

# Rokkatan: Scaling an RTS Game Design to the Massively Multiplayer Realm

JENS MÜLLER AND SERGEI GORLATCH  
University of Münster

---

While massively multiplayer online role-playing games (MMORPG) involve large numbers of simultaneous players, two other popular game classes – first-person shooter (FPS) and real-time strategy (RTS) games - are still only rarely considered for massively multiplayer gaming. A main technical problem for scaling these genres to be massively multiplayer is the absence of a suitable scalable multiserver networking approach. While the commonly used zoning concept performs well for an MMORPG, it is barely suitable for RTS and FPS games. As a scalable networking alternative for these genres, this article summarizes our work on the proxy-server architecture, which uses multiple servers for a single game session and implements a full replication of the game state at all proxies. We present our work on the game *Rokkatan*, our online evaluator game which enables massively multiplayer real-time strategy gaming using our proxy-server network architecture. We discuss the implementation of Rokkatan and analyze the distributed computation of the proxy-server approach by integrating an analytical scalability model into Rokkatan. Our experimental game sessions demonstrate high scalability of Rokkatan, which allows several hundreds of users to participate in a single, fast-paced game session.

Categories and Subject Descriptors: H.4.3 [Information Systems Applications]: Communication Applications

General Terms: Design, Performance, Measurement

Additional Key Words and Phrases: Computer games, scalability, massively multiplayer

---

## 1. INTRODUCTION

*Massively multiplayer online role playing games (MMORPG)* have recently become one of the major gaming applications played over the Internet. In these games, thousands of users play an adventurer in a huge virtual world. However, other multiplayer game styles like *first-person shooter games (FPS)* or *real-time strategy games (RTS)*, have rarely been discussed for massively multiplayer gaming, because the commonly used client-server and peer-to-peer network architectures are not able to support a high number of users. The scalable multiserver zoning concept, commonly used in MMORPG, is unfortunately not really suitable for RTS and FPS games, as the game worlds are too small to be properly subdivided into different zones. Furthermore, it is likely that users of an RTS or FPS game eventually come together for a big fight, thus congesting a single zone. Therefore, novel concepts for scalable networking in FPS and RTS games are required. In order to use a novel network architecture in industrial game development, the topology has to be evaluated in detail. The best way to accomplish such an evaluation is to actually implement a sophisticated game using the architecture.

In this article, we summarize our concepts of a *proxy-server topology* and *game scalability model* for multiplayer online games and present our recent work on the game *Rokkatan*, which has been implemented to evaluate these novel concepts. Furthermore, Rokkatan serves as a case study for the design and implementation of an RTS game suitable for several hundreds of users.

The *proxy-server topology* 00 has been presented as our scalable alternative to the commonly used client-server and peer-to-peer approaches. For a single game session, this architecture replicates the game state at several servers (proxies), which therefore can provide access to the game for an arbitrary client, regardless of the position of the client's *avatar* in the game world. The replicated game state is synchronized between the proxy-servers using the model of *eventual consistency* 0, which introduces only a small overhead in network communication, as compared to stronger consistency models, and therefore makes the architecture feasible for fast-paced action games requiring a high level of responsiveness.

---

Authors' addresses: J. Muller and S. Gorlatch, Dept. of Computer Science, Univ. of Münster, Einsteinstr.62,D-48149 Münster, Germany; email: [jmueller@math.uni-muenster.de](mailto:jmueller@math.uni-muenster.de) and [gorlatch@math.uni-muenster.de](mailto:gorlatch@math.uni-muenster.de).

Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Permission may be requested from the Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036, USA, fax:+1(212) 869-0481, [permissions@acm.org](mailto:permissions@acm.org)

© 2006 ACM 1544-3574/06/0700-ART6E \$5.00

We developed the analytical *game scalability model (GSM)* [0] for evaluating the scalability of our proxy approach. This model provides a detailed forecast of the required computation time and communication bandwidth in the client-server, peer-to-peer, and proxy-server topology. Using the model, we demonstrated that our replication concept in combination with the proxy-server approach performs better in terms of scalability than the common client-server and peer-to-peer approaches, especially for FPS and RTS game designs.

The development of Rokkatan pursued three major goals:

- *Evaluation of our replication concept and the proxy-server topology:* Rokkatan serves as a detailed case-study of how to design and implement a sophisticated and scalable real-time game using the proxy-server approach. In particular, our goal was to evaluate the usage of the *eventual consistency* model for synchronising the game state replications and to detect possible problems in providing the required responsiveness for a fast-paced real-time game.
- *Incorporation of the game scalability model:* The GSM provides the possibility for incorporation in a particular game implementation by measuring execution times for several basic tasks that have to be accomplished during a running game session. Based on these times, the scalability model provides a prediction of maximum player numbers without exhaustive tests. Such a mechanism, integrated into a real game implementation, should help to determine required server capabilities and provide hints for an efficient setup of servers and session rules.
- *Conceptual evaluation of a massively multiplayer RTS game design:* Current large-scale game designs concentrate on massively multiplayer online role playing games (MMORPG) like *Everquest* or *World of Warcraft*, which provide a huge persistent world for the users to adventure in. However, other game genres like first-person shooter or real-time strategy games have so far rarely been adapted to massively multiplayer sessions. With Rokkatan, we propose a possible game design which extends current real-time strategy gaming to the massively multiplayer realm.

The remainder of this article is organized as follows: In Section 2, we briefly describe the game elements of Rokkatan. We introduce our replication approach and the proxy server topology in Section 3 and compare this concept to the well-known zoning approach, already used for MMORPG. We discuss the actual use of our architecture in Rokkatan in Section 4. The incorporation of the game scalability model (GSM) into Rokkatan and its forecast of the maximum possible number of players are presented in Section 5. Section 6 describes our experiments with Rokkatan and compares the actual measured player numbers to the predicted values. We compare the technical concepts and implementation of Rokkatan to related work and summarize our results in Section 7.

