiving node. The Position History-Based Dead Reckoning (PHBDR) [58] protocol transmits only the absolute positions, and object's instantaneous velocity and acceleration are approximated by using the most recent position updates. Also, the source node can apply the same prediction technique as the destination node. When the source node determines that the distance between the real state and the predicted state exceeds a given threshold, the source transmits an update packet. The threshold can be dynamically changed according to the distance between the objects [17]. The idea is that the farther the object is, the less frequent updates are needed. Also, update lifetime can be considered (i.e., the time interval between two consecutive state updates) [70]. The rationale is that a specified level can be set for update lifetime, which limits the rate of sent messages.

When a node receives an update message, the object's predicted position is likely to differ from the position based on the latest information. The object needs to be moved to this new position, and convergence technique defines how this correction is performed. A good convergence technique corrects error quickly and unnoticeably (see Figure 5). The simplest technique is zero-order convergence where the object is just moved to its new predicted position. However, this can cause annoying jerky movement. A better method is to use linear convergence where a future *convergence point* is determined from the new prediction path. Then the object is moved along direct path from the current position to a convergence point. during *convergence period*.

Although linear convergence is clearly better than zero-order convergence, it can still make unnatural turns when leaving the previous predicted path and at entering to the new predicted path. To smooth out these problems, more sophisticated curve-fitting techniques can be applied. The idea is to select, in addition to the current position and convergence point, a number of points along the previous predicted path and the new predicted path. The curve is fitted to go through all the selected points, and it is used as a path to move the entity to its new predicted path. For example, in the case of a third-order curve, cubic spline, we pick one additional point on the previous path before the current position and other additional point along the new predicted path after the convergence point. High-order curves provide smooth transition from the old path to the new path but they can computationally intensive for some applications.

By including object specific information to the dead reckoning technique we can achieve more accurate and natural movement. Therefore, it can be

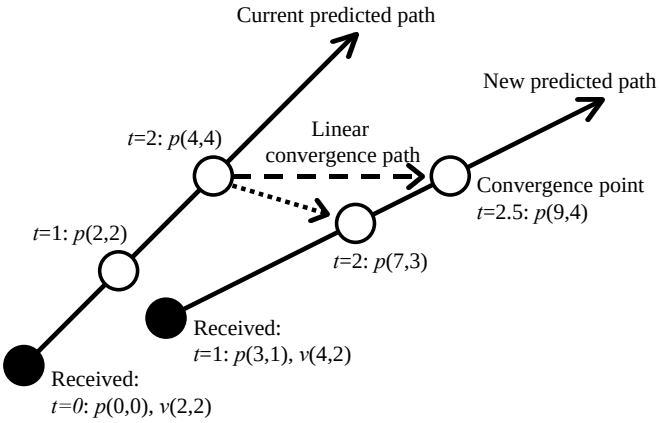



Figure 5: Illustration of dead reckoning. Closed circles represent received information about object's position $p$ and velocity $v$ at given time $t$. Open circles represent object's predicted position at given time. At time $t = 2$ when object's position is predicted to be $(4, 4)$, new information is received that at time $t = 1$ (because of the latency) object's actual position was $(3, 1)$ and velocity was $(4, 2)$. Instead of moving object to its new predicted position $(7, 3)$ immediately, a convergence point is calculated from the new predicted path at 0.5 seconds later. During this convergence period the object is rendered smoothly along the linear converge path to the new predicted path, which it follows later on.

usuful to design specialized dead reckoning techniques for each object type. However, this can be time consuming and maintenance of several algorithms requires special care. Dead reckoning can introduce also other side effects that need to be considered carefully. For example, because all nodes need not to share same view to the entities' states, collision detection algorithms can be difficult to desing.

# 3 Related Work

We have divided the literature on distributed interactive real-time applications into three subsections according to the classification given in the introduction.

## 3.1 Military Simulations

The United States Department of Defense has been developing networked military simulations since the 1980s. The first developed protocol was *SIM-*

*NET*, which intends to provide interactive networking for real-time, human-in-the-loop battle engagement simulation and war-gaming [1]. To achieve this SIMNET aims at providing functional fidelity rather than accurate physical reproduction. Networking utilizes a distributed architecture with no central server, which allows that the participants can join and leave the simulation at any time. The objects interact by broadcasting events to the network, and the receiver is then responsible for calculating the effects (and everybody is expected to uphold fair play). Between the updates object's position information is extrapolated by dead reckoning.

*Distributed Interactive Simulation* (DIS), which was issued as IEEE Standard 1278 in 1992, attempts to formally generalize and extend the SIMNET protocol [41], [47]. The purpose is to allow any type of player on any type of machine to participate the simulation. DIS allows to model different kind of vehicles (e.g., airplanes or battleships), and, consequently, it is used in many specialized systems such as NPSNET [43] or STOW [22].

The current military research efforts concentrate on developing systems based on *High Level Architecture* (HLA), which was issued as IEEE Standard 1516 in 2000 [63]. HLA aims at providing a general architecture and services for distributed data exchange. It does not prescribe any specific implementation or technology. While the DIS protocol is closely linked with the properties of military units and vehicles, the rationale behind HLA is that it could be used with different types of simulations—even in non-military applications—and it is targeted towards new simulation developments.

