Aspects of Networking in Multiplayer Computer Games

Conference Paper · November 2001

CITATIONS
130

READS
201

3 authors, including:

Jouni Smed
University of Turku
128 PUBLICATIONS **837** CITATIONS

SEE PROFILE

Aspects of Networking in Multiplayer Computer Games

Jouni Smed, Timo Kaukoranta, Harri Hakonen

Abstract— **Distributed, real-time multiplayer computer games (MCGs) are in the vanguard of utilizing the networking possibilities. Although related research have been done in military simulations, virtual reality systems, and computer supported cooperative working, the suggested solutions diverge from the problems posed by MCGs. With this in mind, this paper provides a concise overview of four aspects affecting networking in MCGs. Firstly, networking resources (bandwidth, latency, and computational power) set the technical boundaries within which the MCG must operate. Secondly, distribution concepts encompass communication architectures (peer-to-peer, client/server, server-network), and both data and control architectures (centralized, distributed, replicated). Thirdly, scalability allows the MCG to adapt to the resource changes by parametrization. Finally, security aims at fighting back against cheating and vandalism, which are common in online gaming.**

Keywords— **Computer games, networking, online entertainment, distributed interactive simulation, virtual environments.**

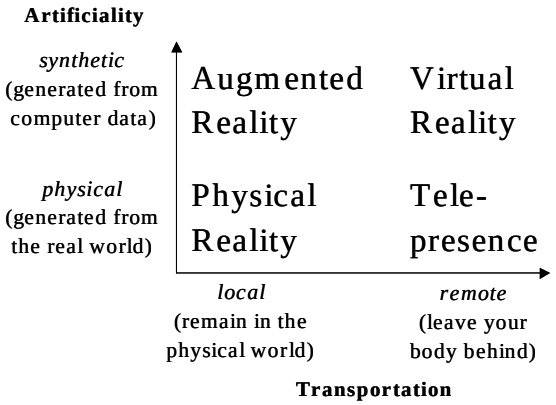


Fig. 1. Classification of shared-space technologies according to transportation and artificiality [2].

I. Introduction

WITH the advent of Internet and wireless communication, multiplayer computer games (MCGs) are becoming more popular. Commercially published computer games are expected to offer a multiplayer option, and, at the same time, online game sites like Electronic Arts' *Ultima Online*, Blizzard Entertainment's *Battle.net* or Microsoft's *MSN Gaming Zone* boast of having hundreds of thousands users. Similarly, the new game console releases rely heavily on the appeal of online gaming, and a whole new branch of mobile entertainment has emerged with intention to develop distributed multiplayer games for wireless applications [1]. In this respect, MCGs will continue to provide both technical and practical challenges in the future.

Generally speaking, MCGs belong to *shared-space technologies*. Figure 1 illustrates a broad classification of shared-space technologies by Benford *et al.* [2]. The *transportation* axis indicates the level to which the participants leave behind their local space, and the *artificiality* axis the level to which a space is computer generated. By using these two dimensions, four strands of technology can be classified: *Physical reality* resides in the local, physical world (i.e., the things are tangible and the participants are corporeal). Conversely, *virtual reality* allows the participants to be immersed in a remote, synthetic world. In *telepresence*, the participants are present at a real world location remote from their physical location (e.g., operating a rover on Mars from Earth). In *augmented reality*, synthetic objects are overlaid on the local environment (e.g., a head-up display indicating the whereabouts of selected targets).

Within this framework, the "virtual reality" category houses various distributed interactive systems, and three branches of research can be easily traced:

- military simulations,
- virtual reality (VR), and
- computer supported collaborative work (CSCW).

Since the 1980s the United States Department of Defense has developed military applications using a protocol for *distributed interactive simulation* (DIS), which was issued as an IEEE standard in 1992 [3], [4]. After that, the military research has concentrated on developing high-level architecture (HLA), which aims at providing a general architecture and services for distributed data exchange. While the DIS protocol is closely linked with the properties of military units and vehicles, the rationale behind HLA is that it could be also used with non-military applications—like computer games. Although these promises have yet to be fulfilled, there has been fruitful cooperation between military and entertainment industries [5].

In turn of the 1990s emerged a new branch of VR research which concentrates on *distributed virtual environments* (DVEs). It has given birth to various experimental systems with the likes of *DIVE* [6], *GreenSpace* [7], *Spline* [8], *Community Place* [9], and *MASSIVE* [10]. While the military research focuses on diverse large-scale systems,

J. Smed and T. Kaukoranta are with the Department of Mathematical Sciences and Turku Centre for Computer Science (TUCS), University of Turku, Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland. E-mail: jouni.smed@cs.utu.fi, timo.kaukoranta@cs.utu.fi. Tel: +358-2-3338684, +358-2-3338797. Fax: +358-2-3338600.

H. Hakonen is with Oy L M Ericsson Ab, Telecom R&D, Joukahaisenkatu 1, FIN-20520 Turku, Finland. E-mail: harri.hakonen@lmf.ericsson.se.

Loo Wai Sing, Wan Hak Man, and Wong Wai (eds.), Proceedings of the International Conference on Application and Development of Computer Games in the 21st Century (ADCOG), pp. 74-81, Hong Kong SAR, China, 2001.