## 2. ROKKATAN: THE GAME

Rokkatan has been developed with two goals in mind: First, the game serves as a technical evaluator for our replication concept and the proxy-server topology and demonstrates the scalability (in terms of increased player numbers) of these approaches. Second, Rokkatan constitutes a game design demonstrator for a massively multiplayer real-time strategy game. There is only very limited design experience for games in this area; Rokkatan shows one feasible way to provide a strategy-oriented, fast-paced massively multiplayer gaming experience. In this section, we describe the game design elements of Rokkatan in detail.

In Rokkatan, each user has control over a single unit, his *avatar*, and belongs to a particular team. The number of teams playing in a single game session is arbitrarily set up upon session creation. After connecting to a running game session, the user chooses a team to join and the class and name of his avatar. Currently, there are two classes implemented in the game, the *warrior*, fighting within close range, and the *archer* who can shoot arrows at distant enemies.

Users of the same team coordinate themselves and move around to occupy *flags* scattered in the game environment. For each flag currently occupied, a team periodically gains *score points*. Each team has an initial amount of score points and the team with most points will win the session after a certain time of play. Therefore, avatars of opposite teams have to fight for supremacy of flags. Such real-time fights, as depicted in the screenshots (Figures 1 and 2) for a small duel and a large battle, play a major role in Rokkatan. Each avatar has a particular number of health points, which decrease when the avatar is hit by an enemy warrior or archer. If the health points of an avatar drop to zero, then he is “dead” for a short time, after which he respawns at the starting area of his team. Additionally, the team score points of an avatar that temporarily lost his life are decremented by one.



Fig. 1. A small duel in Rokkatan.

The game style of Rokkatan is comparable to RTS games like *Command and Conquer* or *Warcraft III*, with the main difference, that a few users do not control large groups of avatars, but each avatar in the game is controlled by a single user. Therefore, it is necessary for all users in a single team to coordinate their actions. Some avatars guard already occupied flags, while others try to conquer new areas of the game environment. The goal of occupying flags is comparable to tactical FPS games like *Battlefield 1942*, in which several flag points have to be captured in order to win the game session.

A Rokkatan game session takes place in a particular game environment, the game *map*, which is described as an easily editable text file. At different locations in the map, *potions* are available, which can be picked up and carried by avatars and used later on. If the user decides to use such a potion, his avatar immediately regains health points, which makes potions valuable when fighting enemies.



Fig. 2. Massive encounter in Rokkatan.

## 2.1 Avatar Control

There are two main aspects in controlling an avatar in Rokkatan: moving and interacting. Both of these controlling aspects can be designed to be either *direct* or *indirect*.

**2.1.1 Indirect Movement.** In Rokkatan, the user controls the movement of his avatar indirectly by pointing and clicking the desired position with the mouse. The game client then calculates the shortest path from the current to the desired position, automatically taking into account obstacles like forest or water regions. This path computation is done with the A\* algorithm <sup>0</sup>, which will be discussed in Section 4.1 in more detail. This method of movement control, as usually implemented in RTS games, is indirect because the actual movement is performed by the game itself. The alternative would be to directly control each single step of the avatar by constantly pressing a single or several keys representing the direction to move in, as usually realized in FPS games.

**2.1.2 Direct Interactions.** There are two main kinds of user interactions in Rokkatan: interacting with the game environment (picking up potions and occupying flags) and interacting with other avatars (attacking an enemy). Both kinds of interaction are performed in a direct fashion by the user. If a user clicks with the mouse onto a potion to be picked up or onto an avatar to be attacked, then the game internally handles these user actions independently of the target object. The target of an interaction is solely based on the static position of the user click and not on the object beneath the mouse cursor. Therefore, a warrior always swings his weapon towards a static position and misses if the opponent already moved away. This is even more obvious when attacking with an archer: The archer shoots an arrow towards the clicked position, regardless of whether the opponent may have already moved away. This direct interaction control is usually implemented in FPS games as well, in which a static flight path of a bullet or missile is determined depending on the current point of view.

We implemented the direct interaction control in Rokkatan in order to achieve a faster-paced game style than in usual RTS or RPG games. Similarly to an FPS game, users constantly have to track and aim at their opponents and are able to dodge approaching arrows by quickly moving out of the arrow's way. With indirect arrow shooting, a single arrow would be “preprogrammed” to hit a marked target and automatically follow the movement of the target avatar instead of just hitting a static target position.

The avatar and ground graphics for Rokkatan were taken from *Reiner's tile sets* [Prokein] a free library of game graphics. A more detailed description of Rokkatan with many screenshots is available at the project's website <sup>0</sup>

## 3. GAME WORLD REPLICATION AND THE PROXY-SERVER TOPOLOGY

Several game servers are required to scale to make an online game massively multiplayer. Then, the real-time state computations of the game session have to be processed in parallel on all participating server hosts. In order to allow this parallel processing, the *game entities* (*player avatars, nonplayer characters (NPC), items*), for which a new state has to be calculated frequently, must be distributed among the server hosts. In the current *zoning* approach for MMORPG, the game world is split up into several zones; entities in each zone can be processed by a single server independently of other zones. Zoning, however, does not work well for FPS or RTS games due to load imbalance and zone overlap issues. For our proxy-server architecture <sup>0</sup>, we designed a *replication*

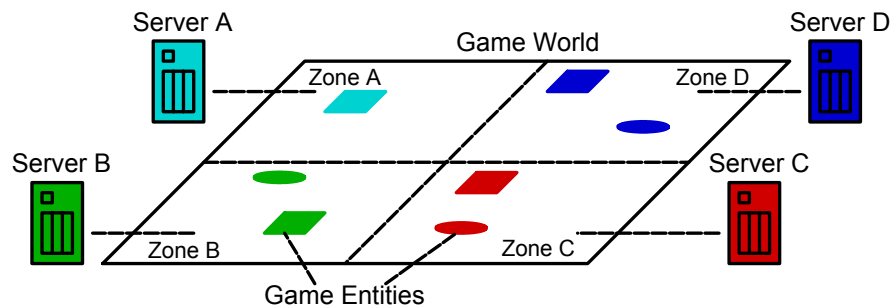


Fig. 3. Multiserver zoning

approach for distributing game entities among the participating servers, which is more suitable for more direct and fast-action and strategy games in smaller game worlds. In the following sections, we will discuss and compare the common zoning and our replication approach and then present the proxy-server architecture used in Rokkatan.

