DiVES: A Distributed Support for Networked Virtual Environments

A.Bonotti, L.Genovali, L.Ricci
Dipartimento di Informatica, Università di Pisa
Largo B. Pontecorvo, 56125-Pisa (Italy)
{ricci}@di.unipi.it

Abstract

This paper presents $DiVES$, a distributed support for the development of networked Distributed Virtual Environments. $DiVES$ exploits the publish subscribe interaction model to define a flexible communication support. An acyclic peer to peer network of brokers has been defined to support an event based communication framework. The network can be dynamically reconfigured and it can tolerate broker crashes by a proper recovery mechanism. A set of optimization strategies of the basic publish/subscribe routing mechanism has been defined through an accurate analysis of the information exchanged in DVE applications. The message traffic on the network is reduced by packing notifications and filters into a single message. Furthermore, approximated filters are introduced to further reduce message traffic. Advertisement are exploited to optimize the routing of filters.

1 Introduction

Distributed Virtual Environments (DVE) [3, 7] are one of the most interesting applications on local and wide area networks. This set of applications includes, for instance, multiplayer games. In a multiplayer game different players connected to a local or to a wide area network interact within a virtual shared world. The avatars controlled by the players generally exchange information regarding their state, their position in the virtual world, their colour or shape. In first person shooters, for instance, each avatar needs the positions of the neighbouring avatars to shoot them. Even if multiplayer games will be considered in the following as a sample application of DVE, our approach can be applied to $DVEs$ in general.

The definition of a support to implement $DVEs$ is a challenging issue and requires the solution of a set of classical problems of distributed systems.

First of all, a suitable support for the inter player communications has to be defined. A naive solution broadcasts the state updates to all other players but this is feasible in a LAN only, where a hardware support for broadcast is generally available and the number of players is small. The main drawbacks of this approach are the large amount of messages, and the number of useless message received by each host. While the former problem can be solved through a suitable communication support, the second issue is more serious because a host with a low bandwidth connection to the network may experience an inadmissible slowdown of the game, due to a bad usage of the available bandwidth and to the CPU time to filter useless messages. On the other hand, in multiplayer games, each player is often interested in a small portion of the shared world, i.e. the world zone surrounding it. Furthermore, each player is generally interested in receiving message only from other entities placed in this zone. This concept has been formalized through the notion of *Area of Interest* of a player P [5] that is a portion of the virtual world including entities that may interact with P. The extent of this zone of the virtual world depends on the characteristics of P, for instance upon his sight capability.

In the following we will consider a *Communication Group* as a pair defining a communication channel and a group of hosts interested in receiving all the message sent to that channel.

Two alternative strategies can be adopted to implement an *Area of Interest*. *Cell based approach*, the simplest one, defines an *Approximation* of the area of interest by statically partitioning the shared world into cells [8] and by pairing a different *Communication Group* to each cell. The area of interest of a player is approximated by the region including the cell where it is currently placed and the surrounding ones. Each player dynamically joins and leaves communication groups while moving in the virtual world, during the evolution of the game. Each player notifies its position to the communication group corresponding to its cell and joins the communication groups corresponding to the surrounding cells. In this way the player receives any message sent by players belonging to the surroundings areas. [7] discusses pro and cons of several alternative choices for the

Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)
1550-445X/06 $20.00 © 2006 IEEE

shapes of the cells.

The second approach,the *entity based approach*, [5] pairs a distinct communication group with each player P and P sends its position to this communication group. Each player joins the communication groups of all the players belonging to its area of interest. To detect these players, a cell based approach is exploited.

Group communications can be implemented by exploiting either *IP Multicast* or publish subscribe systems. Both these strategies are described in section 2.

This paper presents $DiVES$, a communication support for $DVEs$ that exploits a publish subscribe approach to support communications among virtual entities. In $DiVES$ each player periodically notifies, i.e. publishes, its position within a virtual $2D$ shared world. Subscriptions are defined by $filters$ describing the events in which each player is interested. A filter generally defines a circular regions of the shared world including an area surrounding a player.