## 3.2 Networked Virtual Environments

While the military research focuses on diverse large-scale systems, NVEs are mainly designed for local use and to support only a small number of participants. Usually, NVEs also pay closer attention to the virtual representations of the participants (i.e., avatars) and the collaboration between the participants (e.g., operating at the same time with a shared object).

One of the first NVEs is *Reality Built for Two* (RB2) [12]. It uses a central server to manage devices and distribute their data, and additional machines for rendering the data. As the name suggests, it scales up to allow two users to share the same VE.

*MR Toolkit* [53], [52], [66] divides the VE into four components: presentation, interaction, geometric model, and computation. These components can be distributed among the nodes in a network. The master processes of different application instances can communicate with each other which allows multiple users to share the same VE.

*BrickNet* [55] system uses a client/server architecture. Each client connects to a server to request objects of interest and to communicate with other clients. Client can deposit its own objects to the server and thus share them with other clients. The clients run asynchronously, and the server ensures that update messages are sent only to those clients who have subscribed the object in question. Moreover, the subscribers can select the level of consistency ranging from absolute to loose, which affects also the update delays.

*RING* [28], [29] is a client/server system where each object in the VE is managed by exactly one client. The VE is subdivided spatially into cells, and the other clients can have a replica of the object if they reside in the same cell. This filtering is based on precomputed line-of-sight visibility information, and it is carried out by servers, which can alter the client communication.

*DIVE* [27] uses a replicated world database and peer-to-peer communication. When an object is updated, it is done in the local replica and a message is sent to all peers to update their own replicas accordingly. Naturally, this subjects the replicated object to inconsistencies due to network delays. DIVE compensates them by employing dead reckoning and by sending periodically synchronization information. To reduce network traffic DIVE allows to divide the VE into subhierarchies (e.g., based on geography) that are replicated among a smaller set of nodes that have expressed interest in them. The subhierarchies can be associated with multicast groups, which further reduces the traffic.

*MASSIVE* [31], [32] supports different computer platforms and allows the users to interact with each other over a combination of graphics, audio and text media. The system uses awareness-based filtering, where each entity expresses a focus and nimbus for each medium (e.g., the focus for textual medium can be smaller than the focus for graphical medium). MASSIVE uses a client/server architecture and multicasting. The server provides the clients with an initial point of access to the VE. Entities are replicated on demand and they are associated with multicast groups.

The efforts of DIVE and MASSIVE systems are now combined and coordinated in the *COVEN* project [48]

In *Spline* [4], [67] the VE is divided into locales, which have arbitrary size and shape and are associated with multicast addresses. Spline uses a distributed architecture where each node maintains a partial copy of the VE corresponding to the focus of attention. The user can see several locales, but each object is at most in one locale at a time.

In *GreenSpace* [44], [51] each entity use multicasting to transmit its state, and a lightweight server assigns multicast addresses and informs the entities of changes in a VE. The system uses Virtual Reality Modeling Language (VRML) for rendering the VE.

In *Community Place* [37], [38] each entity sends position information to a server. The server uses spatial filtering based on auras to decide which other entities need to be aware of these position changes. The server also distributes updates to local scenes and events. The static elements of a scene are loaded as VRML, while dynamic data is managed through local scripts and message passing. If the server becomes a bottleneck, it can be replicated.

*AGORA* [33] is a VRML-based system that uses a client/server architecture and a shared, centralized database. The server distributes the update messages by adding them into either a sequential (i.e., order preserving) or non-sequential (i.e., possibly unreliable) transmission queue according to their type.

*SmallTool* [16], [15] uses VRML and a distributed worlds transfer and communication protocol (DWTP), which provides daemons for transmitting VE contents, detecting transmission failures and recovering lost packages. In addition to being replicated, each object can specify whether a particular event requires a reliable distribution and what is the event's maximum update frequency.

*Living Worlds* [68] is a VRML-based system which allows anonymous interactions between loosely coupled parties. The VE is divided into zones which are associated with a server. Each object is owned by one client, and the server nominates a new owner if the previous one leaves the zone. The system is subjected to unpredictable delays, which can be analyzed by using Extended User Action Notation (XUAN) [24]. To overcome the problems, the systems requires better level of detail control, world partitioning, and a communication protocol supporting QoS (quality of service) so that when the number of participant grow, the amount information remains relatively constant.

*DVECOM* [20] is a centralized system which guarantees both synchronization and consistency of the displays. The system provides QoS by degrading the rendering of the VE, if the client's resources are lacking.

*Urbi et Orbi* [64], [25], [26] includes a scripting language for giving a high-level description of the objects in a VE. The object description comprises also object's behavior and distribution policy. If an object is deterministic, it is replicated on each node in the network, and each node is responsible for updating the local replica. An indeterministic object is distributed to one node from where it begins to multicast update messages to the network.

In *PaRADE* [50] system replicated databases are maintained through the communication of non-deterministic events and local calculation of deterministic events. Events can be discrete (e.g., a state change) or continuous (e.g., an audio stream). Discrete events require that the before-after relations are preserved, whereas continuous events are stamped with a wallclock time that

is kept synchronized in each node.