### 3.1 Current Approach: Multserver Zoning

In this well-known approach commonly used in MMORPG, a huge game world is partitioned into several independent segments or zones. For each zone, a single dedicated server is assigned to perform the state computations. Figure 3 depicts an example of a game world segmented into four zones, such that a server computes the new state only for the game entities (avatars, non-player characters, items, etc.) residing in its zone. A client has to connect to the server assigned to the particular zone in which the client's avatar resides. If the user decides to move his avatar into a different zone, then the game client will have to establish a new connection to the server responsible for the new zone.

While zoning works well for MMORPG, it is not suitable for FPS and RTS games. The FPS and RTS game worlds are usually much smaller than those in an MMORPG. Because each zone requires a minimum size there can only be a few zones (small zones require a lot of server changes when moving around, and the resulting “loading screens” annoy users and disrupt their immersion in the game). Overlapping zones would make server changes transparent for users, but this approach decreases the scalability of the zoning concept for FPS games, since large overlaps are required due to the long range of view in these games. Another major problem is the clustering of avatars in a particular area, which overloads the single server responsible for the corresponding zone. Especially in shooter-games, where users tend to go “where the action is,” a large fight will attract a growing number of users to participate while other zones are empty. Therefore, a clustering of avatars crowding a single server of that zone is much more likely in FPS games than in MMORPG. Owing to these problems in adopting multiserver zoning for FPS and RTS games, we developed an alternative multiserver replication especially for these classes of games.

### 3.2 A Novel Approach: Multiserver Replication

In our multiserver replication approach, each server holds a copy of the complete state of the game session. The responsibility is equally distributed among the servers for all game entities (user avatars, nonplayer characters, items), for which a new state has to be calculated at each tick. If a server is responsible for a particular game entity, then this entity will be an *active entity* for this server. For all other participating servers, this entity is a *shadow entity*. Figure 4 depicts an example of a game session replicated at three servers. The replications can be imagined as layers; the active entities of each server in the figure are filled, while the shadow entities that other servers are responsible for are only outlined. In the computation of a new state, each server calculates the state only for its active entities. After this calculation, each server sends the new state to the other servers, which in turn update the state of their corresponding shadow entity. This approach scales because the update of a shadow entity using the message received from the responsible server can be applied much faster than a complete calculation of this new state would take.

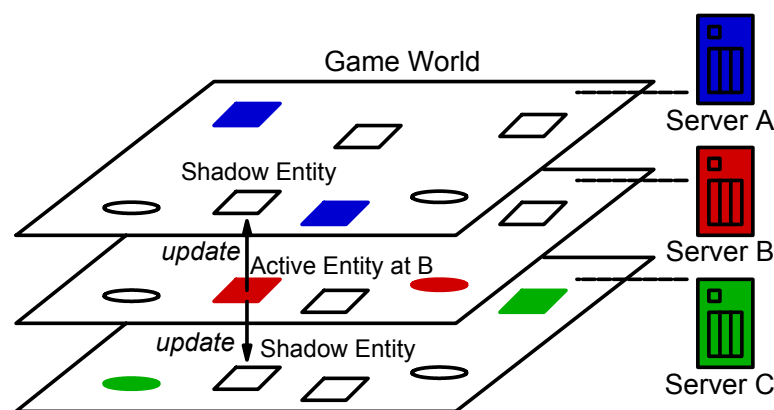


Fig. 4. Multiserver replication.

The replication approach is more feasible for FPS and RTS games played in smaller game worlds than the zoning concept, since clients are not required to change the server connection during the game. Each server can support arbitrary clients independently of the positions of avatars in the game because each server holds a copy of the complete game state. In the case of a clustering of avatars in a particular area of the game world, the required state calculations are still distributed among the servers; this situation in the zoning approach would lead to a single server being congested while other servers are idle. We developed and implemented the *proxy-server topology* as an operational architecture incorporating this replication approach, as described in the next section.

### 3.3 The Proxy-Server Topology: Concept for Rokkatan

The generic concept of the proxy-server architecture for real-time computer games in combination with the replication approach is presented in 0. Rokkatan serves as an in-depth evaluation of the concept and as a case study in the development of a sophisticated computer game on top of this approach. The general architecture, of which an example setup is shown in Figure 5, consists of several proxy-servers for a single game session. The proxy-server topology directly incorporates the replication concept, such that each server has a full copy of the complete game state. Because the complete game state is fully replicated at each proxy, game clients can connect to an arbitrary proxy in order to participate in the session. A game client should choose the particular proxy that provides the best communication quality, i.e., lowest latency and lowest packet drop rate. Servers are denoted “proxies” because clients should connect to servers which are located “near” them w.r.t. communication latency.

### 3.4 Eventual Consistency Model for the Replicated Game State

Because each proxy server stores the complete game state of a session, there has to be a mechanism which ensures a certain model of consistency in the replicated game state copies. The active/shadow entity concept implements the *eventual consistency* model 0: Only a single proxy is allowed to alter a specific part of the game state (active entities for this proxy), which prevents concurrent write access to the same entity at different servers. When a proxy changes the state of an active entity, all remote proxies are informed and eventually update their corresponding shadow copies of the entity accordingly. With this distribution of authority, no explicit mutual exclusion mechanism for write access to data is required, resulting in low communicational overhead to ensure consistency and fast processing of user actions.

Two kinds of game entities have to be treated differently In order to find a reasonable distribution of the active state of entities (i.e., the write access authority) among all proxy servers in Rokkatan:

- *User-dependent entities*: These are all the entities inside the general game state that represent the status of the users' avatars in the game. Examples are the avatar entity itself, an inventory entity, or weapon entities “owned” by an avatar. In order to provide maximum responsiveness for the processing of user actions, each proxy is made authoritative for the avatar data of its directly connected clients.

