RTSenv: An Experimental Environment for Real-Time Strategy Games

Siqi Shen, Otto Visser, and Alexandru Iosup
Delft University of Technology, The Netherlands. Email: {S.Shen,O.Visser,A.Iosup}@tudelft.nl

***Abstract*—Today, Real-Time Strategy (RTS) games entertain tens of millions of players world-wide. This growing population expects new game designs and more scalable games every year. However, few tools and environments exist for game designers and implementers; of these, even fewer are available to researchers and game communities. In this work, we introduce `RTSenv`, an environment and associated set of tools for RTS games. Our environment can configure and manage the main aspects of RTS games, such as maps, computer-controlled units, and game scenarios. `RTSenv` leverages multi-cluster systems and reactive fault tolerance mechanisms to perform robust, multi-machine, and multi-instance game experiments. Using our reference implementation of `RTSenv` in DAS-4, a real multi-cluster system, and Amazon EC2, a commercial cloud, we show that our approach can be used in a variety of scenarios. Our results give evidence that several common assumptions made by researchers about game workloads do not hold in general for RTS games and thus warrant a more detailed investigation.**

I. Introduction

Among the different gaming genres, Real Time Strategy (RTS) games such as StarCraft II (one of the best-selling games of 2010 [1]) are played by millions of players daily. To address the increasing requirements of the gamers and the increasing competitiveness of the market, many new RTS game technologies [2]–[4] have appeared in the past five years, and a new generation of massively multiplayer RTS games, such as Picaroon [5], is currently under development. Although the development of a new RTS game can take multiple years to develop [6], and the abundance of challenges in the design, implementation, and testing of RTS games makes experimental tools valuable, few experimental tools are available for RTS game research and development. To address this situation, in this work we introduce `RTSenv`, an experimental environment for RTS games.

Even the casual players of online RTS games incur a specific near-real-time constraint that limits the acceptable response times for issued commands to 200-300 milliseconds [7]. When this constraint is not met, players have poor gameplay experience, and may quit in favor of other games. The near-real-time constraint is difficult to meet continuously for most consumer-grade computers, even today. As a consequence, online RTS games rely on complex technology that balances the computational, memory, and bandwidth components of the workload on the different system components and players.

The limited availability of experimental tools makes it difficult to generalize or even understand the results of previous studies. It is perhaps symptomatic for the state of this research field that a recent study [8] argues against the practical viability of P2P-based games as a conclusion of real-world experiments, thus contradicting several previous simulation-based studies that indicate otherwise [3], [4]. Moreover, while some large gaming companies perform extensive game studies, including comprehensive user studies [9], [10], few others can afford building the tools necessary for such approaches; even the few that do, have little expertise with large-scale experimental environments [6]. Despite recent interest in experimental tools [11]–[13], there currently exists no public experimental RTS environment. Although simulators may be appropriate for studying RTS games, they represent simplifications of the real systems and games; when their simplifying assumptions do not hold, their results fail to describe reality. Although many tools already exist for testing distributed systems, RTS games have additional, idiosyncratic, performance-affecting configuration parameters, such as the map size and the number of units.

In this work we introduce `RTSenv`, which is designed for experimental RTS game studies. `RTSenv` can be used to evaluate the performance of RTS games under a variety of game configurations and scenarios, from traditional performance evaluation to game design. Besides traditional system performance metrics such as CPU, memory, and network consumption, `RTSenv` can assess RTS-specific operational metrics such as player scores, and the number of active and profitable units. `RTSenv` can operate on a variety of physical platforms, from multi-cluster wide-area environments such as DAS-4 [14], to cloud computing machines and even single, multi-core desktop computers. Our main contributions are:

1) We analyze the requirements of experimental RTS environments (Section II);
2) We design and implement `RTSenv`, an experimental RTS environment (Section III);
3) We show through experiments with a popular RTS game in real multi-cluster and cloud infrastructures how `RTSenv` supports various scenarios (Section IV).