*CIAO* [61] uses an optimistic method for concurrency control for replicated objects. An update is carried out immediately in the local replica and transmitted to the remote nodes. If a conflicting operation occurs, a token associated with the manipulated object is used to maintain consistency. Initially, object's creator has the token. When the owner of the token receives the update message, it validates the operation by giving the token to the node which commenced the update.

*Real-Time Transport Protocol* (RTP/I) [45], [65] is aimed at distributed interactive media. It includes three methods for ensuring that all application instances look as if all operations have been executed in the same order. Inconsistencies caused by operation delays (e.g., network latency) are handled by delaying deliberately the local updates to match the transmission delays. Each node keeps a history, and if an inconsistency occurs, the situation is rolled back, the conflicting operation is carried out, and situation is rolled forward back to the current time. As a last resort, the protocol includes also a method for explicit state request.

*Synchronous Collaboration Transport Protocol* (SCTP) [54] focuses on collaboration on closely coupled, highly synchronized tasks. An interaction creates a stream of update messages, where the most important message is the last one. In the protocol, one message sequence related to user's interaction with a shared object is grouped into one stream. This interaction stream has critical messages (especially the last one) which are sent reliably, while the are sent by best effort transport.

## 3.3 Multiplayer Computer Games

Until recently the problems of MCGs and online gaming have been dealt marginally in the scientific literature. Usually the papers concentrate on simple games and limited problem settings.

*Multi-User Dungeons* (MUDs) [5] are text-based MCGs, that began turn of the 1980s. The players have access to a shared database comprising, for example, rooms, exits, and artifacts. The players are inside a room where they can browse and manipulate the database. They can move between the rooms through the exits that connect them. Users can add new rooms and other objects to the database and give them unique behavior by using an embedded programming language. Users can also communicate directly with each other in real time.

In *Amaze* [9] the players are in a 2D maze and their goal is shoot missiles to other players. Each node uses point-to-point communication to transmit once a second position and velocity updates (thus allowing dead reckoning).

Players can join and leave at any time, and the system supports computer-controlled players.

*XPilot* [60] is a 2D dogfight game which uses a simple client/server architecture. Because it does not employ dead reckoning, the responsiveness is effectively determined by the network latency.

*Artery* [18], [19] system provides a programming interface for MCGs. Networking is based on DIS protocol, and the system tries to reduce the traffic by utilizing application-specific semantic knowledge. The system also supports dead reckoning, message aggregation, and interest management.

In *MiMaze* [39], [30], [23] the players try shoot each other in a 3D maze. It uses a distributed architecture, and requires a server only for initialization. To cope with different transmission delays MiMaze employs a bucket synchronization mechanism. Delays between participating hosts are evaluated by using a wallclock time. A message issued at absolute time is delayed according the measured transmission delay so that all participants can evaluate it at the same time. If the message is missed or it arrives too late, dead reckoning is used to extrapolate the information.

*Distributed Entertainment Environment* (DEE) [49] is an architecture for distributed gaming. It divides the game world into a conceptual model (i.e., rules and object attributes), a dynamic model (i.e., interaction at the spatial level), and a visual model (i.e., rendering information). To reduce network traffic the conceptual and dynamic models are stored in a server, while each participating client have its own instance of the visual model.

A generic model for MCGs on demand is described in [3], [2]. The game data is stored in a server from where it is transferred locally to a CPU server for running a particular game session. The game session data is sent to the network where a front-end server acts as a proxy (e.g., stores level data). From there the data is conveyed to the client, where rendering and all computation intensive work is carried out.

User behavior in MCGs is analyzed and modelled in [14], [35], [34]. The data is drawn from real-world network traffic generated by first person shooter games in a client/server architecture.

Network security and cheating prevention in MCGs are addressed in [6], [69].

People working in the entertainment industry have recently started to publish more openly their ideas and solutions in the trade magazines and conferences [13], [40], [11], [10], [36].

# 4 Discussion

To build a more cohesive picture let us now discuss broader issues affecting networking in MCGs. First, we present how networking resources are interconnected and how the different techniques affect them. Next, we introduce a concept that enables us to understand the problems of achieving a consistent and responsive system. Finally, we consider the relationship between the logical platform and the application itself.

## 4.1 Resource Management

The amount of consumed resources of a networked application is directly related to how much information has to be send and received by each participating computer and how quickly it has to be delivered by the network. Singhal and Zyda have call this rule Networked Virtual Environment Information Principle [56]. They concretize it by giving *Information Principle Equation*:

$$\text{Resources} = M \times H \times B \times T \times P \tag{1}$$

where $M$ is the number of messages transmitted, $H$ the average number of destination nodes for each message, $B$ the average amount of network bandwidth required for a message to each destination, $T$ timeliness with which the network must deliver packets to each destination (large values of $T$ imply a need for minimal delay and vice versa), and $P$ the number of processor cycles required to receive and process each message.

A system designer can use Information Principle Equation as a tool to balance requirements and restrictions. By lowering any of the five variables the resource demands decrease. However, when we reduce the expenditure of one resource we have to compensate it somehow. This means that another variable in the equation increases or the quality of experience to be in the VE becomes lower. The choice of which variables are decreased and which variables are used for compensating depends naturally on the application's requirements and resource bottlenecks.