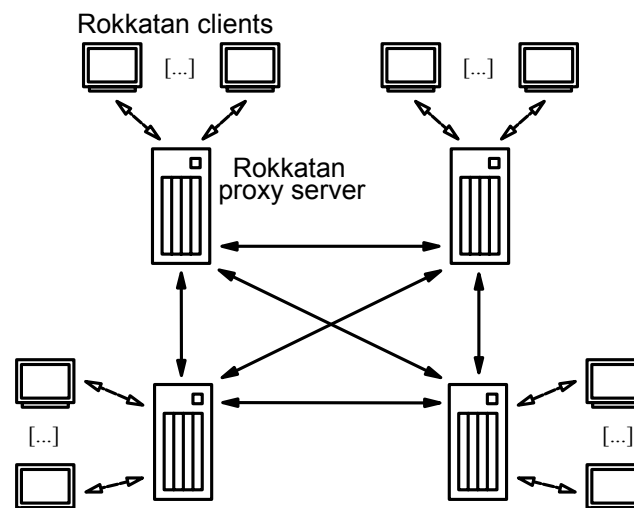


Fig. 5. The proxy-server architecture.

- *User-independent entities:* These game entities are not directly associated with a particular user. In Rokkatan, these are the potions, flags, team score points, and remaining game time. The authority for these entities is distributed at the start of a game session, such that each participating proxy is responsible for nearly the same amount of user-independent entities.

### 3.5 Processing User Actions

As discussed in Section 2.1, there are two main types of user actions: movement commands and interaction commands. A movement command can be processed directly at the proxy to which a client is connected, since this action only affects the state of the user's avatar, and therefore only a single active entity at the corresponding proxy. This proxy is the only process allowed to alter the data for a particular client, such that the position change of the avatar resulting from a movement command can immediately be performed and acknowledged. Additionally, the proxy communicates this state change of the active avatar entity to all the other servers that update their local shadow entity replicas of the remote avatar.

Processing of interactions, however, cannot be done solely by the local proxy of a particular client. The interaction command can affect either other avatars, e.g., by attacking an opponent, or the general game environment, e.g., by picking up a potion. Therefore, the state of the target game entity such as an avatar or a potion has to be changed. In the general case, this target entity will only be a shadow entity for the proxy that receives the interaction command from a local client. Therefore, the interaction command has to be forwarded to the proxy that owns the active entity of the target and is able to evaluate the interaction. Figure 6 depicts an example of user interaction processing in the proxy-server topology as implemented in Rokkatan.

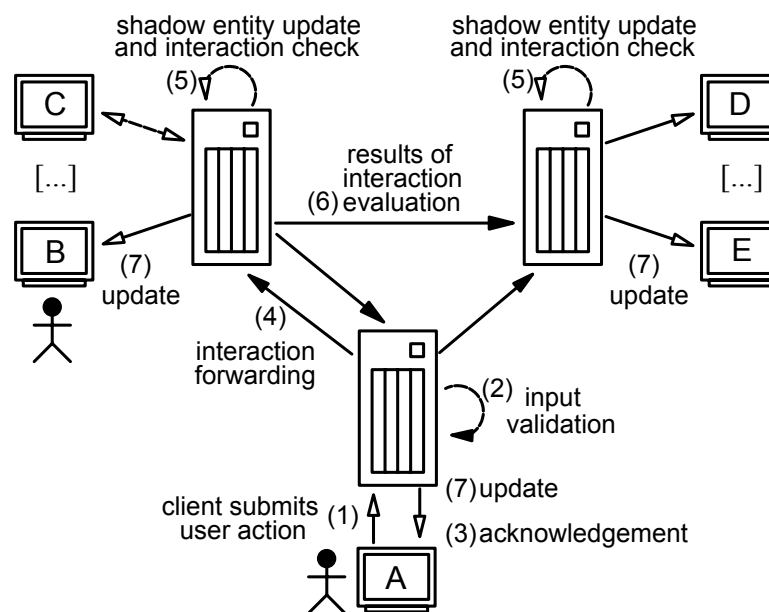


Fig. 6. Interaction processing.

In Figure 6, the user at client A issues an interaction affecting the avatar of client B, e.g., attacking the position of B's avatar in the game environment. In step (1), client A submits the user interaction command to its proxy server, which validates the received input in step (2). If the validation is successful, i.e., the state of the avatar allows the attack, then the proxy sends an acknowledgement back to client A (step (3)), and forwards the interaction to all other participating servers in step (4). The other proxies update their local game state, i.e., they update the shadow entity of the attacking client A in step (5). Additionally, each remote proxy checks whether an active entity that it is responsible for was affected by the attack of avatar A. In this example, the avatar of client B is hit by the attack. The local proxy of client B updates the active entity state of its avatar by decrementing health points and informs all other proxies about this state change (step (6)). Finally, in step (7), all proxies inform local clients that are directly affected by the interaction (clients A and B). Additionally, all clients whose avatars are located near the interacting

avatars are notified about the interaction. For example, users at the clients D and E observe the interaction and the proxies inform the clients about it.

#### 4. ROKKATAN IMPLEMENTATION

Rokkatan is implemented in C++ and uses the *Kyra sprite engine 0* and the *Simple Directmedia Layer (SDL)* for client graphics and sound. The game communication is based on our *Game Proxy Architecture (GPA)* library which we developed to make the proxy-server approach easy to use and convenient for game developers. The library provides a simple API for clients and proxy servers to send and receive game messages at different levels of reliability. For inter-proxy communication, game messages are sent using IP-Multicast. If proxies are not able to participate in the IP-Multicast group, the GPA automatically falls back to sending messages via unicast. This way, scalable multicast communication is used whenever possible and the unicast fallback ensures general functionality of game sessions in networks not supporting multicast.

While the replication concept and the proxy-server approach determine the general design of the Rokkatan implementation, some Rokkatan-specific issues had to be addressed in order to ensure the scalability of the game. For these particular problems, we developed solutions and implemented them directly into the Rokkatan application, as described in the next sections. Although developed for Rokkatan, these solutions can be reused in other games via the replication concept; they provide an extension of the generic proxy architecture approach.

