SimCon - A Simulation and Visualization Environment for Overlay Networks and Large-Scale Applications

Markus Esch
Faculty of Sciences, Technology and Communication
University of Luxembourg
1359 Luxembourg, Luxembourg
markus.esch@uni.lu

Hermann Schloss
System Software and Distributed Systems
University of Trier
D-54286 Trier, Germany
schloss@syssoft.uni-trier.de

Ingo Scholtes
System Software and Distributed Systems
University of Trier
D-54286 Trier, Germany
scholtes@syssoft.uni-trier.de

Jean Botev
System Software and Distributed Systems
University of Trier
D-54286 Trier, Germany
botev@syssoft.uni-trier.de

Alexander Höhfeld
System Software and Distributed Systems
University of Trier
D-54286 Trier, Germany
hoehfeld@syssoft.uni-trier.de

Benjamin Zech
System Software and Distributed Systems
University of Trier
D-54286 Trier, Germany
zech@syssoft.uni-trier.de

ABSTRACT

A substantial number of massive large-scale applications require scalable underlying network topologies. Nowadays structured Peer-to-Peer overlay networks meet these requirements very well. But there is still a need to decide which of these overlay networks is most suitable for providing the best possible performance for a certain application. This paper describes SimCon - a simulation environment for overlay networks and large-scale applications. SimCon allows the comparison of different overlay networks with respect to predefined metrics derived from requirements of the considered application. This approach allows determining which overlay network meets the needs of a given application best, which in turn is a great support for developers of large-scale applications.

Categories and Subject Descriptors

I.6.3 [**Simulation and Modeling**]: Applications

General Terms

Experimentation, Measurement

Keywords

Simulation Framework, Peer-to-Peer Overlay Networks, Large-Scale Applications

1. INTRODUCTION

Distributed large-scale applications make high demands on underlying network topologies in terms of latency, scalability, reliability and performance. For example, think of a massive multiplayer online game (MMOG) where every movement of an avatar has to be propagated to all the other avatars in virtual proximity. The virtual proximity comprises all avatars being at a certain place in the virtual world, whereby the players may actually be distributed all over the real world. That is, there is just a weak correlation between virtual and real proximity. Assuming a huge amount of moving avatars in a dense virtual neighborhood we recognize that latency of movement notifications may become a crucial problem affecting the game quality and user experience. This in turn has a wide influence on the popularity and the success of the game. Moreover, because of the increase in popularity of multiplayer online games observed in the recent years, the number of people playing such games is expected to surge. This rise would make it impossible to manage these kind of games centrally and causes the necessity of underlying scalable distributed network technologies. The problems outlined above highlight the importance of addressing the question of finding a feasible underlying network technology. We argue that this technology is the key to the development of scalable and performing large-scale applications. The claim that MMOGs have become a driving factor for development of large-scale distributed environments, made in [23], supports our argumentation.

Structured Peer-to-Peer overlay networks[1] such as Chord [21], Pastry [19], Tapestry [24], CAN [18], etc. inherently meet the outlined requirements of distributed large-scale applications. As opposed to earlier unstructured Peer-to-Peer networks they all provide very efficient publish/lookup mechanisms. That makes them considerable for latency crucial applications. All overlay networks perform the same tasks in a distinguished way, setting up a trade-off between

[1]An overlay network is a logical communication structure that is built on top of a physical underlying network.

performance and cost [10]. Therefore there is a need to compare them with each other in order to decide which of them is most suitable for a given application. This decision has a huge impact on the performance and quality of the application itself.

We propose an approach for monitoring simulated large-scale applications running on top of different overlay network technologies (see sec. 2). In the course of the simulation we gather certain statistical information, e.g. overall number of messages required for a multicast, and evaluate this information on the basis of predefined metrics derived from requirements of a given application (see sec. 4.4.1). By means of evaluation results, we can figure out which overlay network technology suits the requirements of a given application best.

In order to facilitate the task of selecting the appropriate overlay network technology, we developed *SimCon*, a generic and extensible simulation environment for distributed overlay networks and large-scale applications (see sec. 4). SimCon, which is written in *C#*, provides four extensible programming interfaces:

- `IApp` - An application level API providing an interface for development of large-scale applications (for details see sec. 4.3.1).
- `ITopology` - A network API offering an extensible interface for overlay network implementations (see sec. 4.3.2).
- `IRecorder` - A recorder API designed for capturing user-defined information (e.g. sending events). This interface is described in section 4.4.1
- `IView` - A view API needed to create a customized view to graphically demonstrate the behavior of the application (sec. 4.4.2).

SimCon uses the Internet Topology Generator TopGen (sec. 4.1) for creating underlying Internet topologies. By this means SimCon is able to perform the simulation of an overlay network with exchangeable Internet topologies and variable network sizes providing very realistic results.

Unlike the most comparable simulation environments considered in section 5, SimCon has a strong focus on the analysis and evaluation of overlay network characteristics and in particular on their conjunction with distributed largescale applications. This makes SimCon interesting for use in research projects.