Let us recap how the techniques of Sect.2 correlate with the variables of the Information Principle Equation. In packet compression, we reduce the average packet size ($B$) but because of encoding and decoding processes the computational work ($P$) increases. Packet aggregation merges several packets to reduce bandwidth consumption caused by packet headers. Therefore, the number of packets ($M$) decrease and the average packet size ($B$) increases. Overall bandwidth consumption is reduced at slight processing cost ($P$). Interest management techniques pursue to reduce the average number

of messages ($M$) and bandwidth ($B$) per message. This requires more organizing between the nodes, and, consequently, more messages ($M$). Dead reckoning transmits update messages less frequently ($M$) but has to maintain predicted states for objects ($P$). In addition, the quality of VE experience is sometimes decreased due to inaccurate information.

## 4.2 Consistency and Responsiveness

*Consistency* refers to the similarity of the view to the data in the nodes belonging to a network. Absolute consistency means that each node has uniform information. *Responsiveness* means the delay that it takes for an update event to be registered in the database. Consistency and responsiveness are not independent from each other. Traditionally, responsiveness has always been subjected to consistency requirements in database research. However, because of real-time interaction, responsiveness becomes more important element in MCGs and consistency may have to be compromised.

To achieve high consistency, the data and control architecture must guarantee that processes running on remote nodes are tightly coupled. This usually requires high bandwidth, low latency, and a small number of remote nodes. To achieve high responsiveness, the queries made to the data must be responded quickly, which requires to loosely coupled nodes. In this case, the nodes must include more computation to reduce the bandwidth and latency requirements. In reality, a network architecture cannot achieve both high consistency and high responsiveness at the same time, and the choice of architecture is essentially a trade-off between these two attributes.

We can discern three parts in data and control architectures: the local node, the network, and the relay connecting them [59] (see Figure 6). The relay acts as an intermediary between the local node and the network, and its structure defines how consistent and how responsive the architecture can be.

The relay has two input and output pairs. The local input $i_{\text{local}}$ originated from the local node, and the local output $o_{\text{local}}$ is directed to the node. From the network's point of view, the relay sends ($o_{\text{global}}$) and receives messages ($i_{\text{global}}$). The communication architecture prescribes where these messages are transmitted (e.g., in client/server the messages are transmitted between the server and each client). For instance, an application running in a local node sends control messages (e.g., from keyboard or joystick) into a relay and receives data messages (e.g., vehicle positions) from it. In turn, the relay communicates with the relays of other nodes via a network.

Let us put aside the communication architecture and the operations inside the local node and concentrate on the relay itself. Obviously, the message

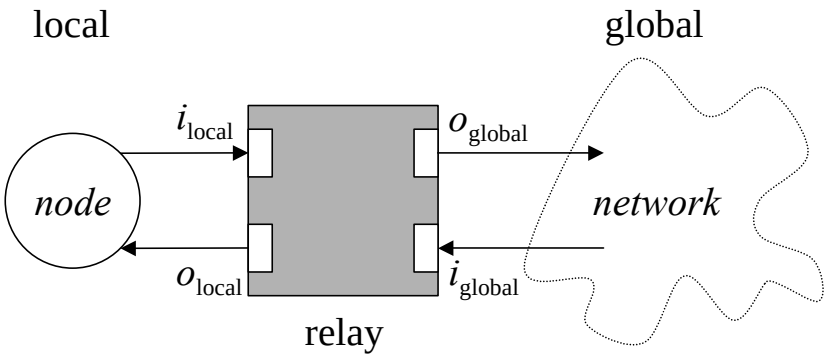



Figure 6: Data and control architecture defines how messages are relayed between local and remote nodes.

flows from $i_{\text{local}}$ to $o_{\text{global}}$ and from $i_{\text{global}}$ to $o_{\text{local}}$ must exist. This gives the minimum form, a *two-way relay* (see Figure 7a). The functions $f$ and $g$ denote that the message may undergo some operations (e.g., compression or time-stamping) inside the relay before it is passed on. The two-way relay is the model used, for instance, in distributed databases and centralized systems. All new local messages are relayed to the network, and they do not appear in the local node until a message from the network is received. In effect, the two-way relay acts as a simple intermediary between the node and the network. For example, a dumb terminal sends the characters typed on the keyboard to a mainframe, which sends back the characters to be displayed on the monitor.

It is easy to see that the two-way relay allows us to achieve high consistency, because all messages have to go through the network, where a centralized server or a group of peers can confirm and establish a consistent set of data. However, the two-way relay cannot guarantee high responsiveness. It will always depend on the networking resources (latency, bandwidth, processing power), and the only way to make the system more responsive is through improving them.