##### 4.1 Efficient Processing of Indirect Movement

As described in Section 2.1.1, a user clicks onto the intended target point to move his avatar. The game then computes a path from the current to the target position using the A\* algorithm [0] and automatically moves the avatar along this path. In general, there are two straightforward implementation solutions: either the proxy responsible for the client computes the path or the client computes and transfers the path to the server. The first solution requires a lot of computation time at the proxy for all connected users, while the second solution requires a relatively high amount of bandwidth to transfer complete paths from the clients to a proxy.

For Rokkatan, we implemented a mixture of both approaches mentioned above. The movement path is computed at the client in order to free the proxy from the required calculation. However, clients only transmit nodes of the movement path that are actually required for the current game processing. Additionally, no data will be transmitted unnecessarily if users change their minds and issue a new target during an ongoing movement. The proxy requires at least two nodes of the path that have not been reached yet; otherwise, there would be a notable pause after a node is reached, because of the communication latency in transmitting the next node from the client.

Figure 7 depicts an example of the path transfer mechanism in Rokkatan. The complete movement path from the initial to the target position has been calculated at the client (left-hand side), after which the first two nodes of the movement path, w1 and w2, are sent to the server (right-hand side). The server moves the avatar from node to node and notifies the client regularly about the current position. If a node is reached, the server immediately continues the movement because the next node is known. In the figure, the avatar reaches w2 and the server continues the movement to the node w3. Upon reaching w2, the client sends the next node w4 to ensure a continuous seamless movement until w3 is reached.

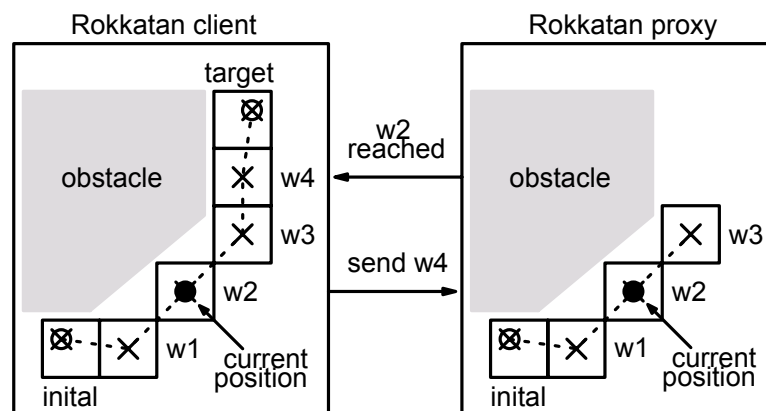


Fig. 7. Successive transmission of a movement path.

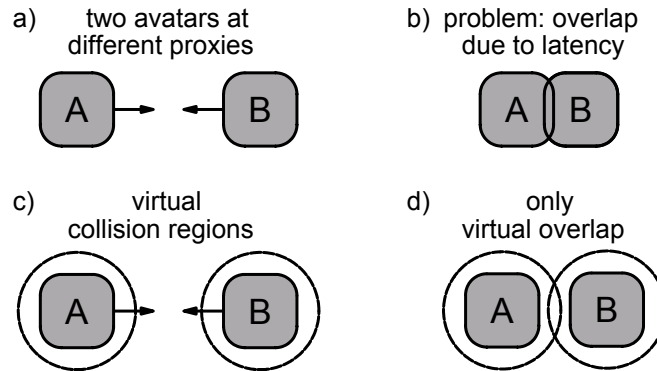


Fig. 8. Handling collisions.

#### 4.2 Collisions Between Avatars

A problematic situation occurs when one avatar collides with another maintained at a remote proxy. In this case, the shadow entity data of the remote avatar might be outdated, since the remote proxy moved that particular avatar, but the message with the updated position is delayed in the network. In such a situation, depicted in Figure 8, both proxies move their active avatars towards each other (a), and do not detect the resulting collision (b), because the update of the remote shadow avatar has not yet arrived. After the updates of the remote movement are received at proxies, the avatars' dimensions overlap, which is an incorrect game situation.

In order to avoid such overlapping avatars, Rokkatan extends the general proxy-server architecture with virtual collision areas for avatar entities (c). All inter-avatar collision checks are performed using these virtual areas to compensate for delayed updates for the shadow avatars of other proxies (d).

The radius of the virtual collision regions has to be adjusted to the maximum inter-proxy latency at which the game is able to run flawlessly. The proxies update their game states at regular intervals, described in detail in Section 5. In the current Rokkatan settings, the game state is updated every 40 ms, i.e., at a rate of 25 updates per second. For a maximum assumed inter-proxy latency of 100 ms and an avatar movement speed of 120 distance units ( $du$ ) per second, the virtual collision area has to extend the actual dimension of avatars by

$$virt.extension = \lceil (2 \cdot up\_interval + latency) \cdot avatar\_vel \rceil = \left\lceil (0.08s + 0.1s) \cdot 120 \frac{du}{s} \right\rceil = 17du$$

for the worst case. This virtual extension is independent of the actual avatar size. In order to prevent the possible user impression that avatars cannot get close due to virtual collision, the graphical avatar size can be increased. Additionally, the range of close combat fighter attacks with swords or maces must be large enough to actually hit the enemy avatar graphics. Otherwise, attacks would only hit the air between the opponents, which would result in nonrealistic game visuals.

#### 4.3 Application-Specific Mutual Exclusion

We use the virtual collision regions to implement a Rokkatan-specific mutual exclusion for the write access to entities collectible by users, such as items, potions, etc. In the general case, these user-independent entities must be assigned to particular proxies as active entities for write access. If a user issues the command to pick up an item, the client's proxy has to forward the action to all other proxies, including the server authoritative for the targeted item. This proxy removes the item from the map and informs all other servers about the successful pickup. The originating proxy finally informs the client about the new content of his avatar's inventory.

