HyperVerse: Simulation and Testbed Reconciled

Jean Botev†, Markus Esch†, Hermann Schloss*, Ingo Scholtes* and Peter Sturm*
*University of Trier
System Software and Distributed Systems
D-54286 Trier, Germany
Email: {schloss,scholtes,sturm}@syssoft.uni-trier.de
†University of Luxembourg
Faculty of Sciences, Technology and Communication
1359 Luxembourg, Luxembourg
Email: {jean.botev, markus.esch}@uni.lu

***Abstract*—When dealing with dynamic large-scale topologies such as those underlying peer-to-peer (P2P) distributed virtual environments (DVE), one inescapably reaches the point where either a) simulations lack human behavior and assessment or where b) practical experiments on a small scale do not yield significant results. The restrictions resulting from the separation of simulation and testbed environments hinder a comprehensive assessment and efficient development of adaptive algorithms and techniques for DVEs as they are investigated in our HyperVerse research project. In this article, we present a hybrid evaluation system designed to combine the advantages of simulations and testbeds. The proposed infrastructure exhibits great flexibility particularly alluring in view of the multitude of potential research in the context of DVEs.**

I. Motivation

It is beyond controversy that the design of scalable DVEs is a challenging task. Their commissioning and maintenance being both laborious and costly, all concepts, topologies and algorithms utilized need to be rigidly tested and evaluated well in advance. For this purpose DVE designers and researchers usually rely on two separate strategies: Computer simulations are used to quickly obtain reproducible results, while small-scale testbed deployments are made to obtain a more realistic view under real-world conditions. Both approaches have unique advantages and disadvantages when evaluating massive-scale distributed virtual environments.

The foremost advantage of testbed deployments is the fact that they provide a realistic view of a system's performance under real-world conditions such as realistic network latency, congestion, user behavior and so forth. Unfortunately the reproducibility and tracing of results is hampered by the real world's intrinsic non-determinism and the scale is limited by the availability of physical devices. The fact that real user behavior is essential for realistic tests furthermore complicates the usage of research testbeds like e.g. PlanetLab[1]. The usability of practical tests in research is usually restricted by the comparably small number of available physical machines and staff. While this could theoretically be alleviated by including external testers, arousing the interest of a massive number of people has proven to be difficult. Furthermore, quickly obtaining results is prevented by the necessary deployment of software, and the development of test prototypes is complicated by the heterogeneity introduced by external resources. At this point, simulations provide adequate means to evaluate DVE concepts using a moderate amount of resources. Here the number of simulated entities can be varied within a wide range, being limited only by available processing capacities and time. Further advantages include that simulations offer the possibility to deterministically perform tests in a synthetic environment in which every aspect can be controlled by the experimenter. Algorithms, parameters and topologies can be compared under different conditions e.g. using different user mobility models, churn dynamics, simulated network latencies and so forth while leaving other simulation parameters unchanged.

Within the HyperVerse project [1], both simulations and practical tests have been performed in the past to evaluate its underlying concepts. Simulations have been performed using the TopGen environment [2], which is being developed within our group since 2006. Practical tests on small scales have so far been conducted using an actual DirectX-based implementation of the algorithms and topologies in question. During these tests, implementation and deployment has proven to be time-consuming, especially dealing with external testers and firewall traversal for peer-to-peer topologies.

Acknowledging the fact that both approaches, simulation and testbeds, offer unique opportunities, a combination of both seems alluring. Within our project the need for such a hybrid evaluation system has been additionally fueled by a special scenario: For a couple of years, our group teaches basic programming principles to secondary level pupils at the age 10 and above. For this we embrace both well-known concepts like simplified programming environments as well as novel approaches like learning and interacting in distributed virtual environments. The latter naturally raises the question whether both projects can synergetically benefit from each other by (a) enlisting the group of pupils as "testing personnel" and (b) using our DVE research prototype as software platform for teaching. In this article we discuss a hybrid evaluation system that has been developed for this purpose. It allows to use a generic user client implementation to be used for the evaluation of different algorithms and topologies without having to deploy any code changes. These actual clients are connected to a simulation environment which simulates actual communication and data distribution while possibly additionally simulating hundreds of more clients.