Another approach to overcome this limitation is to bridge the two flows, which forms a *short-circuit relay* (see Figure 7b). The locally originated messages are now passed back into the local output inside the relay. As before, the function $h$ indicates that the messages may undergo some changes. We do not have to wait for the messages to pass the network and return back us but we short-circuit them back locally. Clearly, we can now achieve high responsiveness but it comes with a price: the local data can become inconsistent with the other nodes. This means that some kind of rollback or negotiation mechanism is required to solve the inconsistencies, when they

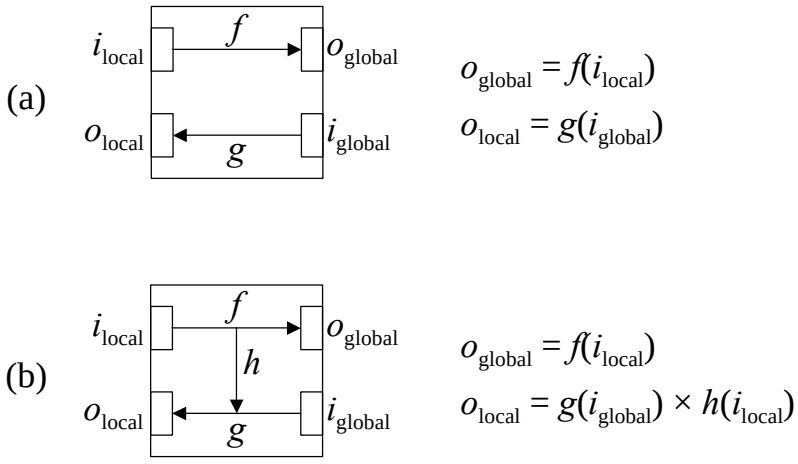



Figure 7: The relay has two alternatives for a structure: (a) A two-way relay sends the local control messages to the network, which sends back data messages to the node. (b) A short-circuit relay sends the local control messages to the network and passes them locally back to the node.

become a problem. However, it should be noted that inconsistent data does not necessarily entail a problem: the problem arises only if two (or more) parties need to interact (or more precisely are aware) with each other and have inconsistent data. In this case, we need a conflict resolution strategy where the parties recognize, negotiate, and agree on the situation. For instance, a foot soldier may content on observing an airplane and some inconsistencies may exist, but another soldier can engage into a gunfight and the parties must agree on their positions and hits.

Data and control architecture defines the nodes' responsibility of the data (see Figure 8). In centralized architecture, the relay mostly conveys local control to the network and receives data from it. This is reversed in distributed architecture. In replicated architecture, the local input and output are a mixture of control and data messages. Each architecture has characteristic problems: in centralized architecture, access to the data may take time; in distributed architecture, the allocation of the data fragments between the nodes must be handled properly; in replicated architecture, updating the data in each replica can be tricky.

## 4.3 Logical Platform and Application

We can discern a three-level hierarchy affecting networked applications: physical platform, logical platform, and application. The logical platform is in-

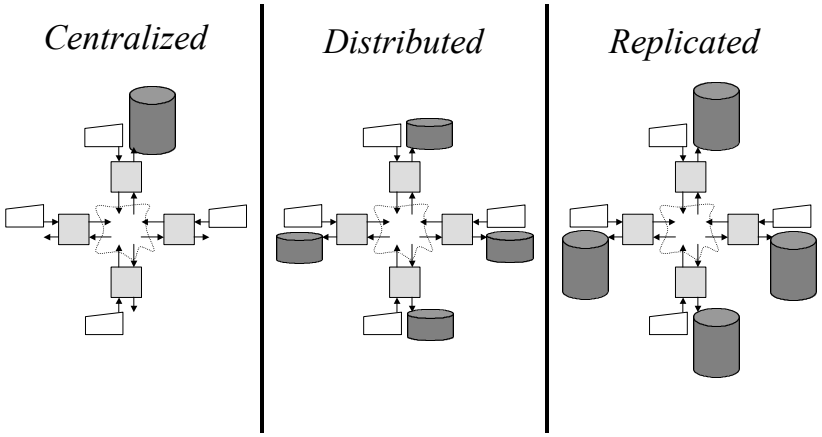



Figure 8: In centralized data architecture, one (data server) node stores all data. In replicated architecture, each node manages a replica of all data. In distributed architecture, the data is distributed among the nodes.

tended for system designers. Here, we have programming language level abstractions like data entities and communication channels. It is important to notice that we need not to know anything about the application itself. Especially in networking, there is an unfortunate tendency to mix logical platform concepts with application level concepts. Data, control and communication architectures do not require knowledge on the application—although they provide the basis for it. The application level adds context interpretation to the data (e.g., an integer value represents position), and, thus, it is related to the end-user.

Closely related issues are *scalability* (the ability to adapt resource changes), *persistence* (leaving and entering the VE), and *collaboration* between players (upholding integrity when sharing an object).

In MCGs scalability concerns, for example, how to construct an online server that dynamically adapts to varying amount of players, or how to allocate the computation of non-player characters among the nodes. To achieve this kind of scalability there must be physical (i.e., hardware-based) parallelism that enables logical (i.e., software) concurrency of computation. Scaling up an MCG brings forth two complementary views: Each new participant naturally burdens the communication resources but, at the same time, it also offers additional computational power to the whole application. In MCGs, the latter viewpoint has not been fully realized yet.