Although this general concept of active/shadow entities is feasible, we developed a solution specific to Rokkatan, which provides better responsiveness for pickup action and requires less communication. In this solution, the game design is required to allow avatars only to pick an item if the avatar's dimension overlaps the item's boundaries, i.e., the avatar has to be at least partially "standing on" an item. The virtual collision regions prevent two avatars from standing on an item at the same time because the item's size is smaller than the virtual collision extension. This way, only one avatar at a time can stand on an item and pick it up immediately. If a pickup action is

received from a local client, only the proxy has to check whether the boundaries of the avatar and the target item overlap. If so, the server immediately removes the item from the map, updates the inventory, and sends a notification to the client, which results in much faster execution of item pickups. Additionally, the proxy notifies all other servers about the removal of the item. This way there is no active/shadow entity distinction of these item entities in Rokkatan, every proxy can change the state of these particular entities because mutual exclusion is guaranteed by the game-specific mechanics of how items are picked up. The mechanics of picking up items by running over them is implemented in a lot of games, especially in the FPS genre. This optimized handling of item pickups can be applied to any game implemented on top of the replication approach, and provides an improvement in terms of responsibility and bandwidth usage not only for Rokkatan, but for a major class of games.

#### 4.4 Proxy Failure

A single proxy server is a central point of failure in a running game session. However, the game state data is completely replicated at all proxies, and therefore not lost due to a single proxy failure. This way, only the authoritative instance for some parts of the game state is lost when a proxy server fails. Additionally, the connection to the session for the local clients of the crashed proxy server is lost. In order to face the first problem, we implemented a redistribution of the active state of entities in case of a server failure.

If a particular proxy is disconnected, all the entities which were active on the crashed proxy are marked as being shadows on all the remaining proxies; no server is allowed to change the state of these entities. In order to recover from this situation, the active state for user-independent entities (potions, flags, or team score points) of the failed host is redistributed among all remaining proxies. So the game session remains playable for all clients but those who were directly connected to the failed proxy. In a future version of the GPA library we plan to automatically and transparently reconnect such clients to another proxy.

### 5. INCORPORATING THE GSM INTO ROKKATAN

In recent work, we developed the *game scalability model 0* in order to analytically compare the scalability characteristics of the client-server, peer-to-peer, and proxy-server approach for different game designs. We found out that the proxy approach outperforms both other concepts in terms of scalability for fast-paced game designs which require high responsiveness and short update intervals.

Besides the generic topology comparison, the GSM provides the possibility of being incorporated into a particular game implementation, with the aim of forecasting the maximum number of participating players in a single game session. After a brief description of the model, we show how it is used in Rokkatan to analyze the use of computation and communication resources.

#### 5.1 The Game Scalability Model

Real-time computer games are soft real-time systems that incorporate certain time deadlines during the distributed processing of a game session. The processing, i.e., the calculation of a new overall game state depending on the previous state, the advance in time, and the user inputs issued, is expected to be finished before the deadline. If there are too many user actions to process in the calculation of a new state, the processing at the server eventually becomes congested and violates the real-time constraints of the game application. In Rokkatan, the *tickrate* is directly given at session setup and indicates the number of updates per second. Several experiments showed that a value of 25 updates per second is a good compromise for the trade-off between game responsiveness and computational load in Rokkatan. Moreover, this update rate matches the usual tickrate values of fast-paced FPS games, which range from 20 to 30 updates per second.

Additionally, the bandwidth required for the communication of the state updates has to be considered as well. If the server computes the new state timely, but afterwards is not capable of communicating the results in time, then the real-time constraints of the game are violated as well.

To provide an estimate of the maximum number of players, the GSM subdivides the computation and communication of a new state in several basic tasks. A session using the proxy-server topology is considered to consist of  $l$  proxy servers, each serving  $k$  directly connected clients. Therefore, a total of  $n = k \cdot l$  clients are participating in the game session. Furthermore, there are  $m$  user-independent game entities like potions or flags that are equally distributed among all proxies for authority, such that each proxy has to take care of  $s = m/l$  active entities. Additionally, the size of the various network messages is obtained by inspecting the data sizes in the implementation.

In the following, we discuss the five main tasks of the GSM for computing a new game state at a proxy server. These tasks will be mapped to the actual implementation details of Rokkatan (like message sizes obtained from source code inspection) and several computation time parameters will be introduced. Then, actual values obtained from measurements will be discussed and used for overall prediction of player numbers in Section 6. During a single game tick, a proxy has to compute the following five steps:

- (1) *Receiving, validating, and processing local client actions.* The time required for processing a single client's action at a Rokkatan proxy is the time parameter  $t_{ca}(n, m)$  (client action). For interactions, this time depends on the total number  $n$  of participating clients and the total number  $m$  of potions and flags, since the proxy has to determine the target of the interaction. The maximum size of a single message from a Rokkatan client is  $d_{cin} = 48bytes$  (client in).
- (2) Each proxy receives up to  $n - k$  remote client actions and  $n - s$  state updates of user-independent entities from other proxy servers for a single state update. The processing of such an update of a local shadow entity takes a constant time of  $t_{rsu}$  (remote state update). A message for a single update of an entity received from remote proxies has a maximum size  $d_{rsu} = 36bytes$ .
- (3) *Each Rokkatan proxy has to update all its active entities.* For avatars, the movement to the next path target has to be continued. Additionally, the local flags, team scores, and potions are updated. This way, a total of  $k + s$  active entities are possibly updated; each single calculation is assumed to require a time of  $t_{se}(n, m)$  (server entity) to be computed. The time depends on the total amount of entities. For moving an avatar in a crowded area, for example, several other avatar entities (active and shadow) have to be tested as to whether they collide with the new position of the updated avatar.
- (4) *Each proxy sends the new state  $S_{i+1}$  to clients.* The proxy server filters the state information for its  $k$  clients, such that each client only receives new state information about entities that reside in the avatar's area of view. This interest management saves communication bandwidth and prevents cheating [0] by using a hacked client that displays information about a larger area than intended. Rokkatan implements the *dead reckoning* concept [0], such that only information about avatars that changed their direction is sent. The identifying and sending of updated entities for a single client is the computation time parameter  $t_{fc}(n, m)$  (filtering client) in Rokkatan. This calculation time depends heavily on the current gaming situation for the particular client: If the avatar moves solely in an area of the map where no other avatars are located, then there is little information to determine and send. Otherwise, if the avatar participates in a massive battle involving a large number of other avatars on the screen, the calculation time for filtering is much higher. The resulting data parameter  $d_{cout}(n, m)$  of data sent to a single client depends on the number of visible avatars as well.
- (5) The updated state information of active entities of a proxy has to be transmitted to  $l - 1$  proxy servers, such that these remote proxies can update their shadow entity copies. In the worst case that all local avatars and user-independent entities were updated in this tick, the new states of  $k$  local clients and  $s$  active server-controlled entities have to be sent, each with an amount of data of  $d_{rsu}$ . If no IP-multicast is available between proxies due to an Internet-wide setup, then the GPA library falls back to unicast, sending data to  $l - 1$  remote proxies, which requires a time of  $t_{sp}(k, s)$  (sending proxy) for each remote proxy.

