Network Simulation and its Limitations

Sebastian Rampfl
Betreuer: Florian Wohlfart, Daniel Raumer
Seminar Future Internet SS2013
Lehrstuhl Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: rampfl@in.tum.de

ABSTRACT

Computer networks technology is subject to constant change and innovation. New ideas and concepts regarding the usage of networks and the Internet demand new network protocols and technologies. Due to its complexity and need for backward compatibility many challenges arise from developing, implementing, testing and understanding these technologies. This is where network simulation comes into play. Many problems can be solved by using network simulators such as NS3. It is a powerful tool which enables an in-depth look on many different aspects of a computer networks technology. Nevertheless the extensive functionality of NS3 cannot overcome some limitations of network simulation regarding for example credibility or scalability. This paper is offering a close look on network simulation by describing NS3 and its core functionality. The description forms a basis for the following discussion of the problems which are inherent in network simulation.

Keywords

network simulation, network emulation, NS3, discrete-event simulation, simulation credibility, model validation

1. INTRODUCTION

Over the last decade network simulation has become increasingly important. One reason for that is the rapid growth of the Internet and networks in general. Therefore new potent network simulators are needed to enable the development of advanced network technologies. NS3 [2] is the result of a long evolution of network simulation and a new generation simulator. It offers many features for creating highly adaptable simulations to fulfill the needs of the growing number of network researchers and developers. Even though NS3 is a very advanced network simulator it fails to overcome some limitations all network simulators have in common. These limitations and their consequences will also be focused on here. In the second chapter network simulation and especially NS3 is described in detail. The design and structure of NS3 is explained first and in the following subsection the NS3 workflow and alternatives to NS3 is addressed. The third chapter focuses on limitations of network simulation and in particular of NS3.

2. NETWORK SIMULATION

A network simulation is the implementation of a simulation that attempts to imitate the real world behaviour of a computer network or certain aspects of a computer network to analyse the captured information and transmitted data. This information can then be used to draw conclusion or for example "investigate characteristics of a new routing protocol" [18] and how a protocol reacts to certain changes. According to the official NS3 tutorial [4] a simulation consists of a description of a network and how the components interact, basic control functions for managing the simulation and some sort of logging functionality to capture data. The main motivation behind network simulation is to accomplish more reliability and less maintenance costs regarding the development of a new technology.

2.1 Concepts and operational Scenarios

There are many diverse applications for network simulation. They are based on either one of the following two concepts. The first one is the pure simulation. This means that every component and every aspect of the network is simulated and the packets or messages, that are created within the simulation, are neither transferred to a real network, nor processed as real network traffic outside of the simulation. One example for such a simulation is the implementation and integration of an experimental network protocol in a network simulation. Hereby different aspects of such a protocol can

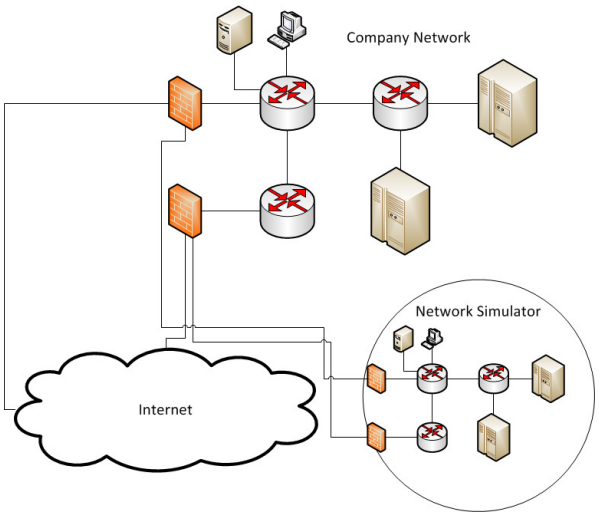

Figure 1: An example for network emulation

be investigated. One of these aspects can be the generation of anomalies regarding the behaviour of the protocol in order to find the cause of the anomaly. The simulation designer can also create circumstances for its technology that might

doi: 10.2313/NET-2013-08-1_08

not be possible on a real world testbeds. One application of this may be the creation of a simulation with connections of higher bandwidths than today's networks are capable of. This makes it possible to investigate the behaviour of the protocol under circumstances which might occur in the future. Another application for network simulation is the new research field of car-to-car networks, which is very costly and even risky when implementing prototypes and testing them. The second concept