Persistence concerns how a remote node can coexist with an application. Initially, the application has a state and the node must be configured to conform this state (e.g., when players join an online server, they receive the

object data corresponding the current situation). Throughout the gameplay the node and application live in a symbiosis, which is determined by the underlying logical platform. For example, if in a distributed architecture a node gets disconnected abruptly, the application looses the objects maintained by the node. When a node leaves the application, the application must have a mechanism to uphold the game state by forwarding the node's responsibilities. To sum up, persistence must account, among other things, configuration, error detection and recovery, and maintenance on both the application and node.

In MCGs, collaboration usually means that there are team members that act together to achieve a shared goal (e.g., eliminate the other team or overcome some common obstacles). To support collaboration, the MCG has to provide a player with rich and accurate information about the rest of the team. Technically, collaboration requires that the communication between players is prioritized. Interest management bases on the observation that the closer two objects are, the more they communicate with each other. However, the distance between team members does not have to be defined on spatial terms (e.g., they may have implicit knowledge about each other's status or share a dedicated communication channel). Clearly, team is an application-level concept. Because the concept of collaboration distance can be complex, team cooperation consumes resources. Hence, it can be implemented more effectively when the physical level mechanism supports it (e.g., in a LAN with multicasting).

# 5 Concluding Remarks

The research work done on distributed interactive real-time applications provides insight into the problems of networking in multiplayer computer games. We presented the techniques for reducing network traffic and surveyed the relevant literature. Finally, we summarized the key factors of networking in physical, logical and application levels.

If this was the story so far, what lies ahead? Naturally, we will see improvements on the hardware as well as the introduction of novel techniques—and perhaps even new media. Also, the entertainment industry is likely to continue to invest in developing online and mobile gaming. As in any software project, the next logical step is encapsulation—this path has already been taken in graphics rendering where developers use off-the-shelf 3D engines. This is bound to happen also in networking. However, this requires that the underlying concepts and their relationships are analyzed carefully to gain consensus. Therefore, the lessons learned from the past should be

valued.

# References


[1] E. A. Alluisi. The development of technology for collective training: SIMNET, a case history. *Human Factors*, 33(3):343–62, 1991.

[2] R. A. Bangun and H. W. P. Beadle. A network architecture for multiuser networked games on demand. In *Proceedings of International Conference on Information, Communications and Signal Processing*, volume 3, pages 1815–9, New York, NY, 1997.

[3] R. A. Bangun and H. W. P. Beadle. Traffic on a client-server based architecture for multi-user network game applications. In *Proceedings of ICT '97*, volume 1, pages 93–8, Melbourne, Australia, Apr. 1997.

[4] J. W. Barrus, R. C. Waters, and D. B. Anderson. Locales: Supporting large multiuser virtual environments. *IEEE Computer Graphics and Applications*, 16(6):50–7, 1996.

[5] R. Bartle. Interactive multi-user computer games. Technical report, British Telecom, June 1990. Electronic version available at http://www.mud.co.uk/richard/imucg.htm.

[6] N. E. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. In *Proceedings of the Twentieth IEEE Computer and Communication Society INFOCOM Conference*, Anchorage, AK, Apr. 2001.

[7] S. Benford, J. Bowers, L. E. Fahlén, J. Mariani, and T. Rodden. Supporting cooperative work in virtual environments. *Computer Journal*, 37(8):653–68, 1994.

[8] S. Benford, C. Greenhalgh, T. Rodden, and J. Pycock. Collaborative virtual environments. *Communications of the ACM*, 44(7):79–85, 2001.

[9] E. J. Berglund and D. R. Cheriton. Amaze: A multiplayer computer game. *IEEE Software*, 2(3):30–9, 1985.

[10] Y. W. Bernier. Leveling the playing field: Implementing lag compensation to improve the online multiplayer experience. *Game Developer*, 8(6):40–50, June 2001.

[11] P. Bettner and M. Terrano. 1500 archers on a 28.8: Network programming in Age of Empires and beyond. In *The 2001 Game Developer Conference Proceedings*, San Jose, CA, Mar. 2001.

[12] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. Reality built for two: A virtual reality tool. *Computer Graphics*, 24(2):35–6, 1990.

[13] J. Blow. A look at latency in networked games. *Game Developer*, 5(7):28–40, July 1998.

[14] M. S. Borella. Source models of network game traffic. *Computer Communications*, 23(4):403–10, 2000.

[15] W. Broll. DWTP—an Internet protocol for shared virtual environments. In *Proceedings of the 3rd International Symposium on the Virtual Reality Modeling Language*, pages 49–56, Monterey, CA, Feb. 1998.

[16] W. Broll. Smalltool—a toolkit for realizing shared virtual environments on the Internet. *Distributed Systems Engineering*, 5(3):118–28, 1998.

[17] W. Cai, F. B. S. Lee, and L. Chen. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation*, pages 82–9, Atlanta, GA, May 1999.

[18] T. Chiueh. Distributed systems support for networked games. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, pages 99–104, Cape Cod, MA, May 1997.

[19] T. Chiueh, A. Ballman, and P. Pradhan. Distributed system support for network-based multi-user interactive applications. In *Proceedings of 1st Distributed Simulation Symposium*, Orlando, FL, Sept. 1997.