The remainder of this article is organized as follows: Having laid out the general architecture of the implemented system in section II, in the following section III we demonstrate how it has so far been used to evaluate concepts and topologies in our HyperVerse research project. After having discussed related work in section IV, we conclude with our contributions and propose the usage of our evaluation system within the DVE research community.

II. A Hybrid Evaluation Infrastructure

In this section, we seek to introduce the architecture of our hybrid evaluation infrastructure. Herein lightweight client

[1] http://www.planet-lab.org

applications can be attached to simulations running in a central simulation server in real-time. Actual users can interact with simulated entities and simulated entities can be programmed to react to user interaction. The central component of this system is TopGen, an open source simulation framework for graph and network mesoscopics that is being developed in our group since 2006. TopGen has so far proven to be capable of simulating systems as diverse as router topologies [2], self-organized monitoring schemes for complex networks [3], DVE topologies and algorithms [1], [4] as well as bio-inspired self-synchronization phenomena [5]. The TopGen simulation framework runs on top of the Microsoft .NET or MONO[2] runtime environment, the latter being an open source implementation of the ECMA/ISO standard for the Common Language Infrastructure (CLI). MONO is freely available for a multitude of operating systems and processor platforms. The basic architecture of TopGen is shown in Figure 1. In order to maximize platform independence and provide for a scripted batch operation, the graphical user interface (GUI) of TopGen has been decoupled from the actual framework. By this means, a memory-efficient command line version can be started without the GUI portions of the software. Furthermore it rids the core framework as well as the command line interface from any particular GUI library dependencies and thus furthers platform independence.

The main advantage of TopGen is that it offers a generic yet intuitive approach to network simulations based on so-called *Experimentation Modules*. These modules may contain topology generation schemes, metrics computations or simulations of complex network algorithms and can be freely combined in order to facilitate a "building blocks"-like definition of complex simulation experiments. Integrated data logging and plotting facilities as well as a real-time visualization component complement the environment's functionality.

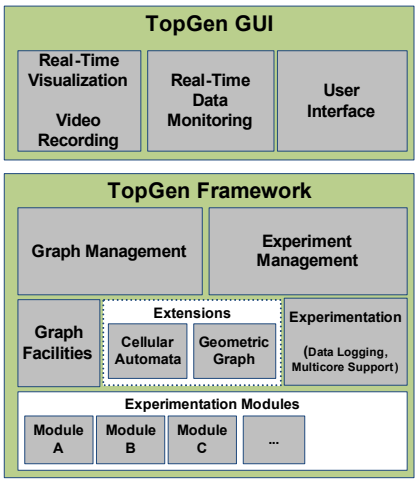

Fig. 1. Basic TopGen Architecture

At the very core of the TopGen framework stand its graph facilities. In this component, thread-safe abstractions of basic graphs, vertices and edges as well as their visualizable counterparts are implemented. In order to facilitate scalable experimentation modules, special support to utilize the inherent parallelism often found in network algorithms is provided. A special module allows to submit tasks that are processed per-vertex on all available processor cores in parallel. The main advantage of using this module is, that the implementor benefits from the future availability of additional processing cores without the need to explicitly deal with multi-threaded programming and synchronization issues. Switching between single-threaded or parallelized code is merely a matter of changing the header of a loop iterator. An important aspect when using multi-threaded simulations is to retain determinism. For this purpose, transaction-based graph and state abstractions are used. Changes performed by a certain thread are not made visible to other threads until all parallel tasks started in a specific simulation step have been completed.

The experimentation component provides a framework that can be used to implement custom experimentation modules. It provides distinct simulation modes for modules that (a) compute graph and network metrics (e.g. diameter, degree distribution, etc.), (b) hook into deterministic, discrete time-step simulations by subscribing to certain events (e.g. vertex added, vertex removed, simulation step, etc.) or (c) intend to perform self-paced (non-deterministic) simulations in a self-managed thread. The experimentation component also provides a framework that can be used to enrich modules with meta-information like e.g. which graph types a module should be allowed to operate on or how result data sets shall be interpreted. Based on the CLI's powerful reflection mechanism, user-implemented experimentation modules can be attached or detached to simulations dynamically at run-time without interrupting running simulations.