2. MOTIVATION FOR SIMCON

In a current research project we intend to provide an efficient global-scale Peer-to-Peer overlay topology. We have developed SimCon to attain the ability to verify and compare different existing and newly-developed topology approaches with respect to different application scenarios. A simulation environment tailored for large-scale Peer-to-Peer overlay networks and applications based on realistic Internet router graphs generated by TopGen. The SimCon simulation environment is a useful tool for developers of distributed applications. Using SimCon, it is possible to test different overlay networks in order to figure out which one is the most suitable for their application scenario. The overlay network algorithms have to be implemented in a self-contained unit called topology module. Each topology module is defined in a single assembly and has only weak dependencies to SimCon. In this way, the topology modules are reusable in real world applications. In section 4.3 we describe in detail how the independence of the topology modules from SimCon is achieved using a so-called communication shim.

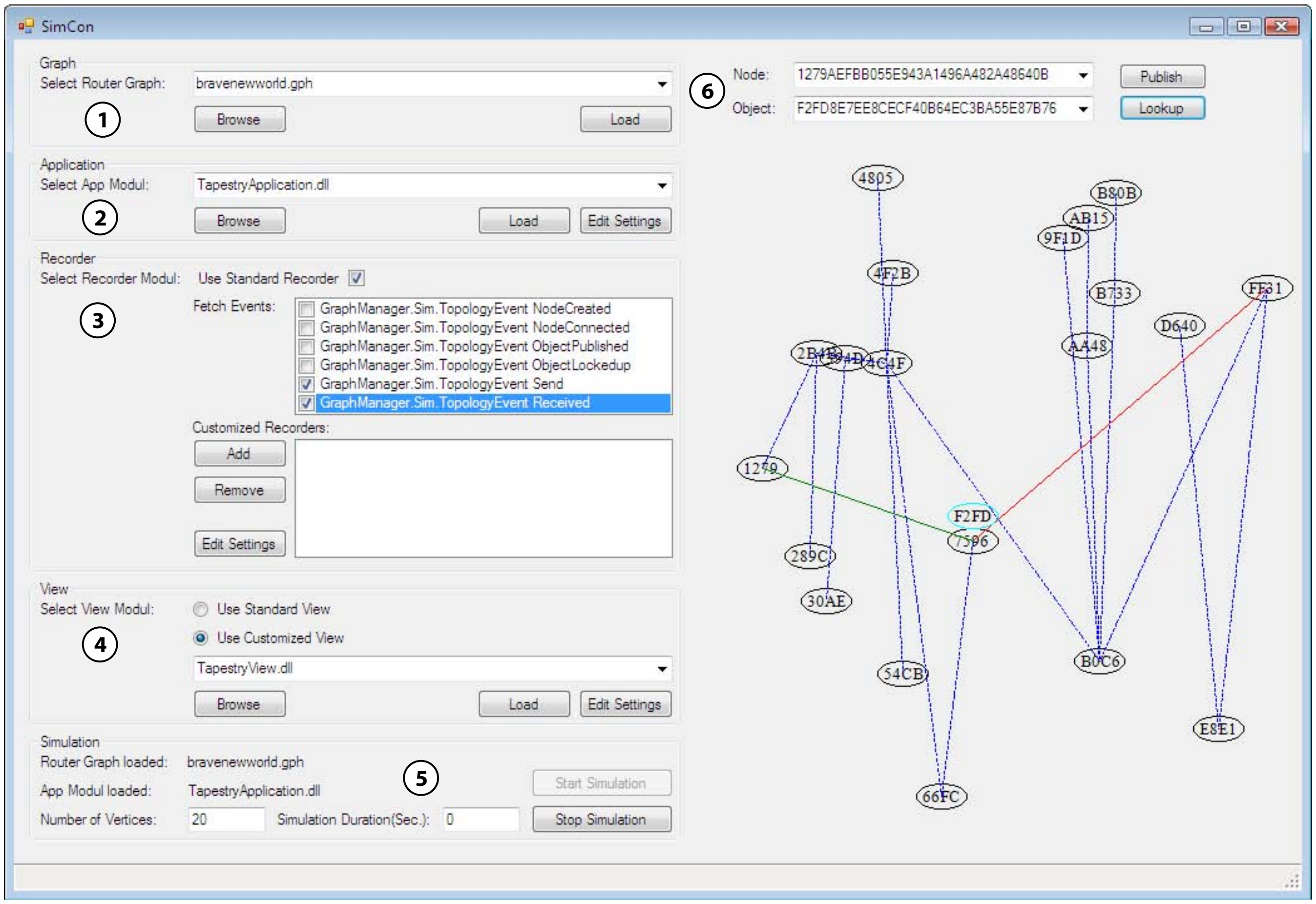
In order to obtain realistic simulation results, it is important to perform the simulation on top of a router graph that is as close to reality as possible. Unlike other overlay network simulators SimCon benefits from the realistic and detailed Internet router graphs generated by the topology generator TopGen. TopGen follows the latest results on the Internet topology presented in [11] and provides the possibility to define different router types and properties like bandwidth and latency for each type. Furthermore, we can define how different router types have to be interconnected. It is, for example, possible to differentiate between core routers that build the Internet backbone and DSL routers that are located at the edge of the Internet to serve as gateway for end hosts. By this means the real Internet topology can be imitated very well. Duo to the fact that SimCon is strongly connected to TopGen we can expect realistic simulation results. Section 4.1 provides further information about TopGen.

Another important feature of SimCon is its extensibility. Thereby the simulation environment can be adapted and extended in many ways. Hence SimCon is not limited to the simulation scenarios that were considered at design time, rather the simulation environment can be customized to the requirements of a simulation. In order to achieve extensibility, customized recorder- and view modules can be loaded dynamically at runtime from an assembly. In section 4.4 we present how to implement customized modules.