[20] Z. Choukair and D. Retailleau. A QoS model for collaboration through distributed virtual environments. *Journal of Network and Computer Applications*, 23(3):311–34, 2000.

[21] S. Deering. Host extensions for IP multicasting. Internet RFC 1112, Aug. 1989. ftp://ftp.isi.edu/in-notes/rfc1112.txt.

[22] Defense Advanced Research Projects Agency. Synthetic Theater of War. Web page, Mar. 2002. http://stow98.spawar.navy.mil/.

[23] C. Diot and L. Gautier. A distributed architecture for multiplayer interactive applications on the Internet. *IEEE Networks Magazine*, 13(4):6–15, 1999.

[24] D. England and P. Gray. Temporal aspects of interaction in shared virtual worlds. *Interacting with Computers*, 11(1):87–105, 1998.

[25] Y. Fabre, G. Pitel, L. Soubrevilla, E. Marchand, T. Géraud, and A. Demaille. An asynchronous architecture to manage communication, display, and user interaction in distributed virtual environments. In J. D. Mulder and R. van Liere, editors, *Proceedings of the Sixth Eurographics Workshop on Virtual Environments*, pages 105–13, Amsterdam, The Netherlands, June 2000. Springer-Verlag.

[26] Y. Fabre, G. Pitel, L. Soubrevilla, E. Marchand, T. Géraud, and A. Demaille. A framework to dynamically manage distributed virtual environments. In J.-C. Heudin, editor, *Proceedings of the Second International Conference on Virtual Worlds*, volume 1834 of *Lecture Notes in Artificial Intelligence*, pages 54–64, Paris, France, July 2000. Springer-Verlag.

[27] E. Frécon and M. Stenius. DIVE: A scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, 5(3):91–100, 1998.

[28] T. A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 85–92, Monterey, CA, Apr. 1995.

[29] T. A. Funkhouser. Network topologies for scalable multi-user virtual environments. In *Proceedings of the Virtual Reality Annual International Symposium*, pages 222–8, Santa Clara, CA, Mar. 1996.

[30] L. Gautier and C. Diot. Design and evaluation of MiMaze, a multi-player game on the Internet. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, pages 233–6, Austin, TX, July 1998.

[31] C. Greenhalgh. Awareness-based communication management in the MASSIVE systems. *Distributed Systems Engineering*, 5(3):129–37, 1998.

[32] C. Greenhalgh and S. Benford. MASSIVE: A collaborative virtual environment for teleconferencing. *ACM Transactions on Computer-Human Interaction*, 2(3):239–61, 1995.

[33] H. Harada, N. Kawaguchi, A. Iwakawa, K. Matsui, and T. Ohno. Space-sharing architecture for a three-dimensional virtual community. *Distributed Systems Engineering*, 5(3):101–6, 1998.

[34] T. Henderson. Latency and user behaviour on a multiplayer games server. In *Proceedings of Third International Workshop on Networked Group Communication*, pages 1–13, London, UK, Nov. 2001.

[35] T. Henderson and S. Bhatti. Modelling user behaviour in networked games. In *Proceedings of ACM Multimedia*, pages 212–20, Ottawa, Canada, Oct. 2001.

[36] R. Lambright. Distributing object state for networked games using object views. *Game Developer*, 9(3):30–9, Mar. 2002.

[37] R. Lea, Y. Honda, and K. Matsuda. Virtual Society: Collaboration in 3D space on the Internet. *Computer Supported Cooperative Working*, 6(2/3):227–50, 1997.

[38] R. Lea, Y. Honda, K. Matsuda, and S. Matsuda. Community place: Architecture and performance. In *Proceedings of the 2nd Symposium on Virtual Reality Modeling Language*, pages 41–50, Monterey, CA, Feb. 1997.

[39] E. Léty, L. Gautier, and C. Diot. MiMaze, a 3D multi-player game on the Internet. In *Proceedings of the 4th International Conference on Virtual System and Multimedia*, volume 1, pages 84–9, Gifu, Japan, Nov. 1998.

[40] P. Lincroft. The Internet sucks: Or, what I learned coding X-Wing vs. TIE Fighter. *Gamasutra*, Sep. 3, 1999. http://www.gamasutra.com/features/19990903/ lincroft_01.htm.

[41] M. R. Macedonia. *A Network Software Architecture for Large Scale Virtual Environments*. PhD thesis, Naval Postgraduate School, Monterey, CA, June 1995.

[42] M. R. Macedonia and M. J. Zyda. A taxonomy for networked virtual environments. *IEEE Multimedia*, 4(1):48–56, 1997.

[43] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. NPSNET: A network software architecture for large-scale virtual environment. *Presence*, 3(4):265–87, 1994.

[44] J. Mandeville, T. Furness, M. Kawahata, D. Campbell, P. Danset, A. Dahl, J. Dauner, J. Davidson, J. Howell, K. Kandie, and P. Schwartz. GreenSpace: Creating a distributed virtual environment for global applications. In *Proceedings of Networked Reality Workshop*, Boston, MA, Oct. 1995.

[45] M. Mauve, V. Hilt, C. Kuhmünch, and W. Effelsberg. RTP/I—towards a common application level protocol for distributed interactive media. *IEEE Transactions on Multimedia*, 3(1):152–61, 2001.

