Max Lehn, Christian Groß, Tonio Triebel:
Benchmarking Peer-to-Peer Systems. In: Wolfgang Effelsberg, Ralf Steinmetz, Thorsten Strufe, vol. LNCS, no. 7847, chap. Peer-to-Peer Overlays for Online Games, p. 143-167, Springer, June 2013.
ISBN 978-3-642-38672-5.

Chapter 8
Peer-to-Peer Overlays for Online Games

Max Lehn, Christian Groß, Tonio Triebel

In the past decade, a number of researchers have focused their work on peer-to-peer technologies for networked multi-player games [10, 4, 2, 8, 3, 14]. Since the usage of such a peer-to-peer overlay has a direct impact on the quality of experience of a game, it becomes apparent that performance evaluation and comparison is an important issue. The major challenge results from the fact that each approach is tailored to a different purpose and has to be evaluated with an individual technique in mind that is specific to the overlay. Thus, it is not a trivial task to compare the performance of different solutions. In order to solve this problem, overlays must be separated into different classes of equivalent functionalities. As introduced in Chapters 2 and 3, a class can be defined by a common interface. Systems that implement this interface are comparable. As a concrete example, systems for interest management and game event dissemination are used in this chapter. A common interface, a representative scenario, a workload and metrics are defined, and an exemplary benchmark is implemented and executed, comparing three different overlays.

8.1 Interface Definition

The first step in the creation of a benchmark is the definition of a common interface. The functionality that should be evaluated must be defined, and a suitable set

Max Lehn
Technische Universtität Darmstadt, Databases and Distributed Systems Group, Darmstadt, Germany, e-mail: mlehn@dvs.tu-darmstadt.de

Christian Groß
Technische Universtität Darmstadt, Multimedia Communications Lab, Darmstadt, Germany, e-mail: chrgross@kom.tu-darmstadt.de

Tonio Triebel
Universität Mannheim, Mannheim, Germany, e-mail: triebel@informatik.uni-mannheim.de

of instructions must be chosen. Systems that are able to implement these instructions can then be benchmarked. For peer-to-peer gaming overlays, there is no such standardized functionality yet. However, looking at the network design of *massively multiplayer online games* (MMOGs), a first interface can be derived.

According to Fan et al. [6], the design of an MMOG can be categorized into six network issues: interest management, game event dissemination, NPC[1] host allocation, game state persistence, cheating mitigation and incentive mechanisms. All of them are relevant to most MMOGs. However, interest management and game event dissemination are of major importance for the quality of experience of a networked game, since they build the basic set of services. In addition, the major part of the network traffic is generated by them. Thus, this chapter focuses on an interface based on interest management and game event dissemination. These two aspects are commonly addressed as one issue by *information dissemination overlays* (IDO). In order to be able to define a formal interface we first take a closer look at the functionalities of each of the two.

Interest Management Every player has a different view of the game world, which defines the parts of the world she can see and the events she can perceive. Interest management decides which information is necessary to build such a personal view. An *area of interest* (AOI) defines the region from which a player needs to receive game event information. Typically, the AOI is centered at the player's position and bounded by his *vision range* (VR).

Game Event Dissemination The game event dissemination system has to ensure that each player receives all relevant game events within his AOI. Real-time games require low latencies in the event dissemination to keep the players' views up-to-date. Since the AOI is bound to game world positions, the dissemination systems are typically based on the proximity in the game world.

In order to choose a suitable set of instructions for the interface, it is necessary to consider the software architecture of a game. An exemplary architecture consists of a *game instance* containing the local game logic and a *network engine* managing the network communication. From the benchmarking point of view, the network engine implements the SUT, and the game instance applies the workload. The network engine regularly gets updates of the local player's state (particularly its position, i.e., its current game world coordinates) from the game instance and disseminates them to all interested players. Depending on the particular IDO, the position information is also used to build the overlay topology. The request and dissemination frequency is chosen by the network engine so that it can adapt the generated update traffic when necessary (e.g., in case of congestion). The network engine notifies the game instance about updates on neighboring players within the AOI and about their positions. Game actions other than player movements are pushed by the game instance to the network engine at the time when they occur. Those messages do not have a semantic meaning to the network engine; they have to be delivered reliably and without modification.

[1] NPC: non-player character

Figure 8.1 sketches the abstract view of the basic layers of a networked game. The player interacts (through a user interface) the with the game instance. The network engine synchronizes the game state using the underlying IP network.

Fig. 8.1: Schematic view of the layers of a networked game.

8.1.1 A Minimal Interface

A minimal interface for such systems could look as follows. Please note that, for simplicity reasons, only the most relevant subset of the actual functions is described here. Generally, there need to be functions for joining and leaving the network, as in any typical peer-to-peer system. Since some peer-to-peer overlays are structured based on the positions of the players in the virtual world, the join function transmits the starting position of the joining player:

```
void join(Position p);
void leave();
```

The core functionality of the basic interest management as described above looks as follows:

```
void setPosition(Position p);
List<Neighbor> getNeighbors();
void setAOISize(int radius);
```

`setPosition` updates the local player's position. `getNeighbors` returns the current set of neighbors within the player's AOI; each `Neighbor` provides at least the last known position of the corresponding player. Optionally, the local AOI size can be adjusted using `setAOISize`. Finally the game event dissemination needs only one additional function, which serves as a broadcast for arbitrary game data to all peers in the AOI:

```
void disseminate(Data d);
```

8.2 Non-functional Requirements

The most important non-functional requirements for network infrastructures for games in general are a high *responsiveness*, a high *view consistency*, good *fairness*, and low network *costs*. Responsiveness typically relates to a low update latency. View consistency means that all players get the same information of all parts of the game world that the respective player can see according to the game logic. A high responsiveness in terms of state updates of the game world elements generally leads to a high view consistency. A high view consistency among all players in turn is a positive factor for fairness. Fairness describes how uniformly both the gaming service quality (i.e., the aspects described above) and the costs are distributed among the players. Finally, the peers should have a low cost in terms of bandwidth utilization, processing power, memory, etc. Since in most cases bandwidth is the most limiting factor, the other factors will be ignored in this scenario.

8.3 Workload

The generation of a typical workload is a fundamental part of a benchmark. It determines the quality of the results as well as the flexibility of the benchmark. Thus, in order to achieve a high quality, the workload needs to fulfill three requirements. Firstly, it must be *reproducible*, so that the test scenario is exactly the same for each evaluated overlay, and the test can be repeated a number of times with the same results. Secondly, it must be *scalable*, so that it allows to simulate an arbitrary number of players. And lastly, the workload used for benchmarking should be as *realistic* as possible to allow making meaningful statements about the quality of an overlay of the game.

Fulfilling these three requirements is not a trivial task, since a real (massively) multiplayer gaming session is a complex combination of interdependent processes. Network and node properties like churn rate or connection quality influence the load as well as the way in which players interact with the game world. Both processes have to be modeled in order to reflect critical situations of the SUT (e.g., a massive join on the network level and a high player density on the game level). Node behavior is not application-specific, and its common modeling is discussed in the general workload Section 3.4.