Apart from its core components, the TopGen framework contains a simple model that allows to extend the framework by custom graph-based abstractions. These can e.g. be used in own experimentation modules while harnessing the power of TopGen's existing components. So far, extension modules implementing abstractions for cellular automata, large Internet router networks and geometric intersection graphs have been implemented. Using a circle as geometric intersection primitive, the latter can e.g. be used to simulate mutual avatar visibility in DVEs. Finally the framework is complemented by two components managing the composability of experimentation modules as well as the loading and saving of graphs.

Although all experimentation modules combined to a certain simulation run share a common graph topology, situations with local and possibly inconsistent per-vertex graphs (as e.g. commonly found in P2P networks) can be simulated in different ways. Since TopGen supports directed graphs, it is simple to represent vertices with differing/inconsistent neighbor lists. Alternatively (e.g. if multiple views with inconsistent undirected edges are required) modules are free to additionally maintain and manipulate local per-vertex graphs.

From the perspective of a user willing to implement a custom experimentation module, its interface to the TopGen framework is of utmost importance. In section III it will be described in more detail by presenting an example experimentation module. By this means, the programming model underlying TopGen will be exemplified.

A. Simulation Mediation

So far, only TopGen's basic simulation facilities have been described. Through extending TopGen with a special experimentation module, a hybrid setting can be achieved in which TopGen additionally acts as a DVE server application. This special module - the so-called *Simulation Mediation* module - provides a network-accessible message sink to which actual 3D client applications can connect. The module's main task

[2]http://www.mono-project.com

is to mediate between incoming messages and simulation events to which other experimentation modules can subscribe. Furthermore, the Mediation module relays incoming messages to other clients based on the currently simulated network topology. By this means, any simulation implemented in an experimentation module (or a set of modules) can be enriched by actual clients that interact with the simulation in real-time. A basic example of such a hybrid simulation which will be described in more detail in section III is a setting in which a customizable number of simulated avatars (moving e.g. according to SecondLife[3] avatar trace files) together with a number of avatars controlled by actual users are used to simulate traffic from a P2P-based retrieval of 3D contents. A network view of such a hybrid scenario is shown in Figure 2. Here a number of simulated entities (denoted by the letter *S*) is enriched with four real clients A, B, C and D which connect to TopGen via its Mediation module.

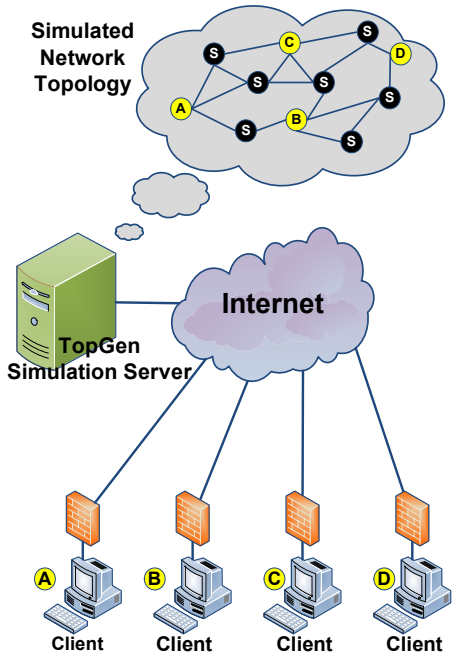

Fig. 2. Network Layout of the Hybrid Evaluation System. Simulated entities in the network topology are labeled with *S*, while representations of actual clients are labeled with their corresponding letter.

All messages received by the Simulation Mediation module are translated to graph events and potentially relayed to other nodes via the simulated network. The decision to which other clients a message is relayed can be based on a user-implemented routing strategy. Since TopGen has a notion of router-level graphs and contains a module for generating realistic router distributions, connectivity and latencies, artificial message propagation delays may be induced based on a customizable router-level graph. For this purpose either virtual overlay nodes can be connected to simulated routers or precomputed end-to-end latency matrices generated by TopGen or other tools can be used. So far, no technical details of the 3D browsing application have been given which can be used to actually connect to and participate in a simulation. In the following section, we intend to provide a better understanding of its architecture and its underlying principles.