DVEs are mainly designed for local use and support only a small number of participants. In addition, DVEs pay closer attention to the virtual representations of the participants (i.e., avatars) and the ways of interaction and communication.

During the 1990s an effort to combine CSCW with VR has given a rise to *collaborative virtual environments* (CVEs) [11], [12]. CVEs differ from DVEs in that they focus on the collaboration between the avatars (e.g., operating at the same time with a shared object). For example, CVEs have been used in product development [13], in 3D editing [14], and even in game designing [15].

All these advances have come about almost unbeknownst to the MCG developers. Obviously, MCGs have the same qualities as DIS, DVEs or CVEs, but—rather than being lumped together with them—they represent a fourth class of distributed, real-time computer applications. Although the problems of MCGs and online gaming have now been realized by the entertainment industry, it is surprising that they have been dealt marginally in the scientific literature. The academic research concentrates on simple games and limited problem settings [16], [17], [18], [19]. However, people working in the entertainment industry have recently started to publish more openly their ideas and solutions in the trade magazines and conferences [20], [21], [22], [23].

In this paper, we intend to narrow this gap and identify relevant aspects of networking affecting the MCGs. In Section II, we discuss networking resources which constrain the designs for MCGs. Section III focuses on distribution concepts and presents models for communication, data and control architectures. We touch upon scalability issues in Section IV. In Section V, we discuss network security in MCGs. Concluding remarks appear in Section VI.

II. Networking Resources

Distributed simulations face three resource limitations: network bandwidth, network latency, and host processing power for handling the network traffic [24]. These resources refer to the technical attributes of the underlying network and they impose physical restrictions, which the system cannot overcome and which must be considered in the design.

A. Bandwidth

Bandwidth refers to the transmission capacity of a communication line such as a network. Simply put, bandwidth is the proportion of the amount of data transmitted or received per unit time. In WANs (wide area networks), bandwidths range from tens of kbps (bits per second) of dial-up modems up to 1.5 Mbps of T1 and 44.7 Mbps of T3. In LANs (local area networks), bandwidths are much larger ranging from 10 Mbps to 10 Gbps. However, LANs have a limited size and they support a limited number of users, whereas WANs allow global connections.

In addition to how often and how large messages are sent, bandwidth requirements depend on the number and distribution of users. Moreover, bandwidth requirements also depend on the transmission technique (see Fig. 2). Early

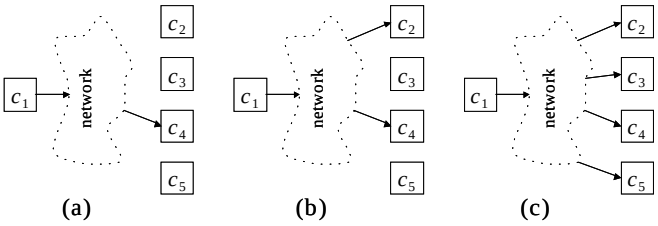


Fig. 2. Transmission techniques: (a) In unicasting, the message is sent to a single receiver. (b) In multicasting, the message is sent to one or more receivers that have joined a multicast group. (c) In broadcasting, the message is sent to all nodes in the network.

implementations in LAN environments broadcast the messages to all participants [25]. Obviously, this leads to problems as the number of participants grows. Unicast communication between a single sender and a single receiver allows to control and direct the traffic. However, since most of the messages are intended to multiple receivers, unicasting wastes bandwidth by sending redundant messages. During the 1990s multicasting [26], which is communication between a single sender and multiple receivers, matured as a technique and gradually gained popularity in distributed systems. Multicasting allows receivers to join groups that interest them. The sender sends only one message (as in unicast) to a group, which is received by multiple receivers (as in broadcast) belonging to the group. There are numerous examples of distributed applications utilizing multicast [3], [6], [8], [10], [27].

B. Latency

Networking latency indicates the length of time (or delay) that incurs when a message gets from one designated node to another. In addition, the variance of latency over time (i.e., jitter) is another feature that affects interactive applications. Latency cannot be totally eliminated. For example, speed-of-light propagation delays and the slowdown of electrical signal in a cable alone yield a latency of 25–30 ms for crossing the Atlantic. Moreover, routing, queuing and packet processing delays add dozens of milliseconds to the overall latency (e.g., in August 2001 a measured average Trans-Atlantic round trip latency[1] was 79.955 ms).

For interactive real-time systems such as MCGs, the rule of thumb is that latency between 0.1 and 1.0 seconds is acceptable. For instance, the DIS standard specifies that the network latency should be less than 100 ms [4]. Latency affects the user's performance nonlinearly: Continuous and fluid control is possible when the latency does not exceed 200 ms, after which the interaction becomes more observational and cognizant. Consequently, the threshold of when latency become inconvenient for the user depends on the type of application. In a real-time strategy (RTS) game, a higher latency (even up to 500 ms) may be acceptable as long as it remains static (i.e., jitter is low) [22]. Interestingly, experiments on CVEs have yielded similar results [12], [28]. On the other hand, games requiring a tight hand-eye motor control such as first person shooters (FPSs) de-

[1] http://www.uu.net/network/latency

mand that the latency runs closer at 100 ms.