Aggregating all computation tasks to be performed in a single state update, the overall calculation time of a particular proxy is

$$T_{proxy}(l, n, m) = k \cdot t_{ca}(n, m) + ((n - k) + (m - s)) \cdot t_{rsu} + (k + s) \cdot t_{se}(n, m) + k \cdot t_{fc}(n, m) + (l - 1) \cdot t_{sp}(k, s)$$

The amount of data a proxy receives in a single tick is

$$D_{proxy}^{in}(l, n, m) = k \cdot d_{cin} + (n - k) \cdot d_{rsu} + (m - s) \cdot d_{rsu} = k \cdot 48bytes + ((n - k) + (m - s)) \cdot 36bytes$$

while the overall data that has to be sent is determined by

$$D_{prxy}^{out}(l, n, m) = k \cdot d_{cout}(n, m) + (l-1) \cdot (k+s) \cdot d_{rsu} = k \cdot d_{cout}(n, m) + (l-1) \cdot (k+s) \cdot 36bytes .$$

These formulas can be used to provide an estimate of the maximum number of participating clients in an uncongested game session. With a tickrate of 25 updates per second, the maximum number of clients  $n_{max}$  in a particular session with  $l$  proxies and  $m$  entities (potions and flags) is given by

$$n_{max} = \{ \max n : T_{prxy}(l, n, m) < \frac{1}{25} s \wedge D_{prxy}^{in}(n) \cdot 25 < D_{max}^{in} \wedge D_{prxy}^{out}(n) \cdot 25 < D_{max}^{out} \}$$

for a single proxy. The bandwidth capabilities of the particular host the proxy is running on are reflected by the terms  $D_{max}^{in}$  and  $D_{max}^{out}$  for incoming and outgoing communication, respectively. We compare the maximum player numbers predicted by this analytical model with the experimental results in the next section.

## 6. SCALABILITY EXPERIMENTS

We ran numerous test sessions to verify the scalability of the actual Rokkatan implementation using the replication concept of the proxy-server architecture and to verify our analytical model. While we tested Rokkatan with various client connection types (modem, ISDN, DSL) in order to confirm the general functionality of Rokkatan under higher latencies, the scalability tests were conducted in the local area network of our department because a large number of hosts was required.

The Rokkatan client includes a special “bot” mode, which automatically participates in a game session. This client-side bot issues actions based on the current gaming situation and makes full use of all possible game interactions like moving, attacking, occupying flags, and picking up potions. It uses potions to recover health points and retrieves from fights when all stocked potions have been consumed. For a server, the bot-mode of a client is transparent and cannot be distinguished from a human user.

The experiments were conducted for two test maps of different sizes (64x64 and 128x128 ground tiles). The dimensions of both maps are comparable to those of commercial real-time strategy games like *Warcraft III*; it takes about 90 seconds for the smaller and 180 seconds for the larger test map to walk diagonally from the upper left to the lower right corner.

In order to provide a forecast of maximum player numbers, we measured the parameter values required by the GSM -  $t_{ca}(n, m)$ ,  $t_{rsu}$ ,  $t_{se}(n, m)$ ,  $t_{fc}(n, m)$  and  $t_{sp}(k, s)$  - for both game maps in several test runs. Our reference server host is a Pentium 4 1.7 GHz system with 640 MB RAM running Linux with kernel 2.6, (we have several

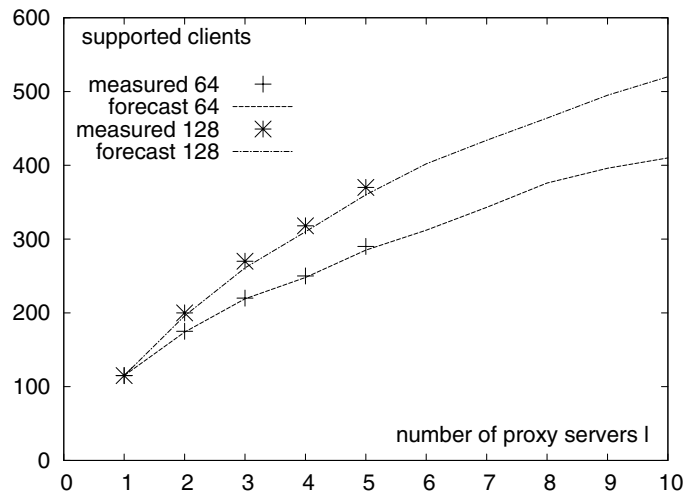


Fig. 9. Maximum number of players.

Table I. Estimated and Measured Bandwidth  
at a Single Proxy Server for the 64x64 Test Map

Session	estimated	measured	deviation
1 pr., 115 cl.	150.7 kB/s	152.0 kB/s	2 %
2 pr., 170 cl.	213.9 kB/s	210.3 kB/s	2 %
3 pr., 220 cl.	245.3 kB/s	235.5 kB/s	5 %
4 pr., 250 cl.	237.0 kB/s	222.7 kB/s	7 %
5 pr., 290 cl.	243.7 kB/s	242.5 kB/s	1 %
6 pr., 310 cl.	257.1 kB/s	-	-
8 pr., 375 cl.	284.8 kB/s	-	-
10 pr., 410 cl.	291.2 kB/s	-	-

systems available). Additionally, the amount of data for outgoing client communication  $d_{out}(n, m)$  was measured for both test maps.

For the tests of the setup with five proxy servers, a total of 25 computers were used. Five of them act as a proxy server, while the other hosts run the bot clients, several of which can be started on a single computer. For the experiments using both test maps, Figure 9 shows the maximum number of clients able to play before servers became congested. Additionally, the graph lines show the maximum client numbers predicted using the game scalability model.