II. Background on Real-Time Strategy Games

A. Use Case: `OpenTTD`, a Real-Time Strategy Game

As main use case for our work we focus on `OpenTTD` [15], which is an open-source, popular RTS game. `OpenTTD` has been developed since 2004 by a community of volunteer game developers lead by Remko Bijker as an extension to the commercial game Transport Tycoon Deluxe by Chris Sawyer (MicroProse, 1994). `OpenTTD` is a business simulation game with a wide appeal: the latest stable release (1.1.1) has been downloaded more than 130,000 times and the game is used as textbook companion for college-level business courses [16].

OpenTTD has the features of commercial RTS games. The game world, which may be randomly generated or selected from the many community-created maps, emulates the real world through a combination of realistic geography, economy, and demographics. The player has complete control over a transport company: buying transport vehicles such as buses and trains; building transportation paths such as roads and train tracks; planning and managing the operation of the company; etc. Rival companies compete against each other to achieve the highest profit by transporting passengers and goods. OpenTTD game sessions (games) can be played against human and/or Artificial Intelligence (AI)-controlled players.

OpenTTD follows the typical program structure of an online RTS game, in which the game world is maintained on a server, and each player connects and interacts with the game world through a client application running on the computer/device of the player. One of the players often runs the server alongside a client; alternatively, the server is placed on a "neutral" computer for ranked games and competitions. The game server repeatedly executes a main game loop comprised of the sequence "get and process player input", "update the global game world", and "send (small) updates about the game world to each player"; based on the latter step, each client can reconstruct the state of the world, effectively performing the same updates of the global game world as the server.

OpenTTD vs. other RTS games Although a variety of RTS games exist, the AAA game market has focused since 2005 on games such as the *Age of Empires* and the *Starcraft* series, each offering the player large worlds, the possibility to control large numbers (tens or hundreds) of vehicles, and many options to build real-world-like cities. With the exception of in-game combat, OpenTTD is similar to these games. OpenTTD also employs a non-violent alternative to traditional combat, in that players may use advanced tactical and micro-managementskills to restrict the free movement of the vehicles of other players. OpenTTD can also offer sufficient challenge to players of various ability and experience: from the over 30 AIs created and made public by the game's community, several can give a good challenge to the good human player, while others can win comfortably even against the expert human player. OpenTTD is highly configurable. Before starting the game, the players can specify the map structure, the maximum number of vehicles, the inflation rate of the economy, etc. Players can select the amount of water (via the sea level), the "hilliness" of the map, and the density of economic resources (via the industry and town density). This level of configuration detail, which is common for RTS games [17], makes modeling such games a difficult and recurrent process.

B. Requirements

Starting from the use case, we synthesize three main requirements for an experimental RTS environment:

1) *Control experiments*: the environment must have the ability to control experiments (start, stop, monitor, use results) without input from a human experiment manager. The test infrastructure may be any of desktop machine of the programmer/researcher, local studio cluster, multi-cluster grid of computers shared by multiple studios, or cloud computing infrastructure. The environment must be able to perform experiments when real users are present, but also when no real users are present; for the latter, the environment must support both the use of traces/replays and the use of AI-controlled players. Last, the environment must support RTS-specific configuration parameters, such as map structure, unit count, number of players (including AIs), etc.
2) *Evaluate and model RTS operation*: the environment must support traditional system performance metrics; in particular, it should be able to report metrics concerning resource consumption (CPU, network, disk, memory) over time. Besides reacting to network conditions, in particular latency and throughput, players may respond to other sources of performance degradation, such as the server's processing (CPU) speed. The environment must also support RTS-specific operation abstractions and metrics, such as player scores, and the number of active and profitable units.
3) *Compare game design choices*: the environment must have support for comparing the results obtained from different scenarios as support for game design decisions. For example, it is common for game designers to package AI players with commercial games; however, selecting a specific AI type or configuration for a specific (random) map should be based on an in-depth analysis of the AI performance/resource consumption trade-off.

We show in Section IV that an environment satisfying these three requirements can be used in a variety of experimental scenarios, from performance evaluation to game design. We conjecture that such an environment can be used for unit testing and debugging (both important parts of game development), and for systems research, for example for comparing algorithms, methods, and techniques under a common and shareable experimental environment.