In contrast, modeling a player's behavior depends on various game features. It requires an understanding of game events that generate network messages as well as models for human interaction in games. Generally, four common game workload generation techniques can be identified: (real) *human players*, *traces* from real game sessions, *mobility models*, and *AI players* (also called *Bots*) playing a game. Each of

these approaches has advantages and disadvantages, and they fulfill the above stated workload requirements to different degrees. While human players obviously create a realistic (because real) workload, it is hardly possible to replay a game in a comparable way with human players. Needless to say, it is moreover difficult to find a representative and large-enough set of humans to play a certain game at a given time for a test run. Traces, in contrast, exactly replay an earlier gaming session, allowing for an exact comparison of different systems under test. However, since traces describe one specific gameplay instance, there is no direct way to scale a given trace in terms of number of players or game world size. For a parameterizable workload, a common approach are mobility models. These models define simple rules for the movements of each participant. The main limitation of mobility models in the context of online games is their inability to generate player interactions such as shooting at each other. Although position updates are a major component of a gaming workload, player interaction cannot be neglected. A more sophisticated technique is to use AI players playing a complete game. This application-aware approach incorporates the whole game logic of a specific game and is thus the most complex option. It also takes interaction among players into account and thus allows the generation of all possible event types of a real game, such as shooting or in-game communication. The degree of realism of the latter two approaches depends on the scenario and the tuning of each particular model.

Figure 8.2 summarizes of the four workload generation techniques based on the involved layers of a game. In each case, the network engine is the system under test. The dark gray boxes indicate the layers on which the workload is generated. Figure 8.3 provides a (simplified) overview of the fulfillment of the workload requirements.

The following sections go into further detail of techniques for a synthetic and an application-based generation of a benchmarking workload.

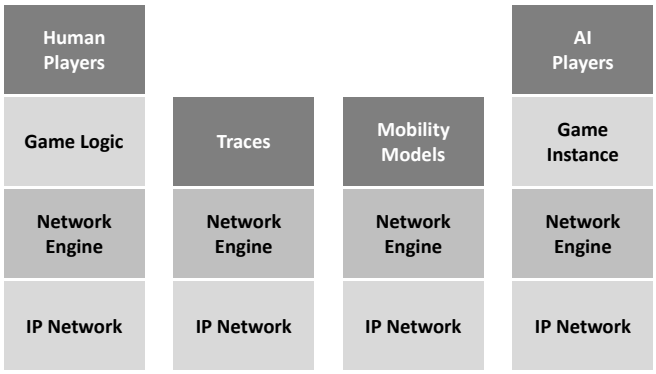


Fig. 8.2: A schematic comparison of game workload generation techniques by the involved layers

	Human Players	Traces	Mobility Models	AI Players
Scalable	✗	✗	✓	✓
Reproducible	✗	✓	✓	✓
Realistic	✓	✓	?	?

Fig. 8.3: A comparison of game workload generation techniques by workload requirements

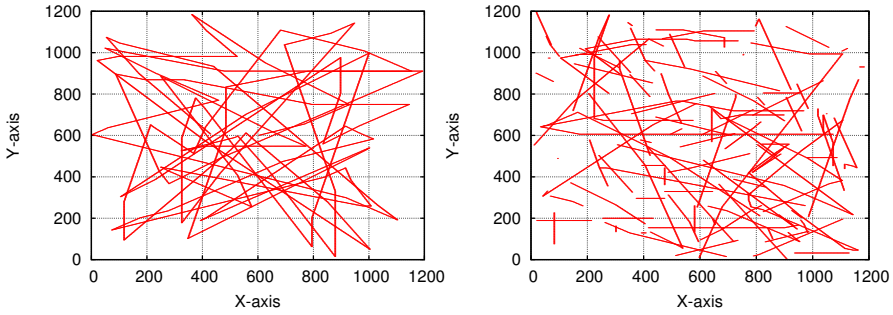
8.3.1 Synthetic Workloads

The synthetic generation of a gaming workload is based on models for player movement and interaction. These models can be plugged onto an overlay implementation using the common interface described above. Thus they emulate game events without having a concrete implementation of a game instance. The degree of realism is lower than with application-aware models, but they can be implemented easily and be used to get a rough performance estimation.

A standard player mobility model is the random waypoint model in Figure 8.4. Each player repeatedly selects a random point in a given coordinate range and navigates to the point with a constant velocity. Such a model can be gradually extended to include interactions like random shooting or random dying. In addition, certain game effects like portals can be modeled as well. Portals enable players to 'teleport', i.e., to suddenly change their position in the game world. This effect causes stress on the system as neighborhood relations change unpredictably and need to be updated immediately in order to assure a smooth playing experience. When the portal effect is enabled, players jump to a random point in the game world, as shown in Figure 8.4.

8.3.2 Application-based Workloads

For the application-based generation of a workload, there are two different approaches, static traces and context-sensitive AI players. *Traces* are complete records of all actions, e.g., movement and interaction, performed by all players in a real gaming session. Such traces are not scalable to any number of players other than the actual number of participants when they were created. On the other side, they provide a reproducible workload which is perfectly realistic. AI players are a context-aware way to generate a workload. They are sensitive to the situations as they occur in the game, and they are able to react to them. The player behavior is recreated



(a) Schematic plot for one player for the random waypoint model without portal effects

(b) Schematic plot for one player for the random waypoint model with portal effects enabled

Fig. 8.4: Random waypoint model with and without the portal effect

much more realistically than in the case of mobility models. In particular, they also allow to model the natural attributes of the players. If implemented well, adjusting the parameters of the AI player allows to imitate even higher level patterns like aggressiveness or skill level. In order to implement such a behavior it is necessary to model two general aspects: The first aspect are the *static constraints* dictated by the game itself. For example, they influence how fast players can move, where they can go and how they can interact. These constraints are mostly invariant, they can be used as workload parameters in order to create different variations of the workload. The second aspect are the *natural attributes* of the players. Some players may be playing more aggressively or defensively, or they can either beginners or highly skilled. It is obvious that the former aspect is much easier to reproduce, but both aspects need to be simulated to create a truly realistic workload. In the following section an example for a concrete implementation of such an AI player is given.

8.3.3 Example AI

This section sketches the conception and development of an AI-based game workload. As a reference, Planet PI4 [17, 11] is used. It is a spaceship first-person shooter (FPS). In order to get an impression of the game the scenario of Planet PI4 is briefly illustrated, and the parameters for the workload adjustment are presented. Figure 8.5 shows a screenshot of the Planet PI4 implementation.

Gameplay Scenario

The game scenario of Planet PI4 consists of n players. Each player is assigned to one of m teams. They compete with other teams. The game world is a 3D space in which the spaceships can freely move in all directions.

Strategic *points of interest* (POI) are randomly scattered within a bounded region of the game world. An example of a POI is a base that can be captured by the teams. For each base a team possesses, it gains points and/or other kinds of rewards such as weapons and energy. Once captured, a team has to defend a base by keeping players from other teams out of the base's range. To capture a base, it is necessary to stay within the range of the particular base with at least one player and to prevent players of other teams to enter that range.

The POI (bases) have two important aspects concerning workload generation:

- The distribution of players in the game world is influenced by the POIs. Particularly, attractive POIs will generate hotspots in player density, while spaces between the POI are expected to have rather low player densities.
- The borders of the region containing the POI are natural borders of the effective game world without the need for artificial boundaries. Although players could move far beyond the borders, there is no incentive to do so. Limiting the effective size of the game world is necessary to be able to control the (average) player density.