One advantage of SimCon is its modular design. This means that the applications simulated by SimCon have to consist of two self-contained parts: An application module and a topology module. An application module emulates the behavior of a certain application. A topology module contains the implementation of an overlay network algorithm. Communication between both modules proceeds via the interface `ITopology`. Hence application module and topology module are independent of each other. By this means it is possible to implement one application module and test it with different overlay networks by interchanging the topology modules. Consequently it is easy to verify which topology is most suitable for an intended purpose.

Miscellaneous metrics can be used in order to compare the performance of different algorithms. For example we can count the number of sent unicast messages to compare two algorithms. SimCon provides a simple mechanism for counting events that occur during the simulations. But even more sophisticated metrics can be accomplished in SimCon by implementing a customized recorder.

Typically, each overlay network algorithm has some parameters that can be used in order to adjust the behavior of the algorithm. These parameters may have a strong impact on the performance of the algorithm. To allow the testing of an algorithm with different parameters, SimCon provides a mechanism to modify this parameters before a simulation runs. SimCon reads the parameters of a given application via .NET Reflection and allows the user to change them at runtime.


SimCon

Graph
Select Router Graph: bravenewworld.gph

Application
Select App Modul: TapestryApplication.dll

Recorder
Select Recorder Modul: Use Standard Recorder [✓]
Fetch Events:
- [ ] GraphManager.Sim.TopologyEvent NodeCreated
- [ ] GraphManager.Sim.TopologyEvent NodeConnected
- [ ] GraphManager.Sim.TopologyEvent ObjectPublished
- [ ] GraphManager.Sim.TopologyEvent ObjectLockedup
- [✓] GraphManager.Sim.TopologyEvent Send
- [✓] GraphManager.Sim.TopologyEvent Received

Customized Recorders:

View
Select View Modul: Use Standard View [✓ Use Customized View]
TapestryView.dll

Simulation
Router Graph loaded: bravenewworld.gph
App Modul loaded: TapestryApplication.dll
Number of Vertices: 20
Simulation Duration(Sec.): 0

Node: 1279AEFBB055E943A1496A482A48640B
Object: F2FD8E7EE8CECF40B64EC3BA55E87B76

Figure 1: SimCon Simulation Environment

3. SIMULATION EXAMPLE

In this section we describe the handling of the SimCon environment using a simple publish/lookup application as example.

Figure 1 shows the Graphical User Interface (GUI) of the SimCon environment. The GUI is divided into two parts. On the left hand side we can see the management and configuration console. Here it is possible to dynamically load the following modules:

- An Internet topology graph
- An application to simulate
- Recorders to capture information
- A view module to display the course of the simulation process

Moreover we can define how many application endpoints should be initialized and how long the simulation should last. The space on the right hand side is allocated for the loaded view module.

In the following we describe how to setup a simulation step by step: First of all we have to load an Internet topology graph generated by TopGen (see fig. 1 label (1)).

Afterwards we load an application module (see fig. 1 label (2)). As mentioned above in our example we load a simple application whose functionality is restricted to publish and lookup operations. After the application module is loaded we can configure the application by editing its settings. SimCon loads the settings dynamically from the application module by using .NET Reflection.

Each application can provide miscellaneous events of the *TopologyEvent* type. These events can be selected for capturing by the standard recorder (see fig. 1 label (3)). Besides the standard recorder, an arbitrary number of customized recorders can be attached to the simulation (fig. 1(3)). Similar to the application module each recorder has certain settings which can be modified via the user interface. As described in subsection 4.4.1 the use of a recorder is mandatory in order to receive the required information about the application's behavior and to enable the evaluation.

It is very important to use a visualization module, in particular in an educational context. For example, it can be used to facilitate the comprehension of application's behavior and of underlying overlay network algorithms. We can load a visualization module as shown in figure 1 label (4).

The loaded view is displayed on the right hand side of the GUI. A view can define certain interaction elements used to gear into a running simulation and to affect its flow. In our example the view has four interaction elements. These are two combo boxes containing the IDs of application endpoints and object keys. Furthermore there are *Publish* and *Lookup* buttons, invoking homonym operations on the selected endpoints (see fig. 1 label (6)).

After setting up all simulation parameters we can assign the number of application endpoints, determine the duration of the simulation and finally start it (see fig. 1 label (5)). Hereby the value 0 means that there is no runtime limitation.

As soon as the simulation has terminated, all measurements are evaluated and the results are written to a file by the recorders. The simulation stops if the specified runtime duration has expired. We can also stop the simulation at any time by clicking the *Stop Simulation* button.

Figure 1 shows the SimCon environment simulating the behavior of 20 application endpoints, based on our Tapestry overlay network implementation. However SimCon is not limited to the simulation of distributed hash table algorithms, other Peer-to-Peer technologies and protocols like e.g. P-Grid [1] can also be simulated. The current visualization was created with an educational focus in order to support the comprehension of the Tapestry algorithm and particularly of the publish/lookup mechanism.

4. DESIGN AND IMPLEMENTATION

This chapter presents the design and the implementation of the SimCon simulation environment. SimCon has been designed with strong focus on extensibility and reusability. Extensibility refers to the fact that the simulation environment can be adapted and extended in many ways. Reusability means that overlay network protocols, implemented to perform simulations and measurements, have only weak dependencies to SimCon. This facilitates the reuse of the code in other applications.