III. The RTSenv Environment

In this section we present the RTSenv environment–in turn, the architecture, the RTS-specific features, and the implementation and adaptation details.

A. Architectural Overview

RTSenv consists of four modules (see Figure 1), Utilities, Analysis, RTS, and Runtime. The Utilities and Analysis modules together automate the environment- and game-specific configurations, and collect and analyze the environment- and game-specific results of the experiments. The RTS module is used to set up and monitor the gaming environment. The *Runtime* module is responsible for provisioning machines for the single node/cluster/multi-cluster grid/cloud computing environment used by RTSenv. This module is also responsible for organizing, managing, and executing the experimental process, which is based on jobs. Part of the *Runtime* module, the *Resource Management* component is responsible for

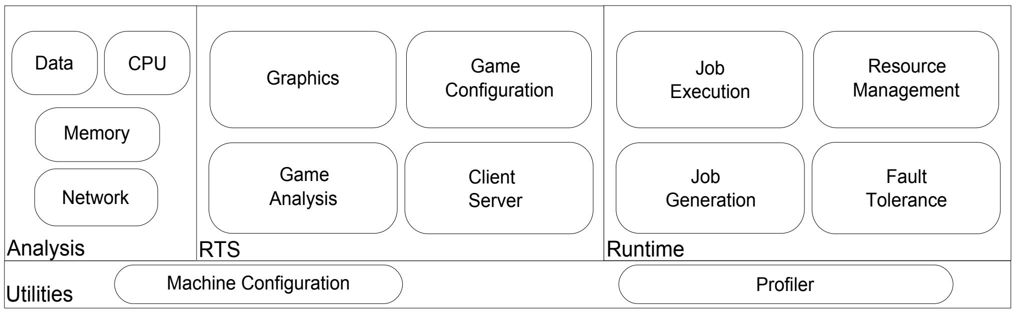

Fig. 1. The `RTSenv` architecture.

organizing the execution of experiments; for example, to manage the execution of experimental jobs in multi-cluster environments, it operates an FCFS queue that interfaces with the resource manager of the environment. This component is also responsible for acquiring and releasing resources, and for invoking the Job Execution component on each allocated machine. The *Job Generation* component parses the experiment description, then generates configuration files and experiment jobs. Jobs can be single-machine clients, single-machine servers, or multiple-machine instances comprising one server and multiple clients. The *Fault Tolerance* component uses a reactive, retry-based fault tolerance mechanism in which failed jobs are re-submitted for execution until they succeed or the number of failures exceeds a user-defined threshold; failures are detected at the end of the job execution by checking the job output.

Experiment Control `RTSenv` allows users to run single and multi-machine experiments using a variety of infrastructures, among them, desktop computers, clusters, multi-clusters, and clouds. For confidence in experiment results, users may specify the number of repetitions for single and multiple-instance experiments. `RTSenv` also allows its users to specify a maximum experiment runtime; it will stop overdue or user-selected experiments. `RTSenv` assumes that the environment in which it runs achieves synchronization between compute nodes; for example, time synchronization through the use of the NTP protocol, machine start synchronization through multi-machine allocation and co-allocation, etc.

B. RTS-Specific Features

Many RTS games, such as StarCraft II, provide a in-game recording mechanism (*replay*), which saves all the game commands issued by players. If the game world simulation is deterministic, replays can be used to reproduce the recorded game. Replays are commonly used by game operators to debug games (*replay-based testing*), and by players to share their gaming achievements and to learn from more skilled players. `RTSenv` supports *replay-based testing*.

`RTSenv` already supports many of the abstractions present in modern RTS games. First, `RTSenv` can control the virtual world's geography by changing the map size and structure (such as density of water or hills). Second, `RTSenv` can control the level of challenge proposed by the virtual world through parameters such as the number of resources present on the map, the amount of starting resources for each player, the types of resource collectors, etc. Third, `RTSenv` can control the maximum allowed player presence, for example by limiting the number of units each player can control. Fourth, our tool can control the in-game duration of a game session, for example one year.