$DiVES$ defines an acyclic peer to peer network of brokers to route the messages among the virtual entities. The system can recover broker crashes through a simple recovery mechanism. Routing is optimized by fully exploiting the peculiar characteristics of the messages exchanged in multiplayer games. For instance, each filter in $DiVeS$ is always produced at the same time of a publication. Messages exchanged through the network are reduced by packing each filter and the corresponding notification in a single message. To reduce the number of filters transmitted on the network, $DiVES$ introduces the notion of *Predicted Area of Interest*. This notion extends the notion of *Area of Interest* of a player P by considering the portion of the world which will be interesting for P in the next interval of time Δt. The number of useless messages received by a virtual entity can be tuned by this parameter. $DiVES$ exploits also $advertisements$ [2] to optimize the routing of filters. Consistency of the distributed shared environment is guaranteed by adopting a *local-lag* based solution [1]

Section 2 reviews the most important proposals to define communication supports for multiplayer games. The overall architecture of DiVES is presented in Section 3. Section 4 presents the publish/subscribe system defined by $DiVES$ while Section 5 describes the defined optimizations. Finally, Section 6 describes the implementation we are currently developing and presents some experiments.

2 Related Work

Current implementations of group communications generally exploit IP Multicast and publish/subscribe systems.

IP Multicast has been successfully exploited in a cell-based multiplayer support [5] where a distinct multicast group is statically paired with each cell of the virtual world and the players dynamically joins and leave the multicast groups when moving within the virtual world. However, mapping of communication groups to multicast groups is not always so trivial. For instance, in entity based approach, the multicast groups are dynamically created according to the position of the players. This imply further communications among the players to detect which multicast groups they have to join. Furthermore, even if multicast is hardware supported in most LAN, IP multicast is not widely supported by network routers, and just an Internet subset supports this protocol.

Publish/subscribe [6, 2] is a valid alternative to implement group communication. In a publish/subscribe system, hosts delivers $events$ and publish them by $notifications$. Furthermore, hosts may express their interest in an event or in a pattern of events, in order to be notified of any event generated by a publisher and matching their interest. The underlying support is able to match these subscriptions with publications. Each subscription is defined by a *filter*.

Matching of subscriptions and publications is generally supported by a network of brokers which also routes notifications to interested subscribers. In [2] several routing algorithms are described. The simplest one is based on *flooding*, i.e. each notification is sent any broker of the network. A Broker connected to one or more hosts filters the incoming notifications according to the subscriptions delivered by its hosts.

Filter Based Routing is a more sophisticated approach [2] where notifications are routed according to the filters delivered by the hosts. This approach is shown in Fig. 1.

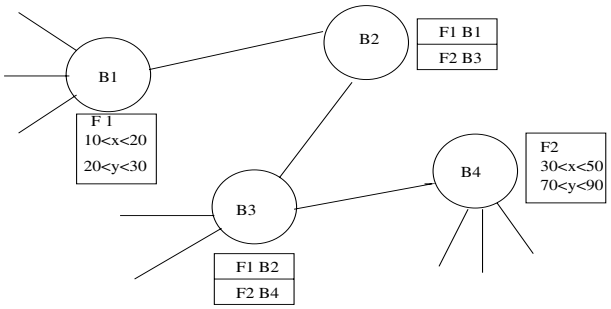

Figure 1. Filter Based Routing

The network of brokers shown in the Fig.1 implements a publish/subscribe system. Brokers B_1, and B_4 are *Terminal Brokers* since they are connected to the hosts. The remaining brokers, i.e. B_2 and B_3 are *Internal Brokers*. The routing tables of B_1, B_4 include some simple subscriptions delivered from the corresponding hosts. In this case, simple linear restraints can be defined by the subscription language. Similar subscriptions are often delivered in multiplayer applications. Terminal brokers forwards to the network the filter delivered by their clients hosts. When an internal bro-

ker B_i receives a filter F from a broker B_j, B_i stores the pair (F, B_j) in its routing table. These tables are exploited to route notifications toward interested clients. For instance, if B_3 receives the notification $(x = 40, y = 80)$, it forwards the notification to B_4 only.

The basic idea of *advertisement based routing* is that each host periodically delivers a filter describing the set of notifications it is going to publish in the next future. Afterwords, only the notifications matching an advertisement will be considered valid. Each broker exploits a further routing table, the *advertisement table*, storing all the received advertisements. When a broker receives a filter F, it forwards F only to the brokers controlling subnets including at least one host which has delivered at least an advertisement intersecting F. Advertisements are periodically delivered by hosts and forwarded in the network by a flooding mechanism.