Figure 2 shows an overview of the SimCon architecture. SimCon consists of three main components:

- Simulated application
- Simulation controller
- Simulation evaluation component

The simulation controller is the core of SimCon. The controller connects the simulated application to the evaluation component, which is composed of the view and the recorders. Whereas the view is responsible for displaying the simulation process, the recorders have to record and analyze the simulation. Recorders, views, as well as applications are dynamically interchangeable. The controller is able to load them at runtime from a .NET assembly. This way, the high degree of extensibility is reached. The following sections describe the different components and their interdependence in detail.

SimCon is strongly connected to the TopGen graph generator. In particularly SimCon uses TopGen's `GraphManager` library to manage the Internet graph. For that reason section 4.1 will deliver insight to the TopGen implementation.

4.1 TopGen

In order to obtain realistic simulation results which respect real world router graph connectivity, bandwidth limitations and latency distribution, we have to use a realistic model of the router topology underlying the simulated topology. Since actually measuring this topology is not desirable (and often not even possible), topology generators are traditionally used to create realistic router-level graphs. For our simulations, we have used TopGen, a topology generator that has been created within our group. It has been built with a special focus on creating router-level graphs that are solely based on the fundamental principles of the Internet as well as on technological and economic constraints applying to routers. In contrast the approach of most existing topology generators is to create graphs offering a certain vertex degree distribution similar to that of the actual Internet.

Graphs generated by TopGen not only provide information on router connectivity (like the ones created by most existing topology generators) but also on bandwidth, latency and the type of a router. Evaluations have shown that graphs generated by TopGen are very similar to real world data sets not only in respect to vertex degree distribution but also clustering coefficient and assortativity. These metrics are commonly used to distinguish graphs otherwise offering the same vertex degree distribution. Several metrics as well as a function fitter for examining vertex degree distribution are built right into TopGen in order to simplify graph generation and their evaluation.

TopGen can export generated graphs into a file and comes with a library that can be used to load such graphs into one's own applications and use them for one's own purposes. Since many classes and functions from this library are frequently used in SimCon, we briefly discuss the most important classes in the following.

- `Vertex`: A vertex in a router level graph that maintains a list of neighbors and of adjacent edges.
- `OverlayVertex`: This class is derived from `Vertex` and represents an end host in an overlay graph of endpoints. It has a reference to exactly one router level vertex. This is the router the `OverlayVertex` is connected to.
- `Edge`: A class representing an edge between two vertices.
- `Graph`: A `Graph` represents a router level graph and contains a list of edges and vertices. It offers methods to compute the edge costs between two vertices in the graph.
- `OverlayGraph`: This type is derived from `Graph` and represents an overlay graph of overlay vertices. The `OverlayGraph` has a reference to the underlying router level graph.

4.2 Simulation Controller

The simulation controller is the central unit of the simulation environment. It starts and controls the different instances of the simulated application. Via the user interface of the simulation controller the user is able to load router graphs and applications as well as recorder and view modules. The controller loads the Internet router graph generated by TopGen from a file using the `GraphManager` library of TopGen.

The controller loads application modules at runtime from a .NET assembly. We describe the implementation of application modules in section 4.3.1. Each application can define a set of application parameters. These have to be defined as public properties of a setting class that is derived from `ISettings`. Via the SimCon user interface it is possible to modify the settings before a simulation run. By this means