C. Computational Power

Network traffic handling puts additional computational strain on the computer running a distributed system. This limited resource can be easily overlooked and usually the network alone is deemed as the source of problems. By using quite conservative estimates, Singhal [24] demonstrates how a simulation handling the network traffic of 100,000 entities (i.e., participating objects) can require up to 80 percent of the total CPU time on a 500 MHz processor. Furthermore, packet delivery demands are unlikely to be satisfied in the future due to increasing processing requirements in distributed systems (e.g., more participants, more interaction) and the slow increase in hardware memory speeds.

III. Distribution Concepts

There is not much we can do about the aforementioned resource limitations. Therefore, the problems of distributed interactive systems should be tackled on a higher level, which means choosing architectures for *communication*, *data*, and *control*. Still, sometimes the architecture alone cannot rid the system of resource limitations, and *compensatory techniques* are needed to relax the requirements.

A. Communication Architectures

A communication architecture can be chosen among different models, which can be arranged as communication graphs according to their *degree of deployment* (see Fig. 3). In a communication graph, the nodes represent the processes running on remote computers and the links denote that the nodes can exchange messages. In the simplest configuration, there is only a *single node* (i.e., one computer and no network). An example of an MCG at this level can employ a split screen to enable two or more players to participate the game.

In *peer-to-peer* architecture, we have a set of equal nodes connected by a network. Since no node is more special than the others, they must be connected to each other. There is no intermediary and each node broadcasts (or multicasts) its messages to every node in the network. Peer-to-peer is widely used in MCGs, because it easy to realize and to expand from a single player game. However, it does not scale up easily due to the lack of hierarchical structure. It is useful when the number participants is small or they communicate in a LAN.

In *client/server* architecture, we have promoted one node to the role of a server. Now, all communication is handled through this server node, while the other nodes remain in the role of a client. For each client, the communication with the server is the most important element, and it needs not worry about the other clients. In contrast, the server becomes the critical part, if it cannot keep up with the network traffic. The client/server architecture is used in the commercial online servers as well as in the classic MUDs (multiuser dungeons).

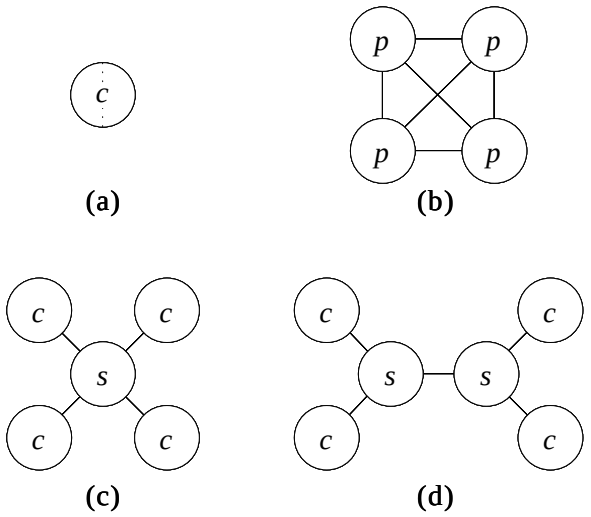


Fig. 3. Degrees of deployment: (a) a split screen on a single computer, (b) a peer-to-peer architecture, (c) a client/server architecture, and (d) a server-network architecture.

In *server-network* (or server pool) architecture, there are several interconnected servers. Here, the communication graph can be thought as a peer-to-peer network of servers over a set of client/server subnetworks. A client is connected to a local server, which is connected to the remote servers and, through them, the remote clients. Server-network reduces the capacity requirements imposed to a server. In consequence, this provides better scalability but increases the complexity of handling the network traffic.

B. Data and Control Architectures

Two attributes define the models for data and control architecture: *consistency* and *responsiveness*. To achieve high consistency, the 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 (or timeliness [29]), the queries made to the data must be responded quickly, which leads to loosely coupled nodes. In this case, the nodes include more computation to reduce the bandwidth and latency requirements. In reality, an architecture cannot achieve both high consistency and high responsiveness at the same time, and the choice of architecture is a trade-off between these two attributes.

Figure 4 illustrates the problem. An application running in a local node sends control messages into a relay and receives data messages from it. In turn, the relay communicates with the relays of other nodes via a network. Here, a relay is a logical concept which illustrates how the control affects the data. There are two alternative structures for the relay (Fig. 5). A *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. A *short-circuit relay* does effectively the same, but it also conveys the input locally

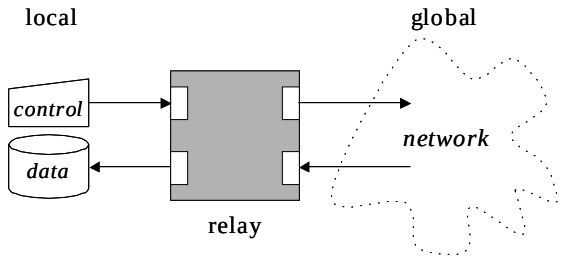


Fig. 4. Architecture defines how messages are relayed between local and remote nodes.