This game scenario provides several parameters that can be utilized to adjust the workload.

Workload Parameters

- *Players and teams.* Each player corresponds to a peer in the network. So, the number of players (n) in the game equals the number of peers. The players are divided into m (almost) equally sized teams.
- *POI (bases).* The bases that have to be captured by the teams cause hotspots in the player density. The hotspot magnitudes can be controlled by adjusting the values (i.e., the benefit for the possessing team) of each base separately. Each base has a range in which it can be captured and a minimum time it takes to transfer the ownership of the base to the capturing team.
- *Gameplay region.* The gameplay region is the region within which the bases are located, thus, in which the gameplay happens. Together with the total number of players, its size influences the average player density. The height of the region may be set relatively small to obtain a flat, thus pseudo-2D game world. Pseudo-2D mode is used for gaming overlays that are designed for a 2D world.
- *Ships' capabilities.* The intensities of game activities, such as moving and shooting, are heavily influenced by the corresponding capabilities of the players' spaceships. A very important factor is the maximum velocity. All position changes affect the players' AOI and, thus, require updates in the gaming overlay.

The ship's maximum forward velocity limits the rate of AOI changes. Additionally, the ship's inertia limit the maximum acceleration. Missile fire events have to be delivered reliably, forming a different category than position update messages. Their rate is limited by the maximum missile firing frequency. Furthermore, the missile range determines the maximum area of influence of each player.

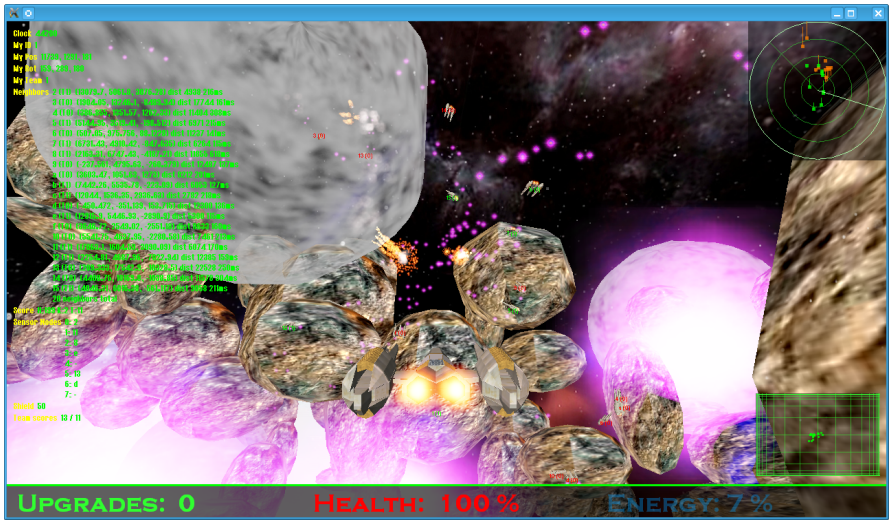


Fig. 8.5: Screenshot of the prototype game Planet PI4

The game scenario and the parameters reflect the static part of the workload. In order to model the dynamic part, artificial intelligence (AI) players called *bots* can be used.

AI Implementation

The goal of implementing an AI is to enable a purposeful behavior of the computer-controlled players (bots). For the workload generation this means that the bots behave in a way that initiates the transmission of network messages in the same way as real players would. Such network messages are mainly triggered by in-game actions. These actions reflect the characteristics of the player's gaming behavior. This includes simple reactions to game events as well as behaviors with a more high-level motivation, like strategies and team play.

An adequate model can be achieved by a goal-oriented AI, since goals can be mapped to behaviors easily. A powerful and easy-to-implement solution are behavior trees (roughly comparable with *hierarchical finite state machines* (HFSM) [12, p. 318–331]). The goals of a behavior tree can be simple or complex. Complex goals

are composed of a sequence of simple sub-goals where each sub-goal is necessary for the goal. The leaf goals of the tree form the interface to the game world. They can gather information about the current game state and interact with the world using concrete actions. Combining goals in such a way allows for intuitive modeling of simple and complex behaviors. The desirability of each goal is periodically evaluated based on the current game state. The goal with the highest desirability score gets executed. For the example scenario a simple implementation with three complex goals based on five sub-goals consist of the following goals:

Go To Position (sub-goal): This goal sets the current speed of the ship to the maximum and steers it towards the destination.

Find Highest Threat (sub-goal): This goal analyzes the enemies that are inside the area of interest. It determines the opponent that poses the highest threat based on distance, angle and shooting frequency.

Attack Opponent (sub-goal): This goal follows the enemy target to take it down. Since an appropriate strategy depends on the distance to the target, we implemented the following strategies: If the target is out of the firing range, approach the target at full speed. If the target is in range, decrease speed, keep following the target and start shooting. If the target is too close, try to flank it by applying lateral thrust to fly around the enemy ship and keep shooting.

Combat(complex): This goal is a sequence of the goals "Find Highest Threat" and "Attack Opponent".

Find Base (sub-goal): The goal checks all bases in the AOI and determines the one that is most desirable to capture. The decision depends on the distance to the base and its current state. Bases that are controlled by the enemy are more interesting to capture than neutral ones.

Capture Base (complex): This goal is a sequence of the goals "Find Base" and "Go To Position".

Find Waypoint (sub-goal): This goal determines an interesting area for exploration. This is done either by selecting a completely random waypoint or by selecting a random point of interest (e.g., bases and repair points).

Exploration (complex): This goal acts as the default behavior. It explores the map until a more meaningful goal arises. It is a sequence of the goals "Find Waypoint" and "Go To Position".

8.3.4 Workload Calibration

The complexity of gaming workloads, and therewith the complexity of the workload generation techniques, induces a high number of possible workload scaling dimensions. This complicates the selection of a representative workload mix and thus requires a particularly careful selection of parameter settings. One option to approach this need is a *calibration* of the artificial workloads with recordings from real game sessions.

For this approach, it is necessary to define a similarity metric for game recordings and to measure the distance of artificial workloads to the targeted real workloads [16]. There are, however two main challenges to overcome. First, game sessions of different sizes must be comparable, because the goal is the scalability of the workload beyond what is achievable in a controlled environment with real players. Second, a sensible compromise between realism (i.e., a high similarity with real workloads) and generality of the workload has to be found. Good solutions to this problem are still an object of active research.

8.4 Metrics

This section describes the metrics used to capture the SUT characteristics. To be independent from a concrete SUT implementation, we focus on macro metrics, i.e., metrics that can be measured on top of the SUT interface (see Section 8.1). In contrast to micro metrics, which refer to internals of specific systems, only macro metrics enable a comparison of all systems implementing the given interface.