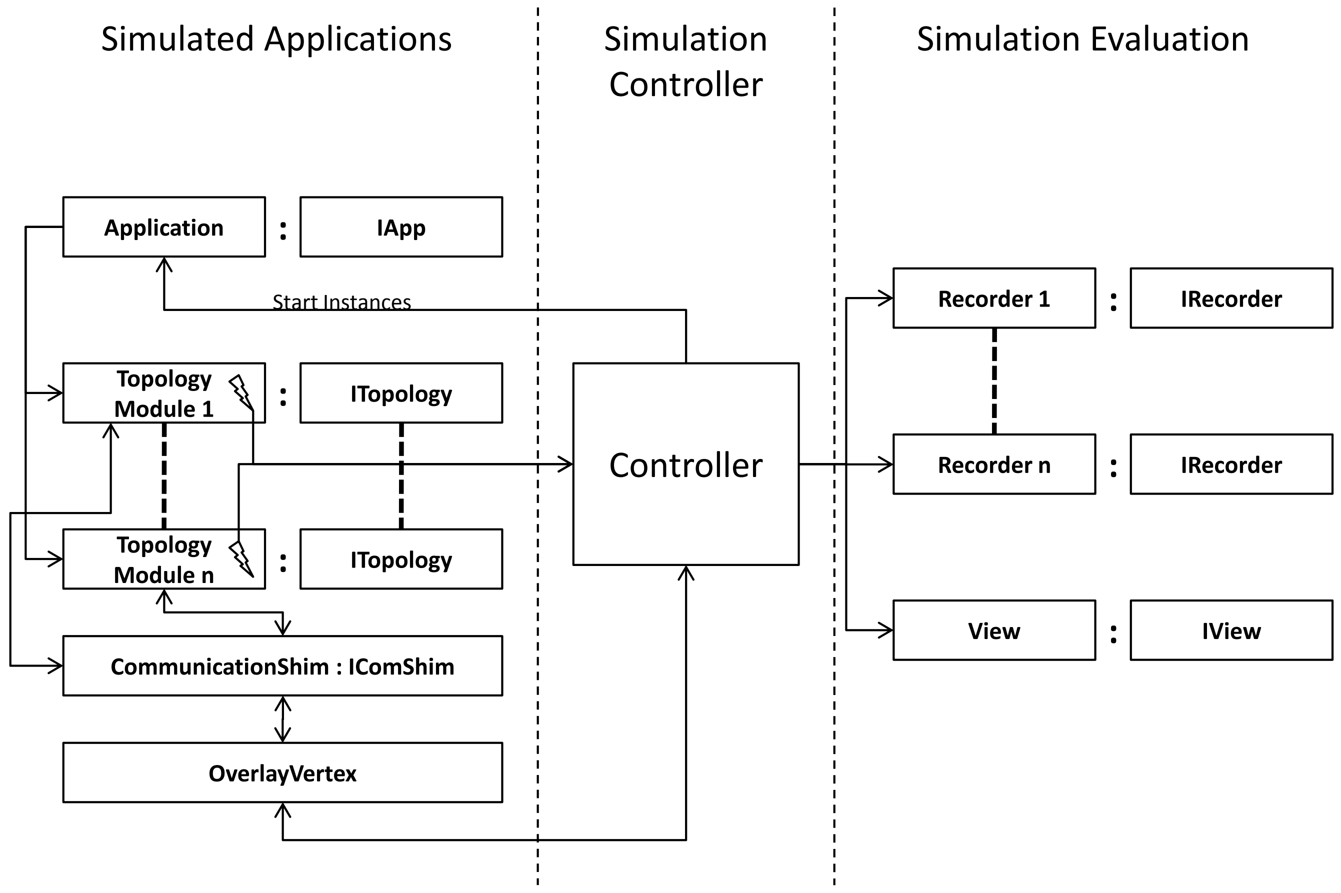


Figure 2: SimCon Architecture Overview

measurement series with different parameters can be performed without changing the application code before each run. Since all applications define their own settings, the application controller has to read the settings from the application assembly by .NET Reflection.

For the purpose of recording the simulation, an arbitrary number of recorders can be registered. In addition to the standard recorder that is an inherent part of SimCon, the user is able to define his own customized recorders. We describe the properties of the standard recorder and the implementation details of customized recorders in section 4.4.1. The second possibility to trace and evaluate the simulation is via the view module that displays the simulation process on the screen. Again, SimCon provides a standard view, but customized views are usable as well. We present details concerning the views in section 4.4.2.

In order to trace the simulation, both recorders and views have to be notified of simulation events. Therefore the simulated applications offer so-called topology events. Before starting the simulation the controller registers view and recorders to the topology events they are interested in.

After view, recorders, router level graph and application have been loaded, the controller is ready to start the simulation. For that purpose the controller creates a user defined number of application instances and registers the topology event handlers for each instance. Each of the application instances has to be connected to one vertex in an Internet graph. The base for this Internet graph is the router level graph generated by TopGen. The vertices in this graph are of type `Vertex` and represent Internet routers. Since the application instances are running on end hosts, an `OverlayVertex` is created for each instance and the instance is connected to this overlay vertex. Therefore a reference to the `OverlayVertex` is given to the application instance. As mentioned above, Internet router graphs generated by TopGen consist of different router types. In reality it is unlikely that end hosts are connected to Internet core routers. Therefore SimCon allows the selection of the router type the end hosts should be connected to. In order to obtain realistic simulation results, end hosts should be connected to the router type that represents edge routers. The `OverlayVertex` provides the methods `SendMessage` and `Receive`, which are used to send and receive messages. The sending of a message is realized by the controller. For that purpose the controller registers an event handler to the `Send` event of each `OverlayVertex`. This event is fired in order to send a message. The event handler reads the ID of the destination of a message from the event arguments and calls the `Receive` method of the destination vertex. We describe how to handle the sending and the receiving of messages inside a simulated application in section 4.3.1.

The control flow of each application instance is started in a single thread by calling the method `Start` of the application. As described in section 4.3.1, every application needs to have such a Start method. Since SimCon uses multiple threads to perform the simulation, it benefits from the current surge of multi-core processor architectures. The simulation ends either after a predefined period of time is elapsed or when the user forces the simulation to stop via the stop button. In each case the controller terminates the simulation by aborting all simulating threads. Afterwards the method FinalizeRecording of all registered recorders is called to notify that the simulation is terminated so that they can write the results of their measurements to a file.

The simulation of random link and node failures is also a task of the controller. For this purpose a user can define a probability for link and node failures. According to this probability the controller shuts down nodes and discards messages randomly.

4.3 Simulated Application

On the left side figure 2 shows the different components of a simulated application and how they work together. As mentioned before, a simulated application consists of two self-contained modules: An application module and a topology module. The application module has to emulate the behavior of a large-scale application, for example a MMOG. The topology modules realize the underlying overlay network topology. In order to achieve exchangeability of topology modules they have to implement the interface ITopology. Thus it is possible to test one application with different topology modules by changing the topology module while keeping the same application module.

4.3.1 IApp - Application Interface

An application module has to be derived from the abstract class IApp. Figure 3 shows an overview of the IApp class. The method SetVertex is used to set the OverlayVertex the application is connected to. Via SetKnownVertices the controller can hand over a set of initially known vertices to the application. The application settings we described above can be set by the method SetSettings. The SetTopologies method is used by the controller to cause the application to instantiate the topology modules. As already mentioned the method Start is used to start the control flow of an application.