[3]http://www.secondlife.com

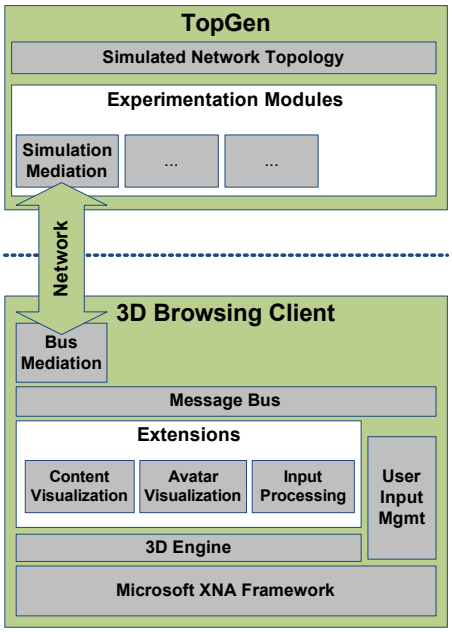

Fig. 3. Basic Architecture of the 3D Browsing Client

B. 3D Browsing Client

The 3D browsing client is an important component in the overall evaluation system. It has been implemented on top of the Microsoft XNA Framework[4]. As described above, its main purpose is to enable actual users to connect to a simulated DVE. As such, it might also be considered an external 3D visualization component which can be used to jump into a running TopGen DVE simulation. In fact, the main idea behind the 3D browsing client's architecture is to provide for its flexible usage. We tried to achieve this goal by not implementing a particular 3D browser but rather providing a generic framework in which modules can be used to implement various browsing applications. For this purpose custom-defined modules can be used to customize every aspect like e.g. content visualization, generated messages or user controls. Another important aspect is that the browser shall not contain any code specific to a certain DVE algorithm or topology. By this means, no redeployment is required when other topologies or algorithms needs to be used. Its basic functionality is to provide an infrastructure that allows a translation of user-input to messages sent to TopGen's Simulation Mediation module as well as providing a 3D visualization of the actual environment based on messages relayed by TopGen. The advantage of sending all messages to the central TopGen simulation server is that the relaying of messages can be left to TopGen. By this means, the actual topology by which both real and simulated clients are interconnected can be defined within a TopGen experimentation module. Due to the dynamism of TopGen, this module can even be switched at runtime without having to interrupt the simulation or stop actual clients. Having motivated its basic underlying principles, the basic architecture of the 3D browsing client is shown in Figure 3.

The so-called *Message Bus* is the browser's basic communication abstraction by which all of its remaining components are interconnected to each other as well as to the TopGen

[4]http://www.xna.com/

server. All messages on this bus are typed via descriptive XML contracts and categorized into so-called *Local* and *Remotable* messages. In addition to being delivered locally, a special *Bus Mediation* component automatically serializes Remotable messages and forwards them to the *Simulation Mediation* module in a running TopGen simulation via a TCP connection. As a general rule, messages that affect other avatars and components within the DVE should be defined as Remotable, an example for this being messages related to avatar movement or interaction. The decision whether a message type should be Remotable is however left to the implementor of the type. Locally, messages are transmitted between software components via the CLR's efficient eventing mechanism. A special user-input component translates input like e.g. pressing a key or moving the mouse to messages and transfers them to the Message Bus. Another (customizable) component can subscribe to these events, e.g. locally visualize avatar movement and produce an appropriate Remotable message to notify nearby actual or simulated clients.

Regarding the visualization of content, again a customizable path has been taken. Rather than predefining content types, custom modules can be used to visualize any XNA-renderable content based on their current state and incoming messages. The source of this content can be chosen by the module, i.e. either a local resource (in scenarios where content is predistributed), the payload of a message routed via TopGen or an arbitrary network accessible external source. The full power of this approach becomes apparent when looking at typical scenarios occurring in the DVE research community. Here, often research prototype clients are deployed on a small scale either for the interested general public, the testing community or collaborating partners. With our centralized hybrid simulation/testbed approach, such small-scale deployments (in a range between dozens to a few hundred concurrent users) are simplified in the following ways: Testing different network topologies and overlay topologies does not require a redeployment of the clients or even the interruption of the service or simulation. It is a mere matter of activating another experimentation module in the TopGen simulation server. Furthermore, results of an experiment can easily be collected and the state of the system can be visualized by a central instance in real-time. Video-recording facilities that are integrated into TopGen's visualization component can even be used to record the global behavior and mobility of users. When addressing P2P topologies for DVEs, another important issue is the handling of end-user firewalls. Another advantage of a TopGen-based small-scale deployment of a DVE prototype is the fact that the clients' only communication channels are client-initiated TCP connections. Since the same connections are used by the TopGen server's Simulation Mediation module to relay messages from peers, firewall traversal is not an issue.