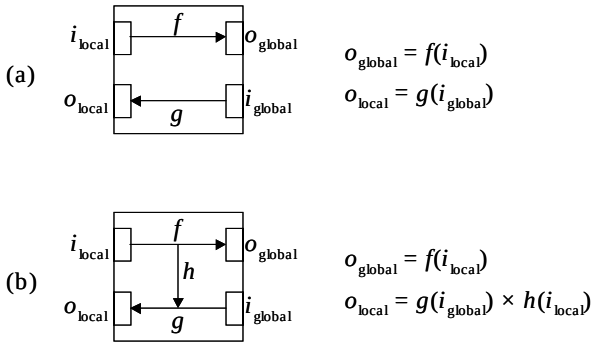


Fig. 5. The relay has two alternatives for a structure: (a) A two-way relay sends the local control messages to the network, which sends back data messages to the node. (b) A short-circuit relay sends the local control messages to the network and passes them locally back to the node.

back to the node. This short circuiting can be realized with immediate feedback (the DIS standard), acknowledgments [6], or buckets delaying arrival of local messages [27].

It is important to differentiate these two structures: A high consistency architecture requires a two-way relay, because all updates require confirmation from the other nodes. On the other hand, high responsiveness entails a short-circuit relay, because the local control messages must appear promptly in the local data. With this in mind, we can now look at the three data and control architectures: centralized, distributed, and replicated.

In a *centralized* architecture, only one node holds the data. Basically, it is shared database that keeps the system consistent at all times. The nodes must use a two-way relay for networking due to the consistency requirements. Obviously, a centralized architecture lacks responsiveness, which is elemental for real-time applications like MCGs.

Distributed and replicated architectures suit better for MCGs because they allow to use the short-circuit relay and, consequently, provide higher responsiveness. In a *distributed* architecture, each node holds a subset of the data. In a *replicated* architecture, a copy of the same data exist in all nodes. The distinction between these architectures is that distributed architecture adapts more easily, for instance, player controlled entities, whose behavior is unpredictable and for whom there can be only one source of commands [30]. Conversely, non-player characters (NPCs) and other computer generated entities are predictable and need not send frequent control messages, and a replicated architecture provides a better alternative. To put it briefly, indeterminism leads to distribution and determinism to replication.

C. Compensatory Techniques

Architectures aim at reducing resource requirements. In most cases, they are not enough and we must reduce communication traffic between nodes. Compensatory techniques provide methods for doing this coherently. We review these techniques briefly (for further details, see [29]).

C.1 Message Compression and Aggregation

By compressing the messages the distributed system can save bandwidth at the cost of computational power. Message aggregation reduces transmission frequency by merging information from multiple messages. Bundling up messages saves bandwidth (less header information) but requires extra computation and weakens the responsiveness.

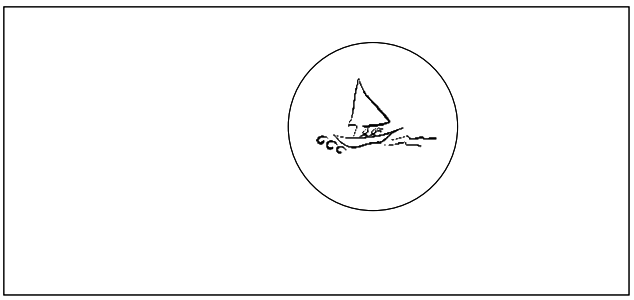
C.2 Interest Management

Interest management includes techniques that allow the nodes to express interest in only the subset of information that is relevant to them [11], [31]. They aim at reducing the number of transmitted messages by specifying the (potentially) interested receivers. 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. Simply put, an aura is an subspace where interaction occurs (Fig. 6). Thus, when two players' auras intersect, they can be aware of each others actions.

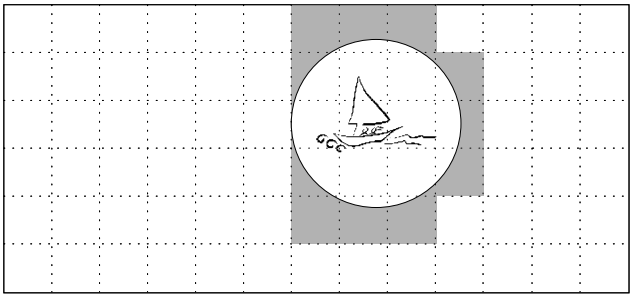
Interest management with auras is always symmetric: If the auras intersect, both parties receive messages from each other. However, aura can be divided further into a focus and a nimbus, which represent observer's perception and observed object's perceptivity [10]. Thus, the player's focus must intersect with another player's nimbus in order to aware of him. By using foci and nimbi it possible to construct a finer-grade message filtering, since the awareness needs not to be symmetric (Fig. 7).