Recently, several aspects of the publish/subscribe paradigm have been deeply investigated. [2] presents a formal specification of a publish/subscribe system, a routing framework and a set of routing algorithms. [2] defines three critical design issue for a publish/subscribe system, i.e. the interconnection topology of the brokers, the routing algorithms and the processing strategy defining the distribution of the computational load of the matching algorithm on the network of brokers. Several solutions are presented for each issue. [2] introduces advertisement based routing as well and presents $SIENA$, an event notification system.

To the best of our knowledge, only a few systems [3, 7] exploit the publish/subscribe paradigm to support multiplayer games. In [7] different partitioning strategies of the virtual world are considered and a channel based publish/subscribe model is exploited. According to this approach a different abstract channel is defined for each cell of the world. The resulting system is less flexible than $DiVES$, since it implements only the cell based approach.

$Mercury$ [3] is a publish/subscribe system specifically defined to support multiplayer games. Mercury partitions the brokers into a set of $hubs$ and distribute the subscriptions to these hubs. Each broker manages any filter intersecting a given region of the shared world. Routing of notification is not optimized since each notifications is sent to all the hubs.

[4] presents an approach to interest management based upon the predicted movement of players in the virtual world and exploits a publish/subscribe system where the frequency of the message exchange depends upon the possibility that player will interact in the near future.

3 DiVES: the Overall Architecture

The communication support of $DiVES$ optimizes the basic publish/subscribe approach by considering the communication patterns exploited in multiplayer games. Here we assume that each event notified by players describes the player position within the virtual world. Other events, for instance events describing shoots, are notified in a similar way [1].

The $DiVES$ communication support is implemented by an *acyclic peer to peer network of brokers*. Brokers can be connected to the hosts and to other brokers. Each player is connected to a single broker. The terminal broker connected to the player P_i will be referred as $Broker(P_i)$.

$DiVES$ defines a master-broker M associated with a known *IP address*. Each player P and/or broker willing to join the $DiVES$ network connects to M to receive from it the list of the brokers currently available in the system. Then P chooses a broker from the list and connects to it. This choice is guided by several parameters defining the QoS of the different brokers. First of all, for each broker B, the number of connections opened by B is considered. Each connection is weighed by a value defining the average load of the corresponding connection. Then, the latency of the communication with each B is considered. This value is estimated in term of the time to receive a reply form B. The full joining algorithm is described in [1]

A simple recovery mechanism is defined by $DiVES$. Each broker detects the crash of a neighbour through timeouts associated to the connection with its neighbours. The description of the recovery mechanism is outside the scope of this paper and is fully described in [1].

Finally $DiVES$ adopts a mechanism to define a correct ordering of events. Ordering of events is required in distributed supports to avoid fuzzy situations. Let us suppose that a player P shoots to player Q and let us consider a third player O observing this event. The system may produce two different events, the first one corresponding to the shoot of P, and the other one to the death of Q. If the second event is notified to O before the first one, O observes the player dying before it has been shot. $DiVES$ exploits the $local-lag$ concept to guarantee consistency.

4 DiVES: the Publish/Subscribe Model

This section describes the publish/subscribe architecture defined by $DiVES$. Several routing algorithms are defined.

In $DiVES$, each player P publishes its current position within a $2D$ virtual world through a notification, *notify(x,y)*, where x and y are integer values defining the player coordinates in the virtual world. Any subscription produced by a player P defines a zone of the virtual world which may be equivalent to the *Area of interest* of P or approximates this area. These choice depends by the grouping communication model chosen, i.e. the cell-based model vs. the entity based one. Each notification is described by a *filter* which defines a zone of the virtual world. The filter depends on

the shape of the subscribed zone, for instance rectangular regions are defined by a pair of linear restraints. Instead, circular regions may be defined by the current position of the player and a value defining the radius of the zone.

In the following, we will suppose that $Broker(P)$ records the characteristics of player P. In this way, $Broker(P)$ can, for instance, compute autonomously the *Area of Interest* of all its players. Furthermore, we will suppose that each broker may access a copy of the partitioned map of the shared world.

As described in the following sections, $DiVES$ implements both the cell-based approach and the entity based one. Several optimizations are proposed in the context of the entity-based approach.