Definitions

We use the following symbols for specifying the metrics:

- $P(t)$ as the set of all peers participating in the overlay at time t.
- T as the global set of sampling timestamps.
- $vis(p)$ as the vision radius of peers $p \in P(t)$, which is considered to be constant over time.
- $pos(p,q,t)$ as the position of peer p perceived by peer q at time t. We define that $pos(p,t) = pos(p,p,t)$ is the 'true' position of peer p.
- $N(p,t) = \{q \in P(t) \mid \Delta pos(p,q,t) \leq vis(p)\}$ with $\Delta pos(p,q,t) = \|pos(p,q,t) - pos(p,q,t)\|_2$ as the *ideal* set of neighboring peers for a given peer $p \in P(t)$ at time t. Note that this definition of a neighborhood differs from the definition given in Chapter 2 in that it defines the neighbors of the vision range, which are not necessarily the same as the neighbors of the overlay topology.
- $M(p,t)$ as the set of neighboring peers *known* to a given peer $p \in P(t)$ at time t.
- $cen(p,t)$ as the centroid of the perceived positions of peer p by the neighbors $q \in M(p,t)$ at time t. The centroid is calculated as

$$cen(p,t) = \frac{\sum_{q \in M(p,t)} pos(p,q,t)}{|M(p,t)|}.$$

- $wcc(t)$ as the weakly connected component at time t. It describes the maximum subset of nodes in the connection graph such that two arbitrarily chosen nodes from the subset are connected via an undirected path.

Performance Metrics

The following metrics are measured per peer in constant time intervals, resulting in a set of samples.

- Using confusion matrices, one can divide the neighborhood set of a peer p into the four categories shown in Table 8.1. Based on the confusion matrix, the metrics recall and precision are defined. Recall, thus, describes the ratio between the known relevant neighbors ($M \cap N$) and all relevant neighbors (N) in a node's neighbor set. Precision defines the fraction of relevant peers (($M \cap N$) / M). In case of an ideal system both recall and precision are equal to 1.

	$q \in M(p,t)$	$j \notin M(p,t)$
$q \in N(p,t)$	true positive, $tp(p,t)$	false negative, $fn(p,t)$
$q \notin N(p,t)$	false positive, $fp(p,t)$	true negative, $tn(p,t)$

Table 8.1: Confusion matrix for peer p

We define the the following metrics based on the neighborhood relation of the peers:

- The mean position error at peer p:

$$err(p,t) = \frac{\sum_{q \in M(p,t)} \|pos(p,q,t) - pos(p,t)\|_2}{|M(p,t)|}.$$

  Note that this metric only includes the neighbors actually *known* by peer p.
- The dispersion of the perceived position of peer p relative to the centroid $cen(p,t)$:

$$s_{pos}(p,t) = \sqrt{\frac{\sum_{q \in M(p,t)} \left(\|pos(p,q,t) - cen(p,t)\|_2\right)^2}{|M(p,t)| - 1}}.$$

  In an ideal system the dispersion is zero as the actual position of the peer $pos(p,t)$ and the centroid of the perceived positions of $cen(p,t)$ are identical.

Cost Metrics

- $traf(p,t)$ as the traffic in bytes per second at peer p at time t. The traffic is measured in order to quantify the *costs* of a system. We assume other costs such as CPU, memory, storage, and energy to be uncritical as most of the modern gaming PCs provide sufficient local ressources for playing games.

Global Metrics

The following metric is calculated on the neighborhood relations among the peers:

- The Global Connected Component Factor (gccf), which is defined as follows:

$$gccf(t) = \frac{|wcc(t)|}{|P(t)|}.$$

  It is 1.0 in case that there exists a path between any two arbitrarily chosen nodes in the network. In other words, the overlay is not partitioned. In case of a partition the gccf describes the share of the biggest partition of the total network.

8.5 Example Implementations

Current P2P-based systems that provide the functionality described above can be classified into Distributed-Hash-Table-based (DHT) and unstructured approaches. SimMud [10], Mercury [1], and MOPAR [18] belong to the first category. They disseminate position updates among nodes using a Publish/Subscribe approach on top of a DHT. The virtual world is split into regions. Each region is assigned to a peer who is responsible for managing the subscriptions. Nodes subscribe to a set of fixed regions or an arbitrary part of the virtual world.

MOPAR [18] is a hybrid system, combining a DHT and unstructured P2P. The virtual world is divided into hexagonal zones. MOPAR defines three roles for nodes for each cell. The *home* node is responsible for the corresponding cell. The assignment is done via a DHT mapping, so that a node is most likely not located in the cell for which it is responsible. The *master* node, located in each cell, distributes the messages for its cell, which are received by the *slave* nodes.

Mercury [1] provides multi-attribute range query capabilities, using concepts of DHTs. It creates multiple so-called *attribute hubs*, one for each attribute of the schema. For a virtual world, there would be typically one attribute for each dimension. Each attribute hub is organized similar to a ring of a conventional DHT. Mercury's range query capabilities allow querying for players and sending messages to the players in a certain region of a two- or three-dimensional virtual world.

In unstructured systems, such as pSense [14] or VON [8], peers directly exchange information about the surrounding peers and their current position. Each peer maintains an individual set of neighbors in its AOI.

pSense [14] provides means for localized multicast, which enables fast and efficient dissemination of a participant's position within a given area of interest. Based on the area of interest, each peer has two lists of neighboring peers. Peers in the *near node list* are within the area of interest of a given local node. The *sensor node list* contains up to eight peers in different directions that are just outside the area of interest. Their purpose is to maintain the connectivity of the network by introducing approaching neighbors. In case that sufficient bandwidth is available on the local

node, position updates are sent in rounds to all peers in both lists. Otherwise, only a subset of the surrounding neighbors get the updates directly. Nodes receiving the position update check whether they can forward the position update to the nodes that were omitted by the original sender.

VON [8] solves the neighbor discovery problem by maintaining a Voronoi diagram based on the peers' positions in the virtual world. According to the area of interest, peers surrounding a given local peer are divided into boundary and enclosing neighbors, which are notified with position updates from the local peer. Boundary neighbors inform a local node about new potential neighbors.

8.6 Benchmarking Results

Having introduced the game basics, possible workloads and metrics, this section presents an exemplary benchmark implementation, execution, and analysis using a workload generated from a mobility model. Thereby, we investigate the characteristics of the tested systems with respect to performance, validity, costs, and fairness. The goal of this evaluation section is to demonstrate the feasibility of our benchmarking methodology and give a first impression of the characteristics of the tested systems. Four different systems for spatial information dissemination, satisfying the functional interface as described in Section 8.1, were implemented and tested. The systems are: the unstructured approaches (i) VON [8] and (ii) pSense [14], (iii) the content-based Publish/Subscribe approach Mercury [1], which is built on top of the Chord DHT, and (iv) a simple client-server-based approach (C/S), which serves as a performance reference. The client-server approach works as follows: All nodes participating in the game are connected to a central server and send their position updates every 200 ms. From these position updates the server computes an updated neighbor set per peer and disseminates this information to all peers in the network every 200 ms. All SUTs have been implemented in the discrete event-based overlay simulator PeerfactSim.KOM [15]. The results of this section are based on previous work [7].