The scalability of a game session depends on the size of the game map. In the smaller map, the density of avatars increases faster than in the larger map, which leads to congestion much earlier. The plots in Figure 9 demonstrate that the GSM forecasts are very near to the actually measured player numbers, with a maximum deviation of 5%. The model's forecasts for larger session setups with more than five proxy servers (which we were not able to measure experimentally) show that more than 500 players are expected to be able to participate in a large session of Rokkatan.

The forecasts and actual measurements for the average bandwidth at a single proxy server are shown in Table I for the smaller test map; again, our measurements were done for up to five servers. With a maximum deviation of 7%, the bandwidth predictions are very accurate as well. Due to the dead reckoning used in Rokkatan, the amount of data sent to a single client is quite low, ranging from about one to ten kBytes per second, depending on the game situation. However, the proxy servers fully synchronize their state at each tick in order to provide the required responsiveness for direct interactions. Overall, the bandwidth utilization at a single proxy is low enough to allow sessions with a large number of users when the servers are hosted at high capacity Internet connections.

## 7. RELATED WORK AND CONCLUSION

There has been a lot of work in the area of scalable network topologies dedicated to massively multiplayer gaming. Most of the architectures presented follow the well-known zoning approach. The authority for such zones, which are commonly used in MMORPG, is either assigned to single servers as in 0, or distributed dynamically in a decentralized way 0. However, due to the much smaller size of a Rokkatan map in comparison to an MMORPG environment, the zoning approach is not feasible. In the worst case, all avatars would be clustered within a single zone and the single server responsible would become highly congested. The proxy-server approach performs much better in a scenario with a high avatar density due to a lot of clustered avatars. Additionally, MMORPG always incorporate the indirect interaction paradigm, and can therefore be run at much lower tickrates than FPS games. On the other hand, the proxy-server approach does not aim at MMORPG with its huge game environments, but rather at FPS and RTS games with comparatively small maps. Rokkatan shows the feasibility of the proxy architecture to host game sessions for hundreds of users in a small game environment at a high responsiveness of 25 updates per second.

In the area of game design for other MMOG genres besides role-playing games, very little research has been done, although there are already commercial games of the FPS genre, suitable for a high number of participating players. Games like *Joint Operations* or *Soeldner* take place in a huge area and simulate small warfare, in which users of the same team have to coordinate to win the session. The single server approach used by these games limits the number of players, although the game design itself would support many more players in a session. In our opinion, the proxy approach is feasible for these fast-paced action games and would allow a much higher number of

users to fully exploit the game design suitable for a large-scale warfare. For the RTS game genre, there are of course other possible ways to extend conventional multiplayer game design to the massively multiplayer realm different from the Rokkatan design. Instead of incorporating direct interactions, a single avatar per user and a high tickrate as in Rokkatan; the RTS game *Shattered Galaxy*, for example, allows over 50 users to command a total of more than 500 units in a single game session with traditional, indirect interactions.

With the development of Rokkatan, we showed the scalability of our proxy-server architecture for game designs requiring high responsiveness. The behavior of the client bots in the experiments was sophisticated enough to make the test sessions comparable to human user sessions. There were always several large battles taking place, bots fought for supremacy of flags, used potions, and tried to save themselves when low on health points. This proves that, with a game map of adequate size, fluent and responsive game sessions involving several hundred users are possible in Rokkatan. The GSM model delivers very accurate forecasts of the use of computation and communication resources and was successfully integrated into the Rokkatan game. The special concepts implemented in Rokkatan, like the handling of proxy failures or the mutual exclusion for elements that can be picked up, can be reused in other game implementations via the proxy server concept.

Many extensions are possible for Rokkatan, for example additional avatar classes like mages or scouts. The concept of building a home base could be included by assigning certain roles to users, such that some players of the team are responsible for building the base or gathering materials. Additionally, there could be certain “commanders” without their own avatars, only giving commands to other players. Such a commander mode, already implemented in games like *Natural Selection* or *Battlefield 2*, would be a feasible way to organize huge teams in a massively multiplayer RTS like Rokkatan.

## REFERENCES

- CAI, W., XAVIER, P., TURNER, S. J., AND LEE, B.-S. 2002. A scalable architecture for supporting interactive games on the Internet. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation* (Washington, DC., May), 60-67.
- KNUTSSON, B., LU, H., XU, W., AND HOPKINS, B. 2004. Peer-to-peer support for massively multiplayer games. In *Proceedings of the IEEE Infocom 2004* (Hong Kong, March).
- LESTER, P. A\* pathfinding for beginners. <http://www.policyalmanac.org/games/aStarTutorial.htm>.
- MÜLLER, J., FISCHER, S., GORLATCH, S., AND MAUVE, M. 2004. A proxy server-network for real-time computer games. In *Proceedings of the Euro-Par 2004 Parallel Processing Conference* (Pisa, Aug.). LNCS 3149. 606-613.
- MÜLLER, J. AND GORLATCH, S. 2005. GSM: A game scalability model for multiplayer real-time games. In *Proceedings of the IEEE Infocom 2005* (Miami, Fla., March).
- PRITCHARD, M. 2000. How to hurt the hackers: The scoop on Internet cheating and how you can combat it. *Game Developer Magazine*.
- PROKEIN, R. Reiner's tilesets. <http://www.reinerstileset.4players.de:1059/englisch.htm/>.
- ROKKATAN WEBSITE. <http://pvs.uni-muenster.de/pvs/projects/rokkatan/>.
- SMED, J., KAUKORANTA, T., HAKONEN, H. 2001. Aspects of networking in multiplayer computer games. In *Proceedings of the International Conference on Application and Development of Computer Games in the 21st Century* (Hong Kong, Nov). 74-81.
- TANENBAUM, A. AND VAN STEEN, M. 2002. *Distributed Systems: Principles and Paradigms*. Prentice Hall, Englewood Cliffs, NJ.
- THOMASON, L. Kyra sprite engine. <http://www.grinninglizard.com/kyra/>.

Received June 2005; accepted March 2006