4.1 Cell-Based DiVES

As discussed in sect. 1, a cell-based approach partitions the shared virtual world into cells and pairs a different communication group with each cell. Each player joins the communications groups corresponding both to its cell and to the surrounding ones. A communication group is defined as the set of players whose subscriptions intersect a given cell C. Any notification delivered by players belonging to C will be forwarded to all these processes.

This is implemented in $DiVES$ by defining proper filters. The virtual world is partitioned into a set of square cells and each player delivers filters describing the square region including its cell and the neighbouring ones. In the following, we will suppose that the area of interest of any player is always included in a cell of the virtual world.

The filter delivered by a player P is not modified until P moves within the same cell. For any movement, P notifies its new position to $Broker(P)$. When P crosses the border of a cell it notifies this event to $Broker(P)$, that exploits both the position of P and the map of the shared world to define its new filter. Let us suppose that a terminal broker B is connected to a single player P. When B receives a notification from the network, it computes the *Area of Interest* of P by exploiting both the last position notified by P and the characteristics of P. Notifications received from players located outside the area of interest P are not sent to P. In this way, the notification filtering is performed by $Broker(P)$ and P is not overwhelmed by useless messages. Both input bandwidth and computational load of P are therefore optimized.

The trade off of this approach is between the number of subscriptions, i.e. filters delivered on the network and the number of useless notifications delivered to each broker. Since a player delivers a new filter when crossing a cell boundary only, the number of delivered subscriptions can be controlled by tuning the size of the cells. Small cell size defines a good approximation of the area of interest of a player, but implies a large number of subscriptions.

4.2 Entity-Based DiVES

We recall that in the entity based approach, a different communication group is associated with each entity, i.e. with each player. Each player joins the communications groups paired with the entities belonging to its area of interest.

In the entity-based approach, a communication group includes all the processes whose areas of interest overlap. Consider, for instance, Fig. 2. The dashed area, corresponding to the intersection of the areas of interest of P_1, P_2, P_3, defines a communication group including these processes. Any notification sent within this area will be received by these processes. It is important to notice that, even if a process P belongs to the area of interest of process Q, this does not imply that Q belongs to the area of interest of P. For instance process P_4 belongs to the intersection of the area of interest of P_1, P_2, P_3, but its area of interest, shown in the figure by the dashed line, does not include any of the previous processes. This implies that all the notification of P_4 will be sent to P_1, P_2, P_3, but no notification of P_1/P_2/P_3 will be sent to P_4.

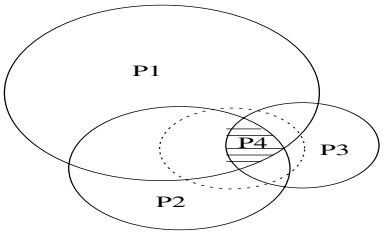

Figure 2. Entity-Based group Definition

In $DiVES$ each player delivers a filter defining its exact *Area of Interest*. In this way, the position of each player belonging to this area will be notified to P. This approach cannot be directly implemented because it requires the exchange of a large amount of filters. As a matter of fact, the area of interest of any player P is modified at each movement of P. This implies that a new filter is generated for each movement of any player and this is the main drawback of this approach. On the other hand, since filters are more accurate than in the cell-based approach, routing of notifications improves and no useless notification are sent to a broker.

Next section shows several optimizations to improve the effectiveness of this method. The resulting approach is a compromise between the cell-based and the entity-based approaches.

5 Reducing the Message Traffic

Let us introduce some preliminary definitions. We will consider the behaviour of each player during an interval of time Δt. Suppose that each player is characterized at any time by a speed v. We define the *Predicted Notification Area* (PNA) of a player P as the set of positions that can be reached by P starting from its current position, traveling along a straight line, during the interval Δt. This set of positions is defined by a circle whose center is the current position of P and depends on the speed of P. We define the *Predicted Area of Interest* (PAI) of P as the subspace of the virtual world including all the areas of interests corresponding to any point in the *Predicted Notification Area*. It can be easily shown that any *Predicted Notification Area* is a subset of the corresponding *Predicted Area of Interest*.