III. A Case Study

So far the TopGen simulation framework has been described at a mere abstract architectural level. At this point, the reader is probably interested in a more detailed case study exemplifying the programming model underlying TopGen's experimentation modules. For this, we describe a module that has been implemented in the HyperVerse context in order to simulate the speculative hoarding approach to content retrieval that is described in [3]. In order to be able to follow the implementation details, the scheme will be briefly described in the following paragraph. The source code of this module is included in the freely accessible TopGen repository which is available from our project's website [5].

1) Epidemic Hoarding: Hot spot regions arising from high object densities represent a fundamental problem in distributed virtual environments. One facet of this problem is the fact, that clients entering hot spot regions may be required to load a large body of data within a short time-frame. This problem is independent from bandwidth available at the content provider. The limited bandwidth of the client's network connection represents a bottleneck that can prevent timely data availability and thus lead to notable load delays. In order to mitigate this problem, a hot spot aware prefetching scheme has been developed. The basic idea is to constantly use a fixed fraction of free network bandwidth for hoarding data selectively from crowded regions in order to mitigate load delays when the user actually enters them. The problem with this approach is that clients need a constantly updated view of the DVE's current load distribution. In [3], a self-organized and scalable solution based on fixed-size random information exchanges with a single one-hop neighbor has been presented. This can be achieved by considering objects and avatars as physical particles having a certain mass that reflects the communication effort involved when loading the object. Each client can simply calculate the cumulative mass and the center of mass within its own area of awarenesses. This information, along with a list of preliminary "most crowded regions" based on the locally available information can then be exchanged with a random neighbor. Based on the epidemic aggregation algorithm presented in [6], list entries quickly converge towards the actual global maxima. In [3] it has been shown that this hoarding mechanism is able to efficiently mitigate load delays in hot spot regions.

2) Simulating Epidemic Hoarding: Based on the concept that has been described in the previous paragraph, in this section we intend to describe how the scheme can be simulated in TopGen. The starting point for the implementation of a TopGen experimental module is to extend the abstract framework class *TopGen.Experiments.ExperimentalModule*. Two special methods *Start* and *Stop* will be invoked by the framework when (a) a deterministic, event-based module is attached to or detached from a simulation or (b) when a non-deterministic simulation module operating in a custom thread shall start or stop operation. By assigning framework-defined attributes to the implemented module, the implementor can explicitly tell TopGen which kind of threading mode is desired for the module. For the epidemic hoarding simulation, we seek to utilize the event-based mode which hooks into discrete events generated outside the module. This mode's main advantage is being deterministic in the sense that the same random seed will generate exactly the same sequence of events. Accordingly in our case we would assign the attribute

[ExperimentType(ExperimentType.Event)]

to the module implementation. The signature of the *Start* method that can be implemented to perform custom module initialization reads:

Start(ref Graph g, Settings s, SimulationContext context);

In the first argument, a graph is given on which the experimentation module can operate. In the case of the epidemic hoarding scheme we seek to implement, this is a collection of vertices representing clients along with their avatars' position in the DVE. An edge between two vertices represents a con-

[5] http://hyperverse.syssoft.uni-trier.de

nection in the underlying network topology. It is not necessary to define the exact topology in the epidemic hoarding scheme. The topology can either be implemented in a separate module or TopGen's integrated geometric graph abstraction can be used. In the latter case, avatars within a certain visibility range are automatically interconnected by the framework. By default, the graph that is passed to a simulation module can be of any type inheriting the framework's basic *Graph* class. Often a module implementor wishes to restrict the types of graphs on which the module can be started to a certain subset. In our case, we need a notion of a geographic position for each vertex. This can be enforced by assigning the following attribute