C.3 Dead Reckoning

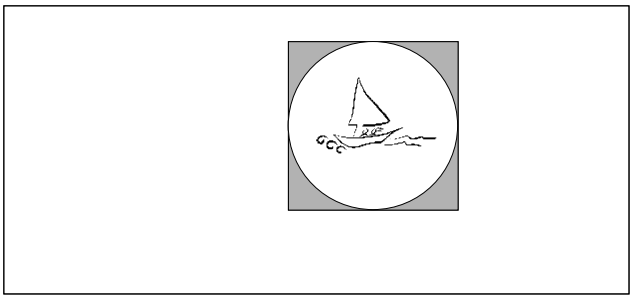
Dead reckoning methods are based on a navigational technique of estimating one's position based on a known starting point and velocity. The same idea can be applied to predicting the data from the other nodes, which allows to prolong the interval of message transmissions and abolish the network latency at the cost of data consistency. When a message is received, the local data is updated accordingly (i.e., we set the known starting point). The message can also include some delta information (e.g., the velocity) which is used in predicting the change of data over time (see Fig. 8). Alternatively, this information can be omitted and the known history be used for extrapolating the data [24]. Moreover, the transmission interval needs not be constant but the messages can be sent only when dead reckoning exceeds some error threshold.



(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.

Fig. 6. Auras (or areas of interest) can be expressed using formulae, cells or extents.

IV. Scalability

Scalability is the ability to adapt to the *resource changes*. In MCGs this 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.

A. Serial and Parallel Execution

The potential speedup obtained by applying multiple nodes is bounded by the system's inherently sequential computations. The time required by the serially executed parts cannot be reduced by parallel computation. Thus, the theoretical speedup S gained is

$$S(n) = \frac{T(1)}{T(n)} \leq \frac{1}{n} \tag{1}$$

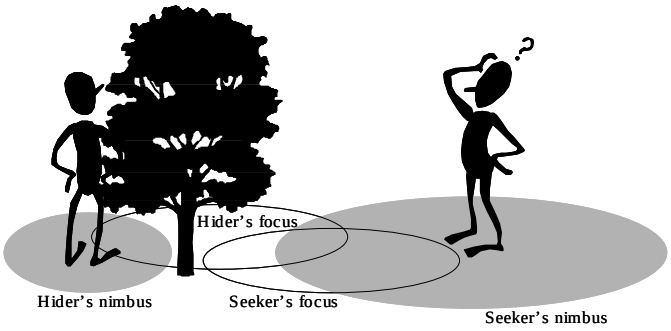


Fig. 7. In hide-and-seek, the nimbus of the hiding person is smaller than the seeker's, and the seeker cannot interact with the hider. Instead, the hider can observe the seeker, since the seeker's nimbus is larger and intersects the hider's focus.

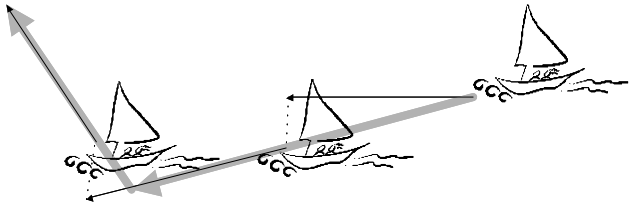


Fig. 8. Dead reckoning is used to calculate positions between the update message. The actual movement (indicated by the gray arrows) differs from the movement predicted by dead reckoning (black arrows). The dotted lines indicate a position change (i.e., a "warp") caused by update message.

where $T(1)$ the time to execute with one node and $T(n)$ with n nodes. The execution time can be divided into a serializable part T_s and parallel part T_p. Let $T_s + T_p = 1$ and $\alpha = T_s/(T_s + T_p)$. If the system is serialized optimally, the equation can be rewritten

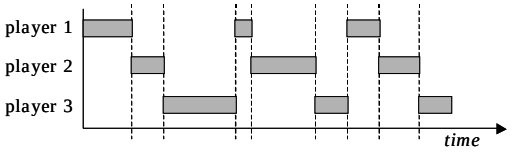
$$S(n) = \frac{T_s + T_p}{T_s + T_p/n} = \frac{1}{\alpha + (1-\alpha)/n} \leq \frac{1}{\alpha}. \tag{2}$$

This is called *Amdahl's law* for a fixed problem setting [32].

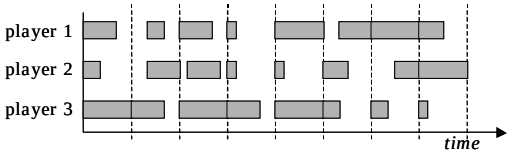
Ideally, the serializable part should be non-existent and, thus, everything would be computed in parallel. However, in that case there cannot exist any coordination between the nodes. The only example of such MCG is that each player is playing their own game regardless of the others. The other extreme is that there is no parallel part with respect to the game state, which is the case in a round-robin or a turn-based game. Between these extremes are the MCGs which provide *real-time interaction* and which, consequently, comprise both parellel and serial computation (see Fig. 9).

For the serializable parts, the nodes must agree on the sequence of events. The most simple approach to realize this is to utilize a client/server architecture, where the server can control the communication by forwarding, filtering and modifying the messages. It should be noted that even in a peer-to-peer architecture the network acts like a server (i.e., the peers share the same serializing communication channel), unless the nodes are connected to each other by a direct physical cable or they communicate by using multicast.

(a) Separate real-time games can run in parallel but without interaction.