The main idea is that P subscribes its PAI, rather than its exact *Area of Interest*. The same subscription remains valid as long as the current area of interest of a player is included in its PAI. When adopting this approach the amounts of filters exchanged through the network, can be tuned by modifying Δt. Since Δt can be updated according to the characteristics and the state of each player, the resulting strategy is more flexible than the cell based approach. As in the cell-based approach, brokers filters incoming notifications according to the area of interest of their player.

Let us consider the example in Figure 3. The left part of the picture shows four players at different positions of the virtual world. The right part shows the brokers network and the relevant part of their routing tables. Let us consider player P_1. When it notifies its position to B_1, B_1 computes the area of interest Y of P_1, and its PAI X. Then B_1 forwards the PAI to its neighbours and the PAI will be flooded on the network. Since both P_2 and P_3 belongs to the PAI of P_1, their positions will be notified to P_1 by the routing algorithm. B_1 discards any notification it receives from P_3, because it does not belong to the Y. On the other side, the notification of P_2 will be sent to P_1 because it belongs to Y. Note that Y is notified to B_1 anytime P_1 moves, but B_1 does not propagate Y on the network. The position of P_4 is not routed to P_1, because P_4 does not belong to the PAI of P_1. Let us now suppose that P_1 moves from position b to position c. B_1 detects that the area of interest of P_1 does not belong to the PAI of P_1. The new PAI of P_1 is computed and flooded on the network. If P_4 now belongs to the PAI of P_1, its notifications will be routed to P_1.

A further reduction of the number of messages flowing in the network is obtained by an accurate analysis of the kind of messages produced by multiplayers applications. A new *Predicted Notification Area* is delivered at the same time of a new notification. A new *Predicted Area of Interest* is delivered when the current area of interest is not included in the previous *Predicted Area of Interest* and this is associated

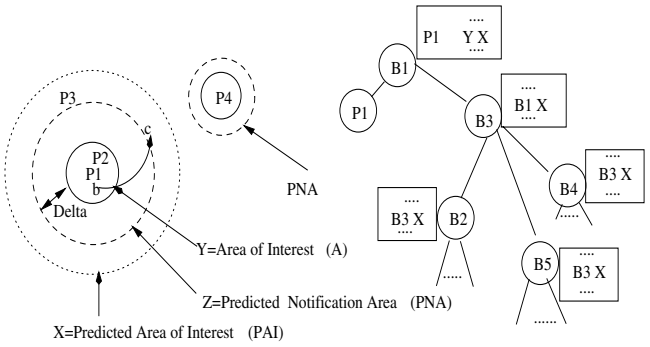

Figure 3. The Optimized Routing Algorithm

again to a movement of a player and, hence, to a notification. In this way, a larger amount of information is packed into the same message. We have defined different kinds of message. For instance, $position(x, y)$ corresponds to the notification of a new position, while $positionpai(x, y, pai)$ corresponds the notification of a new position and of a new PAI. Further messages are exploited in *advertisement based routing*.

Each broker B_i routes a message $position(x, y)$ according its routing table, while the message $positionpai(x, y, pai)$ implies both the forwarding of the notification $notify(x, y)$ according to the routing table and the forwarding of the filter PAI to all the neighbours.

The described approach is based on a simple filter routing strategy, that is flooding. Advertisements proposed in [2] are exploited to optimize the routing of filters on the network.

In $DiVES$ advertisements corresponds to *Predicted Notification Areas*. As a matter of fact, PAI defines the positions each player can reach during the next interval of time Δt. These positions define the notifications P is going to deliver in the next interval. Advertisements, i.e. $PNAs$, are periodically injected in the network. Since a notification is considered valid only if it belongs to the last advertisement delivered on the network, a new PNA is produced when the player approaches the border of its current PNA. The distance from the border depends upon the latency of the underlying network, because any broker has to be notified before the player crosses its PNA. Routing of advertisements is based on flooding. As for PAI, a single message $positionpna(x, y, pna)$ carries both a notifications and a PNA. Each broker propagates notifications according to the routing table and forwards advertisements to all its neighbours. Routing of filters is optimized, because each broker forwards any filter F only to the neighbours storing in their advertisement table at least an advertisement overlapping F.