Besides traditional performance metrics, `RTSenv` can measure various RTS-specific metrics, such as the scores registered per player, the number of (profitable) units per player, etc.

C. Qualitative Analysis of the `RTSenv` Design

Meeting the requirements `RTSenv` meets the three requirements formulated in Section II-B. For Requirement 1, `RTSenv` combines experiment control (described earlier in this section) with replay-based testing abilities. For Requirement 2, `RTSenv` can evaluate and create simple statistical models of traditional performance evaluation and RTS-specific metrics. For Requirement 3, `RTSenv` can be used to test and compare the performance of different AIs under various game settings (map structure, etc.)

Fit with the industry game development process Due to the high costs and risks associated with game development processes, the industry has started to mature, and today most games are developed through the same *basic game development process* [18, Ch.2] [6], which includes steps such as concept discovery, prototyping, pre-production, full production, quality assurance (QA), finaling, pre-launch demos, the launch, and post-launch support. We have summarized in a technical report [19] the use of `RTSenv`-like tools in the basic game development process and various adapted processes; we find that such tools are particularly useful for the critical production, QA, pre-launch, and post-launch support stages. A large project with one year of production and five years of post launch support would benefit from such tools for over three quarters of its duration, and in particular throughout its money-making post-launch stage.

D. Implementation and Adaptation

Our reference implementation of `RTSenv`, which is coded in Python and uses `wireshark` to record the network traffic, is portable and extensible. We have tested our implementation on various Windows, Ubuntu, and Red Hat Linux systems; on single desktops, on single and multiple clusters from the DAS-4 [14] multi-cluster/grid computing environment, and on the Amazon EC2 cloud computing environment.

Our implementation is extensible in the sense that a user can add more experiments, more performance metrics, and more statistical tools by simply adding Python components to the source directory. Adding in RTSenv support for OpenTTD required from us several modifications to the game packaging, but no changes to the game design or core implementation. First, we have modified the visualization of OpenTTD and add a non-graphical client mode into OpenTTD; this is needed for experimenting in cluster environments. Second, we have extended the original in-game limitation to the number of companies(one player per company) from under 16 to 250. Third, we have changed the operation of the OpenTTD's AI module to allow AI-controlled players to operate remotely from the game server; this effectively enables multi-machine game sessions and various scalability tests. Fourth, we have made use of the original debug utilities of OpenTTD to implement the replay functionality. Fifth, we have used the load map function of OpenTTD to verify the correctness of completed games and to obtain game-specific scores and data.

IV. Experimental Results

In this section we present the experiments we have performed using a reference implementation of RTSenv. Our experiments show evidence that RTSenv can be used in a variety of experimental scenarios and on a variety of computational platforms. Although made possible by RTSenv, a comprehensive evaluation of OpenTTD or building a performance model for RTS games are both outside the scope of this paper. Overall, we have conducted using RTSenv over 20,000 game sessions, which amount to over 30,000 in-game years and over 7,000 real operational hours. For more results and detailed analysis we refer to our technical report [19].

A. Experimental Setup

Experiment configuration We configure RTSenv to record the performance of CPU and memory load twice per second. Except for the experiments done for measuring network traffic, each experiment is repeated at least 30 times.

Experimental Environment Unless otherwise noted, we ran our multi-machine experiments using the DAS-4 six-cluster, wide-area system. A typical compute node of DAS-4 has a dual quad-core 2.4GHz Intel E5620 CPU and 24GB memory; the intra-cluster network is 1GBit Ethernet. DAS-4 uses the Sun Grid Engine batch queueing system to manage its resources; compute nodes are synchronized through the NTP protocol. The job execution model of DAS-4 assumes exclusive resource use. To avoid interference effects, when possible each game instance is allocated a single node of DAS-4; for example, when experiments do not focus on the network performance of the game.