(b) A turn-based game is serialized and interactive but not real-time, unless the turns are very short.



(c) An interactive real-time game runs both in serial and in parallel.

Fig. 9. Serial and parallel execution in MCGs.

To concretize, let us calculate the communication capacity in a client/server architecture using unicast. Suppose that each client sends 5 packets per second using IPv6 communication protocol in a 10 Mbps Ethernet. Each packet takes at least a frame of size $68 \cdot 8 + 26 \cdot 8 = 752$ bits (or 94 bytes). Let d equal the number of bits in a message, f the transmission frequency, n the number of unicast connections and C the maximum capacity of the communication channel. Obviously, the following condition must hold

$$d \cdot f \cdot n \leq C. \tag{3}$$

By using values $d = 752 + 32$ (i.e., the payload comprises one 32-bit integer value), $f = 5$ and $C = 10^7$, we can solve the upper bound for the number of clients. Thus, if we are using a client/server architecture, one server can provide serializability for at most 2,551 clients.

B. Communication Capacity

Because the coordination of serialized parts require communication, scalability is limited by the communication capacity requirements of the chosen deployment. Henceforth, we assume that clients are allowed to send messages freely at any moment (i.e., asynchronous messaging). In the worst case, all nodes try to communicate at the same time and the network architecture must handle this traffic without saturation.

Table I collects the magnitudes of communication capacity requirements for different deployment architectures. Obviously, a single node needs no networking. In peer-to-peer, $\sim n$ is reached when all n nodes have direct connections to the other nodes or the communication is handled by multicasting; otherwise, peers use unicasting which yields $\sim n^2$. In client/server, the server-end requires a capacity of $\sim n$, because each client has a dedicated connection to it. In server-network, the server pool has m servers, and n clients are divided evenly among them. If the servers are connected as peer-to-peer, the server communication requires $\sim m \ldots m^2$ in addition to $\sim n/m$ capacity for client communication. If the servers are connected hierarchically (e.g., as a tree), the server at the root is the bottleneck requiring a capacity of $\sim n$.

TABLE I
Communication capacity requirements for different deployment architectures.

Deployment architecture	Capacity requirement
Single node	0
Peer-to-peer	$\sim n \ldots n^2$
Client/server	$\sim n$
Peer-to-peer server-network	$\sim \frac{n}{m} + m \ldots \frac{n}{m} + m^2$
Hierarchical server-network	$\sim n$

Earlier we calculated that a server can support up to 2,551 clients. This demonstrates that, in practice, linear capacity requirement is too large. Therefore, the heart of scalability is to achieve *sublinear communication*. In effect, this means that a client cannot be aware of all the other clients all the time.

To guarantee sublinear communication in a hierarchical server-network we must limit the communication between the servers. Let us suppose that the hierarchy is a k-ary tree. If we can now guarantee that a server sends to its parent $1/k$ th part of its children's messages, we have a logarithmic capacity requirement (i.e., communication in the root is $\sim \log n$). Now, the problem is how to realize this reduction. This where the compensatory techniques provide an answer: Children's messages can be compressed and aggregated, if we can guarantee that the size reduction is $1/k$ on each server level—which is quite unlikely. A more usable solution is, at each step, to apply first interest management (e.g., refrain from passing messages whose potential receivers are already inside the subtree), and then select one of the outgoing messages for the server to pass on. For each suppressed message, the nodes can approximate the information by using dead reckoning.

V. Security and Cheating

Online security has become a major concern for the entertainment industry. The gamesites periodically report on attacks and warn their users against misbehavior and cheating. This problem is unique for MCGs, and it has not been addressed at all in the related research. On the contrary, the military simulations and the DIS standard do not even consider cheating nor security violations—which is understandable since the simulations are not run over a public network and the participants, most of whom belong to military personnel, are considered trustworthy.

Kirmse and Kirmse [33] recognize two security goals for online games: protecting sensitive information (e.g., credit card numbers) and providing a fair playing field (i.e., making cheating is as difficult as possible). Naturally, security and safety *inside* the game world is also an important issue but that falls out of the scope of this paper (see [34]). The

online cheaters are usually motivated by *vandalism* or *dominance*. Only a minority of cheaters try to create open and immediate havoc, whereas most of them want to achieve a dominating, superhuman position and hold sway over the other players.

In the following, we go over the common methods used in online cheating. The sectioning is based on the observations made by Pritchard [35] and Kirmse [36].

A. Packet and Traffic Tampering

In FPS games, a usual way to cheat is to enhance the player's reactions with *reflex augmentation*. For example, an aiming proxy can monitor the network traffic and keep a record of the opponents' positions. When the cheater fires, the proxy uses this information and sends additional rotation and movement control packets before the fire command thus improving the aim. Reversely, in *packet interception* the proxy prevents certain packets from reaching the cheating player. For example, if the packets containing damage information are suppressed, the cheater becomes invulnerable. In a *packet replay* attack, the same packet is sent repeatedly. For example, if a weapon can be fired only once in a second, the cheater can send the fire command packet hundred times a second to boost its firing rate.