The workload is varied in four scenarios:

Baseline. Initially, a constant baseline workload setup is defined, representing an environment with idealized conditions, which gives an impression on the system behavior under such conditions. The baseline parameters are shown in Table 8.2. The parameter values haven been chosen based on the work by Schmieg et al. [14]. For the baseline workload, a simple random waypoint model is used. The update frequency of five units per second was chosen based on an analysis of MMOGs conducted by Chen et al. [5]. The simulation setup is as follows: Initially, 250 nodes join the system within eight minutes of simulated time. Afterwards, a stabilization phase of two minutes takes place. The measurement interval starts at minute ten and lasts until the end of the simulation after thirty minutes.

To investigate the *performance* and *validity*, the neighbour set recall ($recall(p,t)$) and precision ($precision(p,t)$) as well as the position error ($err(p,t)$) and the per-

ceived position dispersion ($s_{pos}(p,t)$) are measured. Precision is less relevant than recall, since – besides a potential traffic penalty – false positives in the neighbour set usually do not hurt the application. The traffic $traf(p,t)$ quantifies the operational *costs*. *Fairness* according to service quality is measured by $F(recall)$, $F(precision)$, $F(err)$, and $F(s_{pos})$, where $F(x)$ is Jain's Fairness Index, as described in Chapter 3. Cost fairness is measured by $F(traf)$.

Parameter	**Value**
Game World Size	1200 x 1200 units
Area of Interest radius	200 units
Number of nodes	250
Simulation duration	30 min
Movement model	Random waypoint model
Movement speed	20 units/s
Update frequency	5/s
Underlay model	GNP Latency Model [13]
Upload capacity	16 kB/s
Download capacity	128 kB/s

Table 8.2: Baseline Parameter Setup

Scalability. For investigating the scalability of the system, the workload is varied both horizontally and vertically. The former scales the total number of nodes while preserving the virtual world size. Horizontal scaling thus affects the player density in the virtual world. Vertical scaling is realized by increasing the player velocities, resulting in a higher fluctuation of AOI neighbors and potentially higher position errors. We define a system to be scalable up to a certain workload level if the performance, validity, and costs do not exceed or drop below predefined thresholds. These thresholds, however, are application-specific.

Stability. The stability of the SUT is investigated in a heavy churn scenario with an exponential session length with an average of 50, 25, and 10 minutes. The relevant metrics for stability are the same as for validity and fairness. While it is expected that the system responses are valid at all times, a degradation of fairness is an indicator for upcoming stability problems. A low fairness in terms of costs reflects potentially overloaded nodes, whereas a low service quality fairness indicates possibly starving nodes.

Robustness. In order to test the robustness of the system, a massive join, a massive crash and a packet loss scenario are applied. For the first scenario, 50% and 100% new peers join the system at once. In the second scenario, the percentage of failing peers is set to 25% and 50%. For the packet loss scenario, loss rates are set to 5%, 10%, and 25%. The system is considered stabilized after a massive join or leave of nodes when the moving average of 10 seconds of a given metric stabilizes at a constant level. A similar method has been suggested by Jain in order to detect a

steady state in a set of measurements [9]. In the message loss scenario, the relevant metrics are the same as for validity and performance.

8.6.1 Results

This section briefly discusses the results of the simulated benchmark runs using our synthetic gaming workload.

Baseline

The results for the baseline benchmark are shown in Table 8.3. When looking at the average recall ($\widehat{recall}$) of all four systems, one can notice that VON provides the lowest neighbour detection rate, whereas the remaining three systems show a recall of above 0.9. With respect to the position error ($\widehat{err}$), however, VON shows the best performance with a position error of around two units, which is about half of the position error of the C/S-based approach. The reason for the good performance of VON is that updates are disseminated directly over one hop, whereas the C/S approach always needs two hops (Player A to Server and Server to Player B). Mercury shows the worst performance with respect to the position error as updates are disseminated over multiple hops in the DHT.

System	Unit	C/S	pSense	VON	Mercury
$\widehat{recall}$	1	0.9791	0.9304	0.8304	0.9162
$F(recall)$	1	1.0000	1.0000	0.9586	0.9989
$\widehat{err}$	units	5.2043	9.8512	2.0186	13.5779
$F(err)$	1	0.9558	0.9865	0.6921	0.8146
$\widehat{traf_{up}}$	kB/s	0.2181	15.5601	3.5732	5.5020
$F(traf_{up})$	1	1.0000	1.0000	0.8469	0.8435
$\widehat{traf_{down}}$	kB/s	2.3199	15.5601	3.5733	5.4955
$F(traf_{down})$	1	0.9965	0.9995	0.8471	0.8554

Table 8.3: Results for the baseline benchmark averaged over time and peers

VON shows a position error fairness index of 0.69, which indicates that the position error that peers in the VON overlay perceive is unequally distributed. In contrast, the C/S-based approach as well as pSense show a fairness value above 0.95 and, thus, provide a good service fairness. Concerning upload traffic, pSense utilizes the full available bandwidth of 16 kB/s, because position updates include the entire receiver list with all neighboring peers in the vision range. As expected, the C/S approach consumes the least bandwidth, because a node sends its position updates only to the server instead of multiple neighboring peers, and the server bandwidth

is not taken into account. When investigating the cost fairness of all systems, one notices that VON and Mercury distribute the load less equally than the C/S approach and pSense.

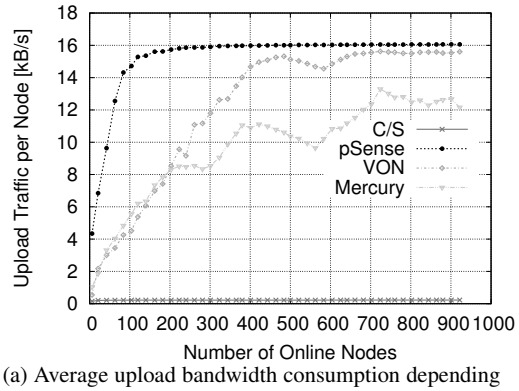
Scalability

Figure 8.6 shows the results for the scalability benchmark using horizontal scaling (number of nodes), followed by Figure 8.7, showing the results using vertical scaling (node speed). When scaling horizontally, the upload bandwidth consumption of pSense is growing rapidly with the increasing number of peers and reaches the predefined maximum of 16 kB/s. With more than 400 nodes, VON's upload consumption also reaches the 16 kB/s maximum, and VON is no longer capable of keeping the neighbour sets up to date. This results in a rapidly decreasing recall and at the same time in a rapidly increasing position error. As VON remains in this state of saturation, the overlay structure collapses, resulting in a recall of almost zero. Mercury also shows a recall decreasing below the clearly unacceptable value of 50% with 500 nodes. The position errors of pSense and Mercury increases linearly with an increasing number of peers. Under the assumption that the server has sufficient bandwidth, the C/S results remain constant.