As described in section 3 using view's interaction elements we can gear into a running simulation manually. To enable this we use a mechanism to send asynchronous events to an instance of a simulated application. For this purpose IApp defines a method AsyncEvent. This method is called by the controller in order to send asynchronous events to an application. This can be done since the controller has references to all application instances.

4.3.2 ITopology - Topology Interface

The interface ITopology (fig. 4), that has to be implemented by the topology modules, defines a set of methods that have to be provided by an overlay network protocol. These are methods like SendUnicast to send a message to another vertex. Moreover there are methods like Join or Leave to join and leave an overlay network that can be used by application endpoints to go on- or offline. In order to enable the recorders and the view to trace the simulation, the topology modules have to fire events if certain incidents occur, for instance if a vertex joins an overlay network. All these events have to be of type TopologyEvent since

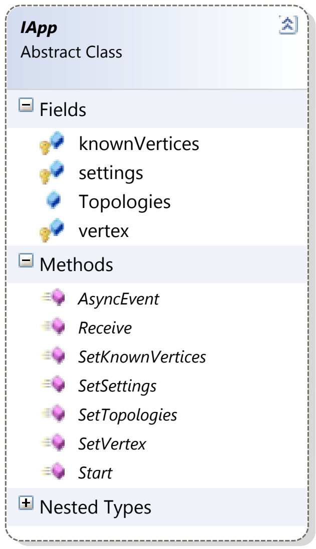


Figure 3: IApp Class Diagram

recorders as well as views expect this type. The arguments of these events have to be derived from TopologyEventArgs.

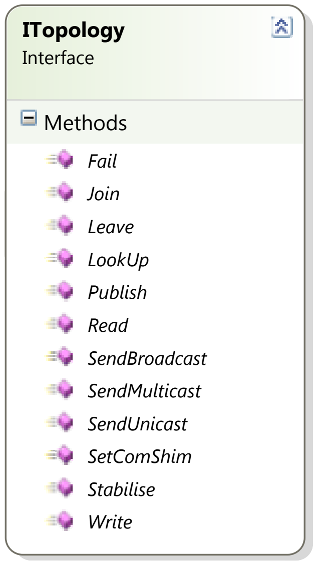


Figure 4: ITopology Class Diagram

As already mentioned, the OverlayVertex class provides methods to send and receive messages in SimCon. But since the topology modules are supposed to be reusable in other applications it is important that they do not have any references to the SimCon specific OverlayVertex. For that reason the communication shim layer is introduced. The communication shim is located between OverlayVertex and topology modules. A communication shim implementation has to be derived from the abstract class IComShim. In a topology module this class is responsible for sending and receiving messages. It defines the abstract method Send and the event Received that is fired if the communication shim receives a message. By this means a topology module is able to send and receive messages via the methods defined by `IComShim` without knowing about the concrete communication shim implementation. By interchanging the communication shim a topology module is usable in different applications. A concrete communication shim implementation is responsible for mapping the sending and the receiving of messages to the underlying communication mechanisms. For example the SimCon communication shim uses the `OverlayVertex` to send and receive messages. Messages sent via a communication shim have to be derived from the class `BaseMessage` that is defined as inner class in `IComShim`. By this means the communication shims allows the above mentioned reusability of the topology module code. Moreover the communication shim mechanism can be used to simulate already existing overlay protocol implementations using SimCon. All we have to do is to implement a corresponding communication shim.

4.4 Simulation Evaluation and Visualization

Recorders and views are responsible for the evaluation and the observation of the simulation process. They can register to arbitrary topology events of the simulated application in order to get information about the course of the simulation process.

4.4.1 IRecorder - Recorder Interface

A customized recorder has to be derived from the abstract class `IRecorder` (fig. 5). This class contains a public dictionary named `Handler` that defines a mapping of topology events to event handler methods. The key of an item of this dictionary is an `EventInfo` object and the value is a delegate to an event handler method. In order to enable a recorder to dynamically define this mapping in dependence of the events provided by the application, the controller calls the method `SetHandler`. An `EventInfo` list of the available events is handed over to the method as an argument.

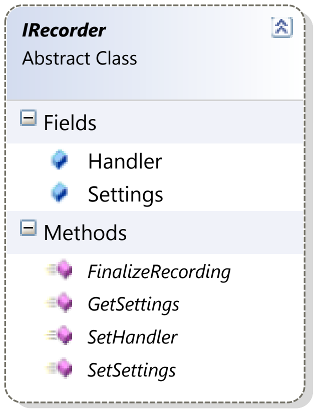


Figure 5: IRecorder Class Diagram

Similar to an application a recorder is able to define a set of properties. These are modifiable before a simulation run is started. The settings have to be defined in a class that is derived from `IRecorderSettings`. To enable the controller to get and set these settings, the methods `SetSettings` and `GetSettings` are available.

The last `IRecorder` method is `FinalizeRecording`. The controller calls this method after the simulation is terminated. Thereupon the recorder is able to finalize the measurement and to write the results to a file.

As described in section 4.2 the controller emulates the sending of a message by calling the `Receive` method of the destination vertex. In order to evaluate the simulation it may be necessary to measure the hop count or the latency of a message. These measurements can be performed by SimCon since the underlying router level graph holds latency values for each link and allows the calculation of the shortest path between two end hosts. By this means hop count and latency can be calculated in a post processing step by using the detailed information about the network infrastructure provided by the TopGen graph. Furthermore, sophisticated evaluations that measure for example the load in certain areas of the router graph can be performed. We believe that this technique of sending end to end messages and evaluating the network traffic from a global point of view is more efficient than simulating additionally the behavior of hundreds of thousands of Internet routers.