[AcceptedGraphType(typeof(GeometricGraph))]

which restricts the application of the module to graph types that are subclasses of the framework's basic geometric graph. To implement modules that initially generate graphs (like in our case a geometric graph consisting of a customizable number of randomly placed vertices), one can explicitly allow TopGen to pass an initial *null* reference by attaching the attribute *[AcceptsNullGraph(true)]*. A module can then create an arbitrary graph and pass the result of this generation back to the environment where downstream modules can use them for simulations. A module for generating arbitrarily sized random geometric graphs is included in TopGen and can be used to create the initial graph for our epidemic hoarding simulation. In the remainder of this section, a vertex in the graph will be denoted as simulated client. Depending on a graph's type, the graph instance as well as the vertices provide certain events that can be used to react to simulation events. An example for such an event in geometric graphs is the vertices' *OnMove* event, which will be fired whenever the geographic position of a vertex changes. The epidemic hoarding scheme needs to subscribe to this event in order to update a list of objects in the client's area of interest whenever the client's avatar has changed position. Technically, by subscribing to an event, a function pointer (or delegate) is passed to the eventing infrastructure of the CLI that will be used to invoke a user-defined handler whenever the event occurs.

The second argument in the signature of the *Start* method is an object containing custom settings for a module. The exact type of this object can be specified by the implementor of a simulation module, all user-defined public fields serving as simulation parameters that can be dynamically set and changed via the TopGen GUI. In the case of the epidemic hoarding simulation module, parameters contain the number of simulated objects, parameters influencing their distribution and sizes, the clients' bandwidths and so forth. Based on these initial settings, modules usually need to perform some kind of initialization before the simulation starts. In the epidemic hoarding case, this e.g. involves the creation of a number of objects with their corresponding simulated transmission sizes. In order to simulate data transmissions, the module uses per-vertex download queues and caches that can take individual object data blocks. The notion of objects, corresponding data blocks and the implementation of a caching scheme has been implemented in a dedicated TopGen extension and can be used by simulations right away. A client's downlink bandwidth can be simulated on a per-step basis by moving a limited number of object data blocks from the download queue of a vertex to its cache.

The third and final argument of the *Start* method is a reference to the so-called *SimulationContext*, an object which - amongst other things - provides the simulation's unique time frame by producing tick events in adjustable intervals. This time frame is the same for all modules attached to the same simulation. Another important task of the *SimulationContext* is the provision of a transaction-based storage for arbitrary data that can be shared across modules. To retain determinism in multi-threaded simulations, changed data are made visible to other modules not before all parallel processing threads started in a certain simulation step have been completed. In order to perform a computation in each discrete time step of a simulation, modules can subscribe to a special *OnTick* event. For the epidemic hoarding module, in each step several actions are performed for each vertex in the graph (and thus DVE client):

- A random neighbor is determined with which information on cumulative particle mass, center of mass and a list of most-crowded spots is exchanged and aggregated.
- A number of data blocks limited by the client's per-step bandwidth is moved from the download queue to the cache. Blocks in the download queue are prioritized based on the object's distance to the client's current position.
- In case there is remaining bandwidth, the hoarding of object data blocks from within predicted hot spot areas is simulated in the same way as described above.
- For both previous steps, a per-vertex (least recently used) caching strategy is used

Finally, the *Stop* method needs to be implemented in order to unsubscribe from the vertices' *OnMove* and the *SimulationContext*'s *OnTick* events.

3) Composability with other modules: So far, the epidemic hoarding scheme has been implemented to operate on a geometric graph created outside the module and passed to it when the simulation starts. It has not yet been described where either the network topology or the mobility of (simulated) avatars is implemented. At this point, it is important to note that both is independent from the epidemic hoarding module. In a hybrid scenario, both might stem from the Simulation Mediation module, i.e. real clients joining the system will trigger the creation of vertices and actual user-input will be reflected in vertex mobility. In scenarios with simulated users, both needs to be done by other modules that can be attached to the simulation and which share the same *SimulationContext*.