Game Configuration All the experiments use Artificial Intelligence (AI)-controlled players; the AI algorithms (the AIs) are real, high quality, open-source, and community-provided. Unless otherwise specified, the game sessions are configured as in typical OpenTTD AI competitions [20]: default configuration of OpenTTD, except for the starting in-game time (1998) and initial capital (high). Each game instance is scheduled to run one in-game year by default, which leads to a *nominal execution time* of about 825 seconds on a typical DAS-4 node.

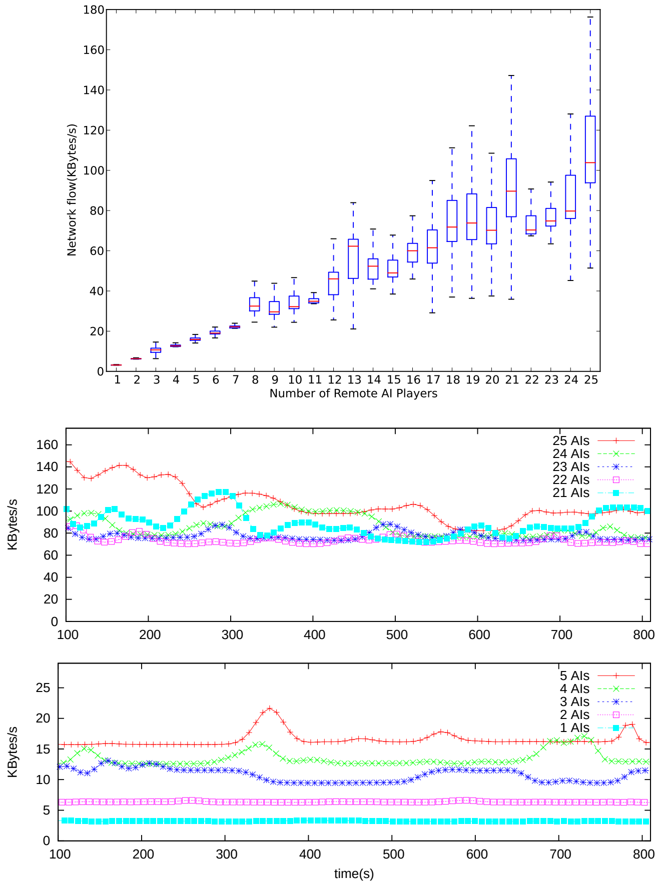

Fig. 2. Network traffic of the OpenTTD server for various remote player counts: (*top*) basic statistics, depicted as box-and-whiskers plots; (*middle and bottom*) consumption over time for various number of AIs.

Longer execution times, which result from server overloads, are noticeable to players only if they exceed 105% of the nominal execution times (the *playable execution time range*).

B. Performance Evaluation Results

Network Measurements, System (Figure 2): We conduct multi-player game experiments to assess network consumption. Each AI player connects to the server remotely, first to download the game map, then to issue commands and to receive small updates (for example, each command issued by the each other player)—this scenario emulates real gamers competing through the services of a commercial game server. The server is executed on desktop computer while the clients are executed on cluster's computer node. Figure 2 shows the latter type of network traffic, as observed for this scenario when the number of remote players is varied from 1 to 25. The statistical properties of the network consumption (Figure 2 (top)) indicate that the median network traffic size increases nearly linearly to the number of players (the *linear model* commonly assumed for games [21], [22]), however, wide value ranges of the traffic size indicates that a linear model may be inappropriate. Players with limited bandwidth may find that the real bandwidth consumption exceed significantly than the linear model's prediction. We now show that the wide value ranges also have long duration. The traffic over time in