The standard recorder is an `IRecorder` implementation that is an inherent part of SimCon. The SimCon controller reads all available topology events from a loaded application. Via the SimCon user interface the user is able to select events he is interested in, as shown in section 3. Before the simulation starts, the SimCon controller registers event handlers of the standard recorder to the selected events. Afterwards the standard recorder is able to count the occurrence of this event during the simulation. For example the number of unicast messages that were sent during a simulation can be counted. Using customized recorders a user is able to perform more sophisticated measurements. As an example we have implemented a customized recorder that realizes the performance versus cost (PVC) metric presented in [10]. This metric uses the average number of bytes sent per node and unit of time as the cost metric. However SimCon is not limited to a set of predefined metrics, since each user has the ability to implement arbitrary metrics within a customized recorder.

4.4.2 IView - View Interface

The design of a view is similar to that of a recorder. Views have to be derived from the abstract class `IView` (see fig 6).

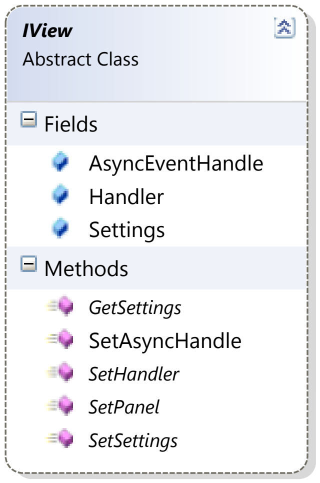


Figure 6: IView Class Diagram

In opposite to the recorders only one view can be attached to a simulation. Similar to `IRecorder`, `IView` defines a dictionary `Handler` and methods `SetHandler`, `SetSettings` and `GetSettings`. In order to display something on the screen the view has to have a reference to a `Panel` object. This object is used by the view to draw to. The `IView` interface has the method `SetPanel` that is used to pass a `Panel` reference to the view.

To enable the user to gear into a running simulation via the view, the view needs a possibility to call the method `AsyncEvent` of an application. Since only the controller has references to the applications, the controller provides a method that can be used to initiate an asynchronous event. The view is able to call this method via a delegate. The controller passes the delegate to the view via the `IView` interface method `SetAsyncHandle`.

5. RELATED WORK

The analysis of overlay network characteristics is the key to the development of distributed large-scale applications based on such networks. In this section we present some environments for overlay network simulation. Most of them come up with one or several implementations of overlay networks (Chord, Tapestry, etc.) and/or provide an extensible API for the development of further network technologies. Some others have been explicitly developed with the objective of presenting the features of a special overlay network.

VisPastry[2] is a Pastry and Scribe [3] visualization tool written in *C#*. It provides a GUI with a map of the world. The nodes are randomly located on the map. Using VisPastry we can get an impression of Pastry's routing mechanism and see how Scribe's multicast tree is built. Since VisPastry is tailored for Pastry and Scribe visualization, it does not support any other overlay network implementations.

Chord also provides a simulation and visualization environment: *Visualizer*[3]. The Visualizer gives the information about some specific characteristics of an assigned node. In this way it can represent the state of the Chord ring and facilitates better understanding of its properties. Similar to VisPastry, which can be used only with Pastry and Scribe, Visualizer is restricted to Chord simulation and visualization.

OverSim [2] is a simulation framework for overlay networks based on *OMNeT++*[22]. The main advantage provided by OverSim in comparison to the above mentioned environments is that it can be used to simulate miscellaneous overlay networks. OverSim supports Chord, Kademlia [14] and Gia [4] implementations which also can be used in real world applications. Moreover, OverSim provides a generic interface to implement additional overlay networks. Similar to SimCon, OverSim may also be used with different exchangeable underlay network models. Overlay networks with up to 100000 nodes can be simulated by OverSim.

Another simulation environment written in *C++* is *P2PSim* [10]. It compares diverse overlay networks such as Chord, Tapestry, Kelips [7], Kademlia and OneHop [6]). P2PSim puts the focus of the simulation on key lookups, crashes and re-joins of overlay nodes. But only a limited set of statistics can be gathered using this environment. P2PSim can be combined with different underlying network models. The simulated network can have up to 3000 nodes.

OverlayWeaver [20] is an overlay construction framework implemented in *Java*. It can be used for development and testing of new and existing overlay networks. OverlayWeaver provides an application user interface which supports some common features like distributed hash table (DHT) or multicast to facilitate the implementation of novel overlay networks. But it also supports multiple existing routing algorithms i.e. Chord, Kademlia, Koorde [9], Pastry and Tapestry. Overlay networks with up to 4000 nodes can be simulated. But neither may statistics be gathered nor is support for modeling underlying networks provided [16].

Narses [12] is a *Java* discrete-event simulator, designed to validate the network scalability with a huge amount of traffic. Narses allows a tradeoff between performance and accuracy and provides a number of underlying network topologies with up to 600 nodes. Using Narses some statistics may also be captured.