Consider again Figure 3. The dotted circle around P_4

represents the PNA of P_4. This information is flooded in the network and is registered in the advertisement table of each broker. Let us suppose that P_1 is located at b and that broker B_3 receives from B_1 the PAI of P_1. Suppose also that $B_4 = Broker(P_4)$ Since no overlap exists between the PAI of P_1 and the PNA of P_4, this PAI is not sent to B_4. A further optimization is related to the reduction of the entries of the routing tables. [2] proposes a strategy to merge filters at internal nodes. A broker can merge the filters corresponding to existing routing entries and forward this merger to a subset of its neighbours. Merging filters introduces another level of approximation in the routing algorithm. $DiVES$ adopts a simple strategy to merge filters. Two filters are merged if and only if size of the resulting region is does not exceed a given threshold. This guarantees that the corresponding filter defines a limited region of the shared world.

6 Experimental Results and Future Works

A testbed implementation of $DiVES$ has been developed on a cluster of 40 $Linux$ hosts interconnected by a fast Ethernet LAN. A set of multiplayer applications characterized by different feautures, FPS, $MMORG$, etc. have been developed to test the routing algorithms defined in $DiVES$. Fig.4 compares the amount of *remote traffic*, i.e. *information exchanged among brokers* generated by the execution of a simple $MMORG$, with respect to the different routings. Simple routing is based on the entity based approach and includes the PAI optimization to reduce the amount of generated traffic. The results show that the simple routing and the advertisements based routing generate the smaller amount of traffic. Cell based routing generates a larger amount of traffic because of the useless notifications. Filter merging based routing generate a reasonable amount of traffic, when the number of players is less than 32. For larger number of players, due to filter merging, a large amount of traffic is generated. The other measure we have considered is the size of the routing table, since it determines the time spent by the brokers to perform routing. Our experiments show that the cell and merge based routing largely reduce this size. The minimum size is obtained by advertisement based routing, because in this case a subscription is sent to a broker only if that broker has emitted an advertisement that satisfies the notification. Our experiments suggest that the best compromise between the amount of remote traffic and the size of the routing tables is obtained by advertisement based routing.

We are currently extending $DiVES$ to include further features to support DVEs. For instance, we are planning to evaluate different alternative algorithms for event ordering and to study a dead reckoning algorithm to further reduce the message traffic on the network. We plan to integrate

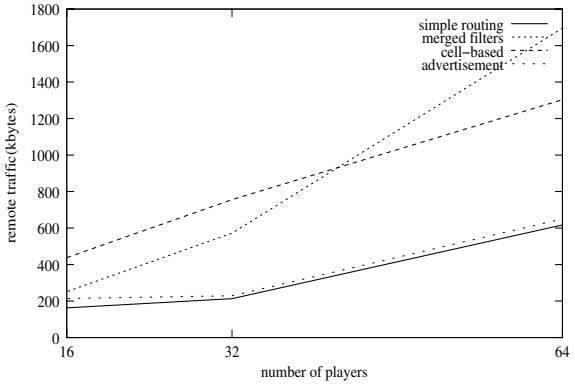

Figure 4. Routing Comparison

cheating detection tools in our environment. Finally, we are currently porting $DiVES$ on a wide area network.

References

[1] A.Bonotti. Un supporto per la implementazione di giochi multiplayer su rete. M.Thesis, Uni. Pisa, 2004.

[2] A.Carzaniga, D.Rosenblum, and A.Wolf. Design and evaluation of a wide-area event notification service. *ACM transactions on Computer Systems*, 19(3):332383, August 2001.

[3] A.Bharambe, S.Rao, and S.Seshan. Mercury: A scalable publish-subscribe system for internet games. Bruanschweig, Germany, April 16 - 17, 2002.

[4] G.Morgan and F.Lu. Predictive interest management an approach to managing message dissemination for distributed virtual environments. In *Richmedia2003)*, October 16th - 17th, 2003. Lausanne, Switzerland.

[5] E.Lety and T.Turletti. Issues in designing a communication architecture for large-scale virtual environments. In *Proc. NGC'99)*, November 1999. Pisa, Italy.

[6] P.Eugster, P.Felber, R.Guerraoui, and A.Kermarrec. The many faces of publish subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[7] S.Fiedler, M.Wallner, and M.Weber. A communication architecture for massive multiplayer gmaes. In *NetGmaes 2002*. ACM, April 2002.

[8] Li Zou, M.Ammar, and C.Diot. An evaluation of grouping techniques for state dissemination in networked multi-user games. In *Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2001. Cincinnati, Ohio.