A common method for breaking the control protocol is to change bytes in a packet and observe the effects. A straightforward way to prevent this is to use checksums. For this purpose, the MD5 algorithm [37] is widely recommended, because it is well tested, publicly available and fast enough for real-time MCGs. However, there are two weaknesses that cannot be prevented with checksums alone: the cheaters can reverse engineer the checksum algorithm or they can attack with packet replay.

By encrypting the command packets, the proxies have a lesser chance to record and forge information. However, the packet replay attack requires that the packets carry some state information so that even the packets with a similar payload appear to be different. Instead of serial numbering, pseudo random numbers (e.g., the linear congruential method [38]) provide a better alternative. Random numbers can also be used to modify the packets so that even identical packets do not appear the same. Dissimilarity can be further induced by adding a variable amount of junk data to the packets, which eliminates the possibility of analyzing their contents by the size.

B. Information Exposure

A cracked client software may allow a cheater to gain access to the replicated, hidden game data (e.g., the status of other players). On the surface, this kind of passive cheating does not tamper with the network traffic, but the cheaters can base their decisions on more accurate knowledge than they are supposed to have. For example, typical exposed data in RTS games are the variables controlling the visible area on the screen (i.e., the fog of war). This problem is common also in FPSs, where compromised graphics rendering drivers may allow the player to see through walls.

Strictly speaking, information exposure problems stem from the software and cannot be prevented with networking alone. Clearly, the sensitive data should be encoded and its location in the memory should be hard to detect. Nevertheless, it is always susceptible to ingenious hackers and, therefore, requires some additional countermeasures. In a centralized architecture, an obvious solution is to utilize the server, which can check whether a client issuing a command is actually aware of the object with which it is operating. For example, if a player has not seen the opponent's base, he cannot give an order to attack it—unless he is cheating. When the server detects cheating, it can drop out the cheating client. A democratized version of the same method can be applied in a replicated architecture: Every node checks the validity of each other's commands, and if some discrepancy is detected, the nodes vote whether its source should be debarred from participating the game.

C. Design Defects

Network traffic and software are not the only vulnerable places in a MCG, but design defects can create loopholes which the cheaters are apt to exploit. For example, if the clients are designed to trust each other, the game is unshielded from client authority abuse. In that case, a compromised client can exaggerate the damage caused by a cheater, and the rest accept this information as such. Although this problem can be tackled by using checksums to ensure that each client has the same binaries, it is more advisable to alter the design so that the clients can issue command requests, which the server puts into operation.

In addition to poor design, distribution—especially the heterogeneity of network environments—can be the source of unexpected behavior. For instance, there may be features that become eminent only when the latency is extremely high or when the server is under a denial-of-service attack.

VI. Concluding Remarks

This paper provided an overview of the problems of networking in MCGs and offered a cursory glance to the topics. We discussed the connection of MCGs to the relevant research on other distributed systems. First, we presented three aspects of MCGs ranging from networking resources to distribution and scalability. Lastly, we described the problems of ensuring security in MCGs.

MCGs open up possibilities for future work. Ideas originating from the related research should be evaluated and their applicability in MCGs tested. Also, a closer look at the work done on cryptography could yield better methods for online security. Finally, the practicers should be made aware of the advances in research and be invited to bring their own insights into the discussion.

References

[1] Peter Clarke, "Mobile giants pick each other for universal games," *EE Times*, Mar. 21, 2001, http://www.eetimes.com/story/OEG20010321S0069.

[2] Steve Benford, Chris Greenhalgh, Gail Reynard, Chris Brown, and Boriana Koleva, "Understanding and constructing shared

spaces with mixed-reality boundaries," *ACM Transactions on Computer-Human Interaction*, vol. 5, no. 3, pp. 185–223, 1998.

[3] Michael R. Macedonia, *A Network Software Architecture for Large Scale Virtual Environments*, Ph.D. thesis, Naval Postgraduate School, Monterey, CA, June 1995.

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

[5] Michael Capps, Perry McDowell, and Michael Zyda, "A future for entertainment-defense research collaboration," *IEEE Computer Graphics and Applications*, vol. 21, no. 1, pp. 37–43, 2001.

[6] Emmanuel Frécon and Mårten Stenius, "DIVE: A scaleable network architecture for distributed virtual environments," *Distributed Systems Engineering*, vol. 5, no. 3, pp. 91–100, 1998.

[7] Jon Mandeville, Thomas Furness, Masahiro Kawahata, Dace Campbell, Paul Danset, Austin Dahl, Jens Dauner, Jim Davidson, Jon Howell, Kigen Kandie, and Paul Schwartz, "GreenSpace: Creating a distributed virtual environment for global applications," in *Proceedings of Networked Reality Workshop*, Boston, MA, Oct. 1995.

[8] John W. Barrus, Richard C. Waters, and David B. Anderson, "Locales: Supporting large multiuser virtual environments," *IEEE Computer Graphics and Applications*, vol. 16, no. 6, pp. 50–7, 1996.

[9] Rodger Lea, Yasuaki Honda, and Kouichi Matsuda, "Virtual Society: Collaboration in 3D space on the Internet," *Computer Supported Cooperative Working*, vol. 6, no. 2/3, pp. 227–50, 1997.