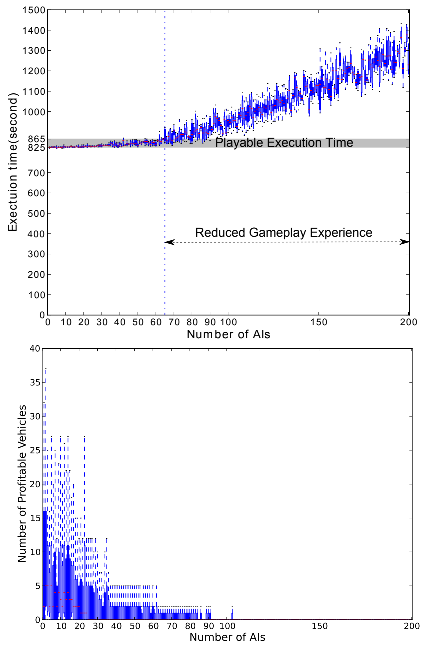

Fig. 3. System and in-game performance, respectively, for the `OpenTTD` server for various player counts. (*top*) game execution time; (*bottom*) number of profitable vehicles.

Figure 2 (bottom)–the network consumption is stable when the number of players(AIs) is low (up to 5 players), but increases and becomes more variable as the number of players increases. Noticeably, minute-long periods of much higher traffic than expected from the linear model occur even for 5 players (see corresponding curve ”5 AIs” between 300 and 400 seconds). Periods of such length will be noticed even by beginning players. *Main Finding: The linear model does not hold for RTS games in general* (we have just shown an example where it does not hold)*, and periods of high variability become common as the number of players increases.*

Scalability, System Performance and User Experience (Figure 3 (top) and (bottom), respectively): We assess the scalability of the game server by increasing the number of players from 1 to 200, that is, to an order of magnitude more players than in today’s commercial games. All the AIs are running on the game server, thus consuming CPU and memory resources—this is the typical mode of operation for current commercial games [21], [23]. The results are depicted in Figure 3. When the number of players is below 30, the game execution time is stable and very close to the nominal execution time (see Section IV-A). The playable execution time range is exceeded at about 60 players; afterwards, the game execution time increases quickly in both median value and range, which results in reduced gameplay experience. The game execution time growth is mainly due to CPU consumption; as shown in our technical report [19, Sec.4.2], the maximum memory consumption is below 700MB (`OpenTTD` is not memory-bound). We further analyze the results of the previous experiment from a gameplay experience perspective. Since in our tests we cannot ensure the presence and follow a group of human testers that adhere to the standards of gameplay experience evaluation [9], [10], we use instead a

TABLE I
CHARACTERISTICS OF THE CLOUD RESOURCE (INSTANCE) TYPES. THE ECU IS THE CPU PERFORMANCE UNIT DEFINED BY AMAZON.

Name	Cores (ECUs)	RAM [GB]	Archi. [bit]	Disk [GB]	Cost [$/h]
m1.small	1 (1)	1.7	32	160	0.1
m1.large	2 (4)	7.5	64	850	0.4
c1.xlarge	8 (20)	7.0	64	1,690	0.8

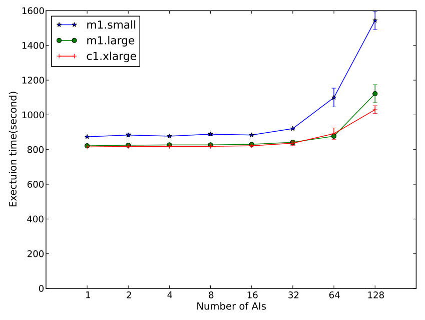

Fig. 4. Execution time for identically-configured game instances on various cloud resources.

proxy metric for gameplay experience, the median number of profitable vehicles. For scalable gameplay experience, the median number of (profitable) vehicles should be scalable, that is, it should at least not decrease when the number of players increases. When the number of vehicles is not scalable, that is, the number of profitable vehicles is low regardless of the player ability, the players experience a reduced feeling of mastery. A comprehensive user study focusing on the relationship between the number of vehicles and the players’ enjoyment falls outside the scope of this paper. Figure 3 (bottom) shows the statistical properties of the number of profitable vehicles as a function of the number of competing players. As the number of players increases, it becomes increasingly difficult for players to have profitable vehicles. Over 25 players, the median value of the number of profitable vehicles is 0 and the maximum value drops quickly below 5. The observed gameplay scalability limit (25 players) is below the system scalability limit (60 players for our platform) and does not depend on the platform; thus, *Main Finding: for (RTS) games, system scalability needs to be analyzed and improved in conjunction with gameplay experience scalability.*