PlanetSim [5] is another simulation framework for overlay networks also written in *Java*. It supports Chord and *Symphony* [13] and provides an API for one's own implementations. In the generation stage a zero-sized network has to be created. Then using join, leave and fail operations up to 100000 nodes can be combined to an overlay network. PlanetSim also allows saving of a current network, or loading of a serialized network from disk. By this means a network that was once created can easily be reused. The PlanetSim-API provides broadcast and DHT operations on the application level. PlanetSim offers a very simple underlying network model, but no statistics can be gathered [16].

PeerSim [8] - a *Java* written simulation environment - supports the simulation of networks with up to 1000000 nodes. The simulation is restricted to joining, departing and failing of nodes (no traffic simulation). Moreover no support for the modeling of underlying network topologies is provided. PeerSim offers an API for gathering statistical data.

More detailed studies of overlay network simulators are provided in [16], [15] and [17].

6. CONCLUSION AND FUTURE WORK

Distributed large-scale applications necessitate the use of distributed underlying topologies and make high demands in terms of latency, scalability, reliability and performance. Unfortunately, it is not trivial to verify which overlay network is the most appropriate to fit the requirements of a certain application.

We propose an approach for gathering statistical information about the behavior of overlay networks and evaluate this information on the base of certain metrics derived from the application's requirements. By this means we figure out which network technology meets the given requirements best.

Our simulation environment SimCon facilitates the conjunction of large-scale applications with different underlying network technologies. Its strong focus on the analysis and evaluation of network characteristics makes it interesting for use in research projects and is very helpful for developers of large-scale applications. Moreover, the provided view interface facilitates the use of SimCon in an educational context.

At the moment SimCon can be used with several Internet topologies generated by TopGen, however there is only one (Tapestry) implementation of an overlay network technology. Our short-term objective is to implement some more

[2]`http://research.microsoft.com/~antr/Pastry`

[3]`http://pdos.csail.mit.edu/chord/howto.html`

overlay networks (Chord, P-Grid). This would allow us to compare them in terms of performance and scalability, and to provide first evaluation results. We are also to improve the usability of the SimCon environment, for example by supporting scripts to enable automated simulation configuration.

The current version of the SimCon implementation does not support clustering, constraining the number of simulated application endpoints to ca. 2000[4]. Therefore our middle-term objective is to extend our implementation in terms of clustering. This would enable simulation with a considerable higher number of endpoints and allow us to provide even more significant evaluation results. The multi-threading awareness of SimCon and the usage of communication shims facilitate this aim.

In order to compare our simulation environment with other available tools, we intend to perform measurements and evaluations with respect to performance and scalability. Hereby analyzing whether these simulations can benefit from multi-core processor platforms is an interesting point.

7. REFERENCES

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Sixth International Conference on Cooperative Information Systems (CoopIS 2001), Lecture Notes in Computer Science*, 2172:179–194, 2001.

[2] I. Baumgart, B. Heep, and S. Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, May 2007.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), oct 2002.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, New York, NY, USA, 2003. ACM Press.

[5] P. Garcia, C. Pairot, R. Mondejar, J. Pujol, H. Tejedor, and R. Rallo. Planetsim: A new overlay network simulation framework. *In Journal: Software Engineering and Middleware*, Volume 3437/2005:123–136, 2005.

[6] A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 9–9, Berkeley, CA, USA, 2004. USENIX Association.

[7] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[8] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. Peersim: A peer-to-peer simulator. Available: `http://peersim.sourceforge.net/`.

[9] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.

[10] J. Li, J. Stribling, R. Morris, M. F. Kaashoek, and T. M. Gil. A performance vs. cost framework for evaluating dht design tradeoffs under churn. In *Proceedings of the 24th Infocom*, Miami, FL, March 2005.

[11] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the internet's router-level topology. In *SIGCOMM*, pages 3–14, 2004.

[12] P. Maniatis, D. S. H. Rosenthal, M. Roussopoulos, M. Baker, T. Giuli, and Y. Muliadi. Preserving peer replicas by rate-limited sampled voting. *SIGOPS Oper. Syst. Rev.*, 37(5):44–59, 2003.

[13] G. S. Manku, M. Bawa, and P. Raghavan. Symphony: distributed hashing in a small world. In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 10–10, Berkeley, CA, USA, 2003. USENIX Association.

[14] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proceedings of IPTPS02, Cambridge, USA, March 2002.*, 2002. `http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf`.

[15] S. Naicken, A. Basu, B. Livingston, and S. Rodhetbhai. A Survey of Peer-to-Peer Network Simulators. In *Proceedings of The Seventh Annual Postgraduate Symposium, Liverpool, UK*, 2006.

[16] S. Naicken, A. Basu, B. Livingston, S. Rodhetbhai, and I. Wakeman. Towards yet another peer-to-peer simulator. In *Proceedings of The Fourth International Working Conference on Performance Modelling and Evaluation of Heterogeneous Networks (HET-NETs), Ilkley, UK*, 2006.

[17] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The state of peer-to-peer simulators and simulations. *SIGCOMM Comput. Commun. Rev.*, 37(2):95Ű98, 2007.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[19] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[20] K. Shudo. Overlay weaver home page. Available: `http://overlayweaver.sourceforge.net/`.

[21] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[22] A. Varga. Omnet++ community site. Available: `http://www.omnetpp.org/`.

[23] K.-H. Vik, C. Griwodz, and P. Halvorsen. Applicability of group communication for increased scalability in mmogs. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 2, New York, NY, USA, 2006. ACM Press.

[24] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, Jan. 2004.

[4]Tested on a Core 2 Duo T7400 CPU 2.16 GHz machine with 2 GB RAM.