[10] Chris Greenhalgh, "Awareness-based communication management in the MASSIVE systems," *Distributed Systems Engineering*, vol. 5, no. 3, pp. 129–37, 1998.

[11] Steve Benford, Chris Greenhalgh, Tom Rodden, and James Pycock, "Collaborative virtual environments," *Communications of the ACM*, vol. 44, no. 7, pp. 79–85, 2001.

[12] Shervin Shirmohammadi and Nicolas D. Georganas, "An end-to-end communication architecture for collaborative virtual environments," *Computer Networks*, vol. 35, no. 2–3, pp. 351–67, 2001.

[13] John Maxfield, Terrence Fernando, and Peter Dew, "A distributed virtual environment for collaborative engineering," *Presence*, vol. 7, no. 3, pp. 241–61, 1998.

[14] Ricardo Galli, *Data Consistency Methods for Collaborative 3D Editing*, Ph.D. thesis, Universitat de les Illes Balears, Palma de Mallorca, Spain, Nov. 2000.

[15] Kamen Kanev and Tomoyuki Sugiyama, "Design and simulation of interactive 3D computer games," *Computers & Graphics*, vol. 22, no. 2–3, pp. 281–300, 1998.

[16] Eric J. Berglund and David R. Cheriton, "Amaze: A multiplayer computer game," *IEEE Software*, vol. 2, no. 3, pp. 30–9, 1985.

[17] Bjørn Stabell and Ken Ronny Schouten, "The story of XPilot," *ACM Crossroads*, vol. 3, no. 2, 1996, http://www.acm.org/crossroads/xrds3-2/xpilot.html.

[18] Emmanuel Léty, Laurent Gautier, and Christophe Diot, "MiMaze, a 3D multi-player game on the Internet," in *Proceedings of the 4th International Conference on Virtual System and Multimedia*, Gifu, Japan, Nov. 1998, vol. 1, pp. 84–9.

[19] Simon Powers, Mike Hinds, and Jason Morphett, "DEE: An architecture for distributed virtual environment gaming," *Distributed Systems Engineering*, vol. 5, no. 3, pp. 107–17, 1998.

[20] Jonathan Blow, "A look at latency in networked games," *Game Developer*, vol. 5, no. 7, pp. 28–40, July 1998.

[21] Peter 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.

[22] Paul Bettner and Mark 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.

[23] Yahn W. Bernier, "Leveling the playing field: Implementing lag compensation to improve the online multiplayer experience," *Game Developer*, vol. 8, no. 6, pp. 40–50, June 2001.

[24] Sandeep K. Singhal, *Effective Remote Modeling in Large-Scale Distributed Simulation and Visualization Environments*, Ph.D. thesis, Standford University, Standford, CA, Aug. 1996.

[25] Michael R. Macedonia and Michael J. Zyda, "A taxonomy for networked virtual environments," *IEEE Multimedia*, vol. 4, no. 1, pp. 48–56, 1997.

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

[27] Christophe Diot and Laurent Gautier, "A distributed architecture for multiplayer interactive applications on the Internet," *IEEE Networks Magazine*, vol. 13, no. 4, pp. 6–15, 1999.

[28] Kyoung Shin Park and Robert V. Kenyon, "Effects of network characteristics on human performance in a collaborative virtual environment," in *Proceedings of IEEE International Conference on Virtual Reality*, Houston, TX, Mar. 1999.

[29] Sandeep Singhal and Michael Zyda, *Networked Virtual Environments: Design and Implementation*, Addison Wesley, 1999.

[30] Didier Verna, Yoann Fabre, and Guillaume Pitel, "Urbi et Orbi: Unusual design and implementation choices for distributed virtual environments," in *VSMM 2000: Sixth International Conference on Virtual Systems and Multimedia*, Hal Thwaites, Ed., Gifu, Japan, Oct. 2000, pp. 714–24.

[31] Katherine L. Morse, Lubomir Bic, and Michael Dillencourt, "Interest management in large-scale virtual environments," *Presence*, vol. 9, no. 1, pp. 52–68, 2000.

[32] John L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–3, 1988.

[33] Andrew Kirmse and Chris Kirmse, "Security in online games," *Game Developer*, vol. 4, no. 4, pp. 20–8, July 1997.

[34] Derek Sanderson, "Online justice systems," *Game Developer*, vol. 6, no. 4, pp. 42–9, Apr. 1999.

[35] Matt Pritchard, "How to hurt hackers: The scoop on Internet cheating and how you can combat it," *Gamasutra*, July 24, 2000, http://www.gamasutra.com/features/20000724/pritchard_01.htm.

[36] Andrew Kirmse, "A network protocol for online games," in *Game Programming Gems*, Mark DeLoura, Ed., pp. 104–8. Charles River Media, 2000.

[37] Ronald Rivest, "The MD5 message digest algorithm," Internet RFC 1321, 1992, http://theory.lcs.mit.edu/~rivest/Rivest-MD5.txt.

[38] Donald E. Knuth, *Seminumerical Algorithms*, vol. 2 of *The Art of Computer Programming*, Addison-Wesley, Reading, MA, 1969.