C. Comparing Deployment Choices

Cloud resource performance Resources of various configurations (computational power, available memory, OS, preconfigured software such as MySQL, etc.) can be now leased from commercial cloud providers, such as Amazon, and used to deploy on them game servers. Cloud resources have preagreed leasing costs and operational/performance guarantees, expressed in public Service Level Agreements. Through virtualization, that is, a technology for emulating various computational machines on the same physical machine with little performance loss, a game server can be run without reimplementation or even re-compilation on a machine leased from a cloud. However, a game operator would still have to decide, from the many offers available, which suits best the game to be deployed. To emulate this situation, we ran performance experiments on various types of computing nodes leased from the Amazon EC2 commercial cloud, with the characteristics summarized in Table I. Figure 4 depicts the performance (execution time) of identically-configured game executed when run on three different cloud resources. The use of the most expensive resource in this test, `c1.xlarge`, has an impact on performance only for a large number of AIs (close to 128 and above). With this information, the game operator can implement various deployment strategies, from "lease the best resources" to "optimize the game performance/operational cost trade-off". *Main Finding: `RTSenv` enables the selection of resources for deployment.*

V. Related Work

We contrast `RTSenv` with previous experimental RTS/game studies, experimental RTS/game platforms, and RTS/game simulators, and others in our technical report [19]. In comparison with this body of related work, `RTSenv` has different scope (system and game-specific experiments), focus (RTS games), and application (a *real* game, `OpenTTD`, and many high quality artificial players.)

Few *experimental RTS game studies* exist. Closest to the experimental part of our work, a study on the prototype RTS game Rokkatan [2] tests scalability with about 400 clients, but the clients are controlled by prototype instead of production, that is, high-quality and CPU-intensive, AIs. *Experimental game platforms* exist [11], [12], [24], but they do not have RTS-specific support and the studied resource is mainly the network. *Simulation-based studies of online games* Simulators developed using standardized, industrial-grade simulation platforms such as HLA and the older DIS and FIPA, have been used to investigate online game-like scenarios [25]. A simulator co-developed by an author of this work was used [26] to study how data centers and cloud computing can support online gaming. A variety of peer-to-peer online game simulators exist [3], [4], [22], [27].

VI. Conclusion and Future Work

The increasing popularity of RTS games fosters demand for new designs and technical solutions, which emphasizes the need for experimental environments. In this work, we have introduced `RTSenv`, an experimental environment for RTS games that is useful for both researchers and developers. Our environment can control and measure many RTS-specific aspects, and enables a variety of experimental scenarios–from performance evaluations to taking game design decisions. `RTSenv` can operate in several types of computing environments, from single desktop computers to wide-area multi-clusters and commercial clouds, and leverages reactive fault tolerance techniques to perform robust, multi-machine, multi-instance RTS game experiments. We have used `RTSenv` in DAS-4, a real multi-cluster environment, and Amazon EC2, a commercial cloud provider, to conduct an extensive study of the popular RTS game `OpenTTD`.

The experimental results show that `RTSenv` can help e-valuate many RTS-specific features and can help comparing different game design choices. They also show that `RTSenv` can lead to new findings, such as:

1) Common assumptions made by researchers about game workloads, including the linear dependence between network traffic and the number of players, do not hold for RTS games in general;
2) The scalability of an RTS game should be evaluated not only from the performance but also from the gameplay experience perspective;

We are currently working on integrating a network emulator into the environment, and on comparing technical rather than game-oriented design choices (e.g., various algorithms for game state dissemination).

References

[1] Blizzard Inc., "StarCraft II: Wings of Liberty Becomes Biggest PC Game Launch of the Year," 2010, http://us.blizzard.com/en-us/company/press/pressreleases.html?100803.

[2] J. Müller and S. Gorlatch, "Rokkatan: scaling an rts game design to the massively multiplayer realm," *Computers in Entertainment*, vol. 4, no. 3, 2006.