Simulated mobility can be implemented by modules which - in each simulation step - change the position of each vertex of the geometric graph according to a certain model. While custom models can be implemented in a similar fashion as described in the previous paragraph, there are a couple of pre-defined models that can be attached to simulations right away. In addition to the well-known random waypoint model, and a (probably more realistic) model in which vertices move towards crowded regions with higher probability, TopGen also comes with a module that allows to replay avatar trace files from SecondLife that have been recorded and made available to the public in [7].

4) Evaluation: Another important aspect of implementing simulations in TopGen is data evaluation. For this, the framework provides a special component that can be used to log result data sets produced by experimentation modules. Logged results can automatically be written to log files or plotted in real-time. Technically experimentation modules can create custom-defined result structures and post them to the analysis component where they will be buffered. In the epidemic hoarding example, the module can compute e.g. in each step the number of objects in the field of view for which not all data blocks reside in the client's cache, a number representing

perceivable load delays. At the end of the simulation step, the current result structure can be posted to the environment via a special call. The contained result data fields will be accessed by TopGen via the CLI's reflection mechanism.

IV. Related Work

PlanetLab and G-Lab [6] are research testbeds supporting the development of new network technologies such as distributed storage, network mapping, P2P systems, distributed hash tables, and query processing. However, just as in simulations, user behavior can only be simulated since tests run in an unattended fashion. Furthermore, access to those testbeds is limited and the deployment is comparably laborious.

Our hybrid approach has been inspired by the JAVA-based simulation platform JANE [8], which has been developed in our group for a past research project. The JANE software can be used in a pure simulation mode, in a hybrid setting with real devices being attached to a running simulation and, finally, in a setting using real devices only. Developed to support ad-hoc network researchers in application and protocol design it addresses a different application scenario though. Other hybrid approaches in the domain of sensor networks include e.g. the system described in [9].

The authors of [10] describe the Generic Visualization System (GVS), which combines physical simulations and a real-time visualization to display results in a realistic 3D environment. For this, it can also merge inputs from different simulation programs running simultaneously. Developed with a clear focus on simulating military operations and without notion of a customizable underlying network topology, it is however not applicable in our scenario.

V. Conclusion

In this article, we described a hybrid evaluation infrastructure that combines simulations of a large number of entities with a small-scale testbed deployment. At this point we must emphasize that our evaluation system is not intended to be a scalable DVE hosting solution for large-scale deployments. Although we are currently looking into the possibility to support a larger amount of users by adding cluster support, the current centralized approach limits the number of connectible actual clients to typically a few dozens or a few hundreds (depending on the complexity of the simulated topology, data traffic and used hardware). The main intention of our system is to provide a rapid prototyping and simulation tool for DVE implementors and the scientific community alike in which simulations of massive numbers can be enriched with a moderate number of actual clients and vice versa. The main advantages of such a hybrid approach are the following:

- It saves implementation efforts since modules implemented for simulation can be used to drive actual experiments and vice versa. Furthermore, the implementation of topologies for testbeds is simplified by TopGen's high-level graph and communication abstractions.
- Actual 3D clients can be used to get a first-hand view of a simulation in real-time. They might thus also be seen as 3D visualization clients that can be attached to a running simulation, possibly without interfering with it.
- Especially in the context of DVEs, there are issues that are hard to evaluate by metrics, like e.g. the users' perception of certain latency characteristics, inconsistent views, load delays, etc. A hybrid mode provides the possibility to simulate a large back-end system while letting actual users judge the perceived quality of service.
- It allows to easily extract traces of real user mobility and behavior as well as to study the interdependencies of simulation mobility and user behavior. In this respect, TopGen provides means to extract and validate new mobility and behavioral models.

Finally, in the following we intend to summarize some technical and practical advantages of the proposed system.