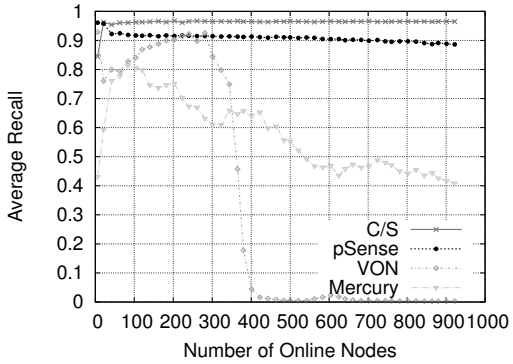
Figure 8.7a shows the relation between the node speed (vertical scaling) and the global connected component factor. As one can observe from the plot, the four systems remain fully connected up to a node speed of 250 units per second. Above this threshold, the connectivity of VON starts to deteriorate, whereas the remaining three systems remain fully connected. The reason for this deterioration is that a VON node p does not maintain connections to other nodes beyond its vision range $vis(p)$. In the case where that the node speed exceeds the vision radius per second (200 units/s), VON is unable to detect new bypassing nodes because the relative speed of two nodes passing each other is twice the average node speed. With respect to recall, all four systems show a similar behavior as their recall drops with an increasing node speed, as shown in Figure 8.7b. All four systems show a linearly increasing position error up to a speed of 200 units/s (Figure 8.7c). Above this value, all peer-to-peer-based systems experience higher fluctuations in the position error. VON's position error does not increase further as its overlay structure has completely collapsed.

Stability

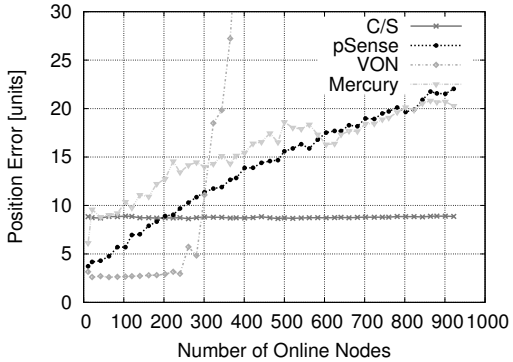
The results of the stability benchmark for different levels of node churn are shown in Table 8.4. With respect to recall, C/S, Mercury, and pSense remain stable with an increasing churn rate, whereas the recall of VON is dropping to 0.864 at ten minutes mean session length. Another interesting observation relates to the position error. The C/S approach achieves a position error of around 5.4 units whereas pSense and Mercury have position errors of about 9.2 and 11.5 units, respectively. VON shows the best performance with a position error of about 3 units. The reason for this is,



(a) Average upload bandwidth consumption depending on the number of online nodes

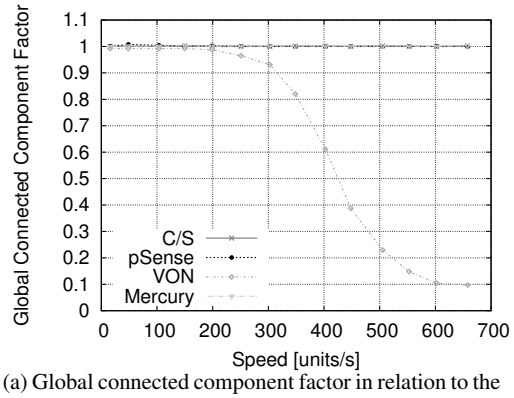


(b) Average recall of the neighbor sets depending on the number of online nodes

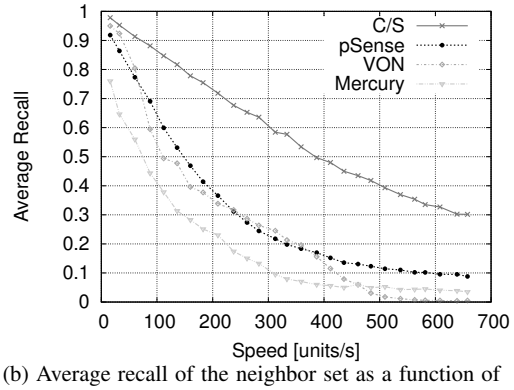


(c) Average position error depending on the number of online nodes

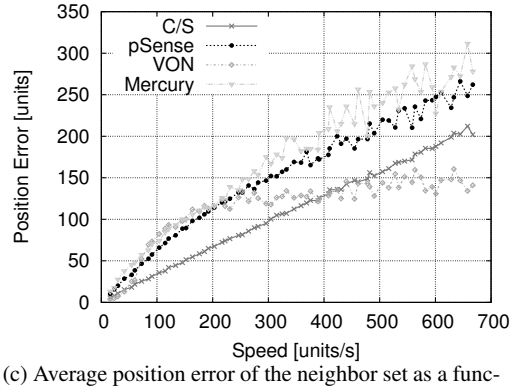
Fig. 8.6: Results of the scalability benchmark using horizontal scaling by increasing the number of nodes



(a) Global connected component factor in relation to the average speed



(b) Average recall of the neighbor set as a function of the speed



(c) Average position error of the neighbor set as a function of the speed

Fig. 8.7: Results of the scalability benchmark using vertical scaling by increasing the node speed

again, the dissemination of position updates in one hop (VON) and two hops (C/S), as already described in Section 8.6.1. pSense shows the highest position error, because position updates are sent redundantly, saturating the upload bandwidth, which in turn causes the loss of update messages. Similarly to the recall results, Mercury's multi-hop dissemination leads to higher position errors.

System	**Unit**	**C/S**			**pSense**			**VON**			**Mercury**		
Mean Session Time	**min**	**50**	**25**	**10**	**50**	**25**	**10**	**50**	**25**	**10**	**50**	**25**	**10**
$\widehat{recall}$	1	0,977	0,976	0,973	0,929	0,928	0,925	0,974	0,957	0,864	0,939	0,939	0,940
$F(recall)$	1	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,992	0,955	0,970	0,942	0,880
$\widehat{err}$	units	5,412	5,416	5,403	9,705	9,270	8,955	3,311	2,918	2,652	11,670	11,519	11,922
$F(err)$	1	0,959	0,959	0,958	0,987	0,985	0,984	0,893	0,877	0,845	0,958	0,933	0,873
$\widehat{traf_{up}}$	kB/s	0,219	0,219	0,219	15,631	15,607	15,583	9,578	9,031	7,036	7,560	6,908	5,882
$F(traf_{up})$	1	1,000	1,000	1,000	1,000	1,000	1,000	0,993	0,981	0,931	0,766	0,773	0,734
$\widehat{traf_{down}}$	kB/s	2,107	2,056	1,936	15,211	15,172	15,094	9,337	8,801	6,853	7,206	6,417	5,160
$F(traf_{down})$	1	0,997	0,996	0,995	1,000	0,999	0,999	0,993	0,981	0,931	0,941	0,922	0,886

Table 8.4: Averaged results of the stability benchmark for different levels of churn

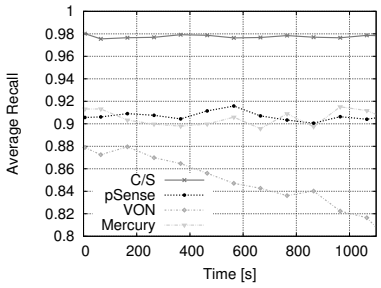
Figure 8.8 shows the results for the stability benchmark with portal effects enabled. As shown in Figure 8.8a, VON's recall is dropping over time as the frequent jumps of nodes causes VON to lose those contacts, and it becomes unable to reintegrate a node into the virtual world after it uses a portal. This loss in connectivity is also visible in Figure 8.8c, which shows the Global Connected Component Factor $gccf(t)$ as a function of time. VON is constantly losing connectivity over time due to players using the portals. In contrast, the recall of Mercury and pSense remain constant at a level of 0.9 and remain fully connected with a Global Connected Component Factor of 1.