[3] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "VON: a scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.

[4] A. R. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: enabling large-scale, high-speed, peer-to-peer games," in *SIGCOMM*, 2008, pp. 389–400.

[5] Nice Technology, "Picaroon, massively multiplayer strategy game," 2011, [Online] Available: http://www.picaroonthegame.com/.

[6] G. McAllister and G. R. White, "Evaluating user experience in games," R. Bernhaupt, Ed. London, UK: Springer Verlag, 2010, ch. Video Game Development and User Experience, pp. 107–128.

[7] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu, "The effect of latency on user performance in Warcraft III," in *NETGAMES*, 2003, pp. 3–14.

[8] J. L. Miller and J. Crowcroft, "The near-term feasibility of P2P M-MOGs," in *NETGAMES*, 2010, pp. 12–17.

[9] R. J. Pagulayan, K. Keeker, D. Wixon, R. L. Romero, and T. Fuller, "The human-computer interaction handbook," J. A. Jacko and A. Sears, Eds., 2003, ch. User-centered design in games, pp. 883–906.

[10] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. R. Wixon, "Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems," in *CHI*, 2008.

[11] S. Tolic and H. Hlavacs, "A testbed for P2P gaming using time warp," in *Transactions on Edutainment II*, ser. LNCS, 2009, vol. 5660, pp. 33–47.

[12] M. Lehn, T. Triebel, C. Leng, A. Buchmann, and W. Effelsberg, "Performance evaluation of peer-to-peer gaming overlays," in *P2P*, August 2010, pp. 1–2.

[13] M. Buro, "RTS Games as Test-Bed for Real-Time AI Research," *JCIS*, pp. 481–484, 2003.

[14] ASCI, "Distributed ASCI Supercomputer-Version 4," 2010, http://www.cs.vu.nl/das4/.

[15] OpenTTD team, "OpenTTD," 2010, http://www.openttd.org.

[16] H. Huang, *Logistics Operations Management Simulation Practice Guide (In Chinese)*. Higher Education Press of China, 2010.

[17] A. Rollings and E. Adams, *Fundamentals of Game Design*. Prentice Hall, 2006.

[18] T. Fields, *Distributed Game Development: Harnessing Global Talent to Create Winning Games*. Focal Press, 2010.

[19] S. Shen, O. Visser, and A. Iosup, "Rtsenv: An experimental environment for real-time strategy games on multi-clusters," TU Delft, Tech.Rep. PDS-2011-002, June 2011. [Online]. Available: http://www.pds.ewi.tudelft.nl/research-publications/technical-reports/2011/

[20] "TJIP OpenTTD Challenge," 2008, www.tjip.com/tjip-challenge.html.

[21] A. Shaikh, S. Sahu, M.-C. Rosu, M. Shea, and D. Saha, "On demand platform for online games," *IBM Sys. J.*, vol. 45, pp. 7–20, 2006.

[22] S. D. Webb, W. Lau, and S. Soh, "NGS: an application layer network game simulator," ser. IE '06, 2006, pp. 15–22.

[23] V. Nae, A. Iosup, and R. Prodan, “Dynamic resource provisioning in massively multiplayer online games,” *TPDS*, no. 3, pp. 380–395, 2010.

[24] J. Kienzle, C. Verbrugge, B. Kemme, A. Denault, and M. Hawker, “Mammoth: a massively multiplayer game research framework,” in *FDG*, 2009, pp. 308–315.

[25] P. Morillo, S. Rueda, J. M. Orduña, and J. Duato, “Ensuring the performance and scalability of peer-to-peer distributed virtual environments,” *Future Generation Comp. Syst.*, vol. 26, no. 7, pp. 905–915, 2010.

[26] V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. H. J. Epema, and T. Fahringer, “Efficient management of data center resources for massively multiplayer online games,” in *SC*, 2008, p. 10.

[27] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. P. Buchmann, “pSense - Maintaining a Dynamic Localized Peer-to-Peer Structure for Position Based Multicast in Games,” in *P2P*, 2008, pp. 247–256.