- By disabling the possibility to attach real clients, TopGen can be used in a pure simulation mode with simulations being deterministic in the sense that any run can be reproduced exactly. In this mode, TopGen can handle network topologies with as much as several hundred thousand simulated nodes.
- Due to TopGen's modular architecture, avatars using simulated mobilities based on models (like e.g. random waypoint) can easily be combined with trace-driven avatars.
- TopGen can be run in a pure testbed mode if no simulated entities are used. In this case, extensive logging, tracing and evaluation is facilitated by the fact that all clients connect to a central machine. When using P2P topologies, messages are routed within a virtual network controlled by user-implemented modules. In this case, TopGen acts as gateway to the simulated network and performs routing based on the simulated topology. While it is clear that the centralized approach sacrifices scalability, for the targeted number of attached real clients we argue that the advantages outweigh this disadvantage.
- In a hybrid mode, real clients can be combined with simulated entities (based on mobility models and/or mobility traces).
- In all of the above modes, the same implementation of algorithms and topologies can be used.
- In all modes, simulated topologies, algorithms as well as any dependent parameters can be changed in real-time without interfering with users currently in the system.
- Clients can be deployed without dealing with complex firewall traversal while at the same time utilizing simulated P2P topologies.
- The simulation framework provides support for multi-processor platforms while retaining the determinism of single-threaded simulations by means of a transactional memory abstraction.

We conclude this article by pointing out some open issues. Although all components of the infrastructure (the TopGen simulation environment, the Simulation Mediation module as well as the 3D browsing client) have been implemented and tested, an extensive performance evaluation must be considered future work. So far, only TopGen's scalability on multi core processors has been evaluated and found to be linear for typical network algorithms. Further studies regarding the exact amount of supported users and simulated agents as well as its interdependency with the simulated network topology are required to prove the suitability in practical scenarios. Regarding our hybrid approach to DVE evaluations, the interrelations between simulated and real agents as well as their influence on realism require further attention.

All components of our proposed infrastructure are available both as source code and binaries from our project's website. At this stage we would like to encourage the community to test and evaluate our toolkit. We also actively solicit feedback and suggestions for its future development.

[6] http://www.german-lab.de

REFERENCES

[1] J. Botev, M. Esch, A. Höhfeld, H. Schloss, and I. Scholtes, "The hyperverse - concepts for a federated and torrent-based "3d web"," The 1st International Workshop on Massively Multiuser Virtual Environments (MMVE), 2008.

[2] I. Scholtes, J. Botev, M. Esch, A. Hoehfeld, H. Schloss, and B. Zech, "Topgen - internet router-level topology generation based on technology constraints," in *Proceedings of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools)*, February 2008.

[3] I. Scholtes, J. Botev, M. Esch, H. Schloss, and P. Sturm, "Minimizing load delays in distributed virtual environments using epidemic hoarding," in *Proceedings of the 4th International Conference on Collaborative Computing (CollaborateCom)*, November 2008.

[4] I. Scholtes, J. Botev, M. Esch, A. Höhfeld, and H. Schloss, "Awareness-driven phase transitions in very large scale distributed systems," in *Proceedings of the Second IEEE International Conferences on Self-Adaptive and Self-Organizing Systems (SaSo)*. IEEE, 2008.

[5] I. Scholtes, J. Botev, M. Esch, and P. Sturm, "Epidemic self-synchronization in complex networks," in *Proceedings of the 1st International Conference on Complex Sciences: Theory and Applications*, 2009.

[6] M. Jelasity and A. Montresor, "Epidemic-style proactive aggregationin large overlay networks," in *Proceedings of The 24th International Conference on Distributed ComputingSystems (ICDCS 2004)*. Tokyo, Japan: IEEE Computer Society, 2004, pp. 102–109. [Online]. Available: citeseer.ist.psu.edu/jelasity04epidemicstyle.html

[7] H. Liang, I. Tay, M. F. Neo, W. T. Ooi, and M. Motani, "Avatar Mobility in Networked Virtual Environments: Measurements, Analysis, and Implications" *arXiv:0807:2328v1*, 2008.

[8] D. Görgen, H. Frey, and C. Hiedels, "Jane-the java ad hoc network development environment," in *ANSS '07: Proceedings of the 40th Annual Simulation Symposium*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 163–176.

[9] S.-H. Lo, J.-H. Ding, S.-J. Hung, J.-W. Tang, W.-L. Tsai, and Y.-C. Chung, "Semu: A framework of simulation environment for wireless sensor networks with co-simulation model," in *Advances in Grid and Pervasive Computing*. Springer Berlin / Heidelberg, 2007, pp. 672–677.

[10] C. Holmes, J. Wolff, D. Challou, and P. Huang, "Generic Visualization System: The Link Between Digital Simulations and a Virtual Environment," in *Virtual Concepts 2005, Biarritz, Frannce*, 2005.