Concerning the position error $\overline{err(t)}$, all systems remain constant at a certain level. VON shows the lowest position error of about three units, followed by the C/S approach with an average position of about five units. This is close to the theoretical minimum given the speed of 20 units/s and an update frequency of 5 per second.

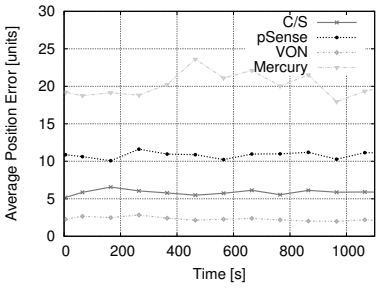
When comparing the position error results with the results obtained during the churn workload, C/S, pSense, and VON provide the same performance. Only Mercury shows a position error twice as high as in the minimum churn workload.

In terms of upload traffic, which is shown in Figure 8.8d, pSense again consumes the most bandwidth due to updates being sent redundantly to surrounding neighbors.

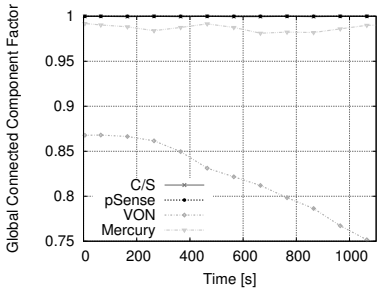
Figure 8.9 shows the results for the single point movement model, where all players in the game world move towards a single point and meet there at the same time. The movement model was configured such that one contraction cycle (minimum density $\rightarrow$ maximum density $\rightarrow$ minimum density) takes ten minutes of simulated time. This results in a much lower node speed of about three units/s in comparison to the speed in the baseline benchmark. When investigating the recall of the four systems, it can be observed that VON's and Mercury's recall degrade in the moment of maximum density of nodes ($t = 300s$). When the nodes de-contract, resulting in a decreasing node density, VON and Mercury are able to recover, resulting in an



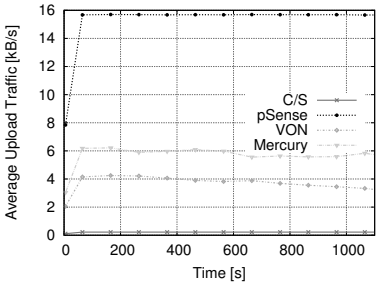
(a) Average recall with portal effects enabled



(b) Average position error with portal effects enabled



(c) Global Connected Component Factor with portal effects enabled



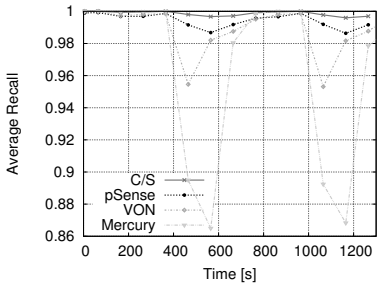
(d) Average recall for the nodes neighbor set for different levels of message loss

Fig. 8.8: Selected results of the stability benchmark using the portal workload scheme

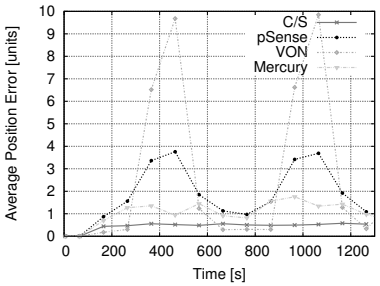
increasing recall of almost 1 and a position error drop of about one unit. Looking at the upload and download utilization, which are shown in Figure 8.9c and 8.9d, it can be observed that VON shows the highest sensitivity to the node density induced by the single point movement model. Again, pSense fully utilizes the upload and download bandwidth. In contrast, Mercury's bandwidth usage remains almost constant at 6 kB/s. As one would expect, all three peer-to-peer-based systems show a symmetrical upload and download usage, whereas the clients' bandwidth utilization of the C/S approach is highly asymmetrical.

Robustness

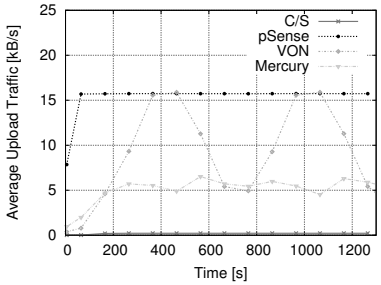
Figure 8.10a shows the recall as a function of time ($\overline{err}(t)$) in the 100% join scenario. While pSense shows a small temporary drop in recall, VON and Mercury are not able to handle the increase of nodes and level out at around 60%. Figure 8.10b shows the CDF of the position error right after the the massive join. VON's position



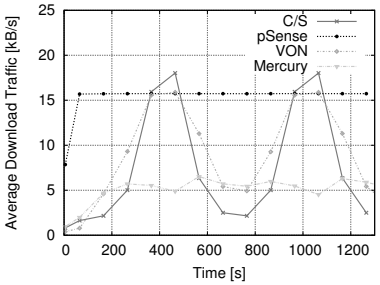
(a) Average recall for the single point movement model



(b) Average position error of nodes for the single point movement model



(c) Average upload bandwidth usage for the single movement model



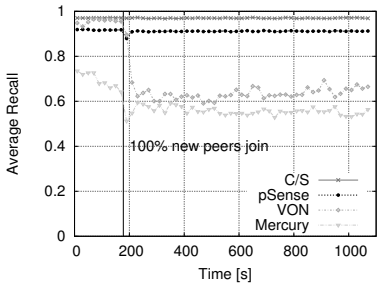
(d) Average download bandwidth usage for the single movement model

Fig. 8.9: Selected results of the stability benchmark using the single point mobility scheme

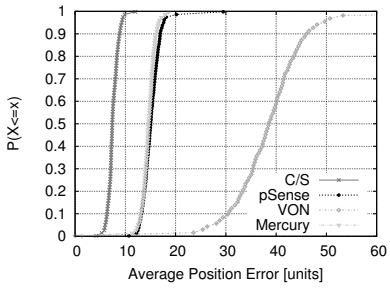
error is much higher and also has a higher variation than the position error of the other systems. The higher spread can also be observed as a lower value of $F(err)$ in Table 8.4. Finally, Figure 8.10c shows the recall in the 50% crash scenario. Again, pSense's recall drops only for a few seconds. Both VON and Mercury, however, drop and remain at a significantly lower level than they would be able to achieve with 250 nodes without a crash (cf. Fig. 8.6b).

8.6.2 Discussion of the Benchmark Results

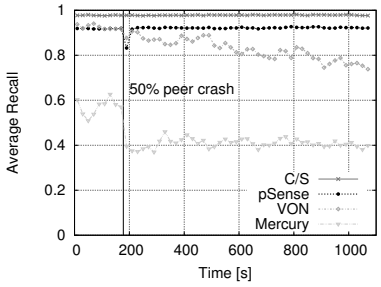
This section discusses the overall results and compares the findings with the results and evaluation methodologies of the respective papers. In the original VON paper[8], the authors evaluated their approach with respect to scalability, validity, and robustness, which the authors call reliability. For evaluating the scalability of their approach, the authors increased the number of nodes in the game world and



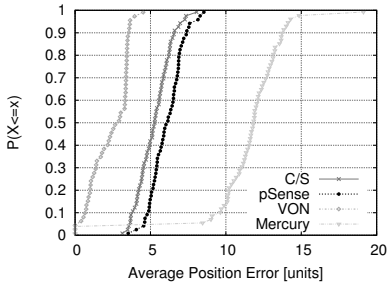
(a) Average recall with massive join from 250 to 500 nodes



(b) CDF of the average position error after a massive join from 250 to 500 nodes



(c) Average recall with massive crash from 250 to 125 nodes



(d) CDF of the average position error after a massive crash from 250 to 125 nodes

Fig. 8.10: Selected results of the robustness benchmark using the massive join and massive leave workload

measured the load on the nodes in the system. Besides looking into the number of nodes, this benchmark has shown that the speed of the nodes also has a significant impact on the system performance. The authors already mention that a "problem arises when a node moves faster than what boundary neighbors can notify". Indeed, the performance of VON degrades significantly at higher levels of node speed. Furthermore, the mobility model has a significant impact on the the robustness, stability, and scalability capabilities of the tested overlay and should also be varied for the performance evaluation. Looking into robustness, VON was originally evaluated under different levels of message loss. This benchmark adds an evaluation under massive join and leave of peers, which also plays an important role; it has shown to be a discriminating factor.

In its original publication, pSense has been evaluated with respect to the scalability by increasing the number of nodes. In addition, the authors considered the performance vs. cost trade-off. Comparing results and methodology of the pSense paper with the above results, this work has complemented the tests with an evaluation under an increasing speed of the nodes or a varying mobility model. Further-

more, the authors did not look into the stability and robustness capabilities of pSense by considering churn or an unreliable underlay (message loss or varying delay).

Mercury has originally been evaluated with respect to scalability by increasing the number of nodes in the system up to 100 peers. The authors measured the normalized load on the nodes as well as the delivery delay. The latter is correlated to the position error metric presented here. In their evaluation, the authors assume an idealized environment with no churn or message loss. Here, a detailed evaluation of the stability and robustness capabilities of the system has been added.

This section has demonstrated that the existing evaluation gaps in the evaluations of the respective systems can be closed using a systematic benchmarking methodology, as presented in this book.

8.7 Conclusion

In this Chapter, a benchmarking approach for peer-to-peer gaming overlays has been presented, focusing on the two aspects of interest management and game event dissemination. After the identification of the targeted class of SUTs, the first key issue is the definition of a SUT interface. The interface that has been specified is generally applicable to different first or third person online games where players interact in a large 2D or 3D world. Also, the interface has been designed to be implemented without much effort on top of the common IDOs such as VON or pSense.

Two alternative gaming workload generation techniques have been presented: a simple synthetic method using a mobility model as well as a detailed method incorporating a real gameplay and autonomous AI players. In the latter, the important workload factors do not describe the players' behavior, but rather the limits set by the game mechanics and setup (number of players, teams, region size, etc.). This approach takes the high degree of interactivity of a game into account, which is not the case for a mobility model or traces.

Furthermore, a set of metrics that are important for the user experience in a game has been defined. Those metrics are mapped to the generic peer-to-peer quality aspects. This shows that the important aspects are covered by the metrics, besides a clear classification of those.

Finally, a concrete implementation of a benchmark for peer-to-peer gaming overlays, using a workload based on a mobility model, has been described. The evaluation of four exemplary systems has shown that a benchmarking approach allows for a systematic comparison of such systems. It closes the evaluation gap arising from the piecemeal evaluations of the respective original publications.

References

[1] Ashwin Bharambe, Mukesh Agrawal, and Srinivasan Seshan. "Mercury: Supporting Scalable Multi-Attribute Range Queries". In: *SIGCOMM Comput. Commun. Rev.* 34 (2004), pp. 353–366.

[2] Ashwin Bharambe, Jeffrey Pang, and Srinivasan Seshan. "Colyseus: a distributed architecture for online multiplayer games". In: *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*. San Jose, CA: USENIX Association, 2006, pp. 12–12.

[3] Ashwin Bharambe et al. "Donnybrook: enabling large-scale, high-speed, peer-to-peer games". In: *ACM SIGCOMM 2008 conference on data communication (SIGCOMM '08)*. Seattle, WA, USA: ACM, 2008, pp. 389–400.

[4] Luther Chan et al. "Hydra: a massively-multiplayer peer-to-peer architecture for the game developer". In: *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*. Melbourne, Australia: ACM, 2007, pp. 37–42.

[5] Kuan-Ta Chen et al. "Game traffic analysis: An MMORPG perspective". In: *Computer Networks* 50.16 (2006), pp. 3002–3023.

[6] Lu Fan, Phil Trinder, and Hamish Taylor. "Design Issues for Peer-to-Peer Massively Multiplayer Online Games". In: *International Journal of Advanced Media and Communication* 4.2 (2010), pp. 108–125.

[7] Christian Gross et al. "Towards a Comparative Performance Evaluation of Overlays for Networked Virtual Environments". In: *Peer-to-Peer Computing, 2011. P2P'11. Eleventh International Conference on*. Sept. 2011, pp. 34–43.

[8] Shun-Yun Hu and Guan-Ming Liao. "VON: A Scalable Peer-to-Peer Network for Virtual Environments". In: *IEEE Network*. Vol. 20. 4. 2006, pp. 22–31.

[9] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc, 1991.

[10] Bjorn Knutsson et al. "Peer-to-peer support for massively multiplayer games". In: *Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '04)*. 2004.

[11] Max Lehn et al. "An Online Gaming Testbed for Peer-to-Peer Architectures". In: *The 2011 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM '11)*. ACM, Aug. 2011.

[12] Ian Millington and John Funge. *Artificial Intelligence for Games, Second Edition*. Morgan Kaufmann Publishers Inc., 2009. ISBN: 978-0123747310.

[13] T. S. Eugene Ng and Hui Zhang. "Towards global network positioning". In: *1st ACM SIGCOMM Workshop on Internet Measurement*. ACM, 2001, pp. 25–29.

[14] Arne Schmieg et al. "pSense - Maintaining a Dynamic Localized Peer-to-Peer Structure for Position Based Multicast in Games". In: *IEEE International Conference on Peer-to-Peer Computing*. 2008.

[15] Dominik Stingl et al. "PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems". In: *Proceedings of the 2011 International Conference on High Performance Computing & Simulation*. 2011, pp. 577–584.

[16] Tonio Triebel et al. “Generation of Synthetic Workloads for Multiplayer Online Gaming Benchmarks”. In: *International Workshop on Network and Systems Support for Games (NetGames12)*. Venice, Italy: IEEE, Nov. 2012.

[17] Tonio Triebel et al. “Peer-to-Peer Infrastructures for Games”. In: *NOSSDAV ’08: 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. Braunschweig, Germany, 2008, pp. 123–124.

[18] Anthony P. Yu and Son T. Vuong. “MOPAR: A Mobile Peer-to-Peer Overlay Architecture for Interest Management of Massively Multiplayer Online Games”. In: *International Workshop on Network and Operating System Support for Digital Audio and Video*. 2005, pp. 99–104.