[46] K. L. Morse, L. Bic, and M. Dillencourt. Interest management in large-scale virtual environments. *Presence*, 9(1):52–68, 2000.

[47] D. L. Neyland. *Virtual Combat: A Guide to Distributed Interactive Simulation*. Stackpole Books, Mechanicsburg, PA, 1997.

[48] V. Normand. The COVEN project: Exploring applicative, technical, and usage dimensions of collaborative virtual environments. *Presence*, 8(2):218–36, 1999.

[49] S. Powers, M. Hinds, and J. Morphett. DEE: An architecture for distributed virtual environment gaming. *Distributed Systems Engineering*, 5(3):107–17, 1998.

[50] D. J. Roberts and P. M. Sharkey. Maximising concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimising the effect of network delays. In *Sixth Workshop on Enabling Technologies*, Cambridge, MA, June 1997.

[51] P. Schwartz, L. Bricker, B. Campbell, T. Furness, K. Inkpen, L. Matheson, N. Nakamura, L.-S. Shen, S. Tanney, and S. Yen. Virtual Playground: Architectures for a shared virtual world. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pages 43–50, 1998.

[52] C. Shaw and M. Green. The MR Toolkit peers package and experiment. In *IEEE Virtual Reality Annual International Symposium*, pages 463–9, Sept. 1993.

[53] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled simulation in virtual reality. *ACM Transactions on Information Systems*, 11(3):287–317, 1993.

[54] S. Shirmohammadi and N. D. Georganas. An end-to-end communication architecture for collaborative virtual environments. *Computer Networks*, 35(2–3):351–67, 2001.

[55] G. Singh, L. Serra, W. Png, and H. Ng. BrickNet: A software toolkit for network-based virtual worlds. *Presence*, 3(1):19–34, 1994.

[56] S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. Addison Wesley, 1999.

[57] S. K. Singhal. *Effective Remote Modeling in Large-Scale Distributed Simulation and Visualization Environments*. PhD thesis, Standford University, Standford, CA, Aug. 1996.

[58] S. K. Singhal and D. R. Cheriton. Exploring position history for efficient remote rendering in networked virtual reality. *Presence*, 4(2):169–93, 1995.

[59] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of networking in multiplayer computer games. In L. W. Sing, W. H. Man, and W. Wai, editors, *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*, pages 74–81, Hong Kong SAR, China, Nov. 2001.

[60] B. Stabell and K. R. Schouten. The story of XPilot. *ACM Crossroads*, 3(2), 1996. http://www.acm.org/crossroads/xrds3-2/xpilot.html.

[61] U.-J. Sung, J.-H. Yang, and K.-Y. Wohn. Concurrency control in CIAO. In *1999 IEEE Virtual Reality Conference*, pages 22–8, Houston, TX, Mar. 1999.

[62] United States Department of Defence. Defence Modeling and Simulation Office. Web page, Mar 2002. http://www.dmso.mil/.

[63] United States Department of Defence, Defence Modeling and Simulation Office. High Level Architecture. Web page, 2002. http://www.dmso.mil/public/transition/hla/.

[64] D. Verna, Y. Fabre, and G. Pitel. Urbi et Orbi: Unusual design and implementation choices for distributed virtual environments. In

H. Thwaites, editor, *VSMM 2000: Sixth International Conference on Virtual Systems and Multimedia*, pages 714–24, Gifu, Japan, Oct. 2000.

[65] J. Vogel and M. Mauve. Consistency control for distributed interactive media. In *Proceedings of ACM Multimedia 2000*, pages 259–67, Ottawa, Canada, Oct. 2001.

[66] Q. Wang, M. Green, and C. Shaw. EM—an environment manager for building networked virtual environments. In *IEEE Virtual Reality Annual International Symposium*, pages 11–8, Mar. 1995.

[67] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W. Yerazunis. Diamond Park and Spline: A social virtual reality system with 3D animation, spoken interaction, and runtime modifiability. Technical Report TR-96-02a, MERL—A Mitsubishi Electric Research Laboratory, Cambridge, MA, Nov. 1996.

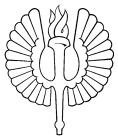
[68] M. Wray and R. Hawkes. Distributed virtual environments and VRML: An event-based architecture. *Computer Networks and ISDN Systems*, 30(1–7):43–51, 1998.

[69] J. J. Yan and H.-J. Choi. Security issues in online games. In L. W. Sing, W. H. Man, and W. Wai, editors, *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*, pages 143–50, Hong Kong SAR, China, Nov. 2001.

[70] S.-J. Yu and Y.-C. Choy. A dynamic message filtering technique for 3D cyberspaces. *Computer Communications*, 24(18):1745–58, 2001.

**Turku Centre for Computer Science**
**Lemminkäisenkatu 14**
**FIN-20520 Turku**
**Finland**

http://www.tucs.fi



**University of Turku**
- **Department of Computer Science**
- **Department of Mathematics**

**Åbo Akademi University**
- **Department of Computer Science**
- **Institute for Advanced Management Systems Research**

**Turku School of Economics and Business Administration**
- **Institute of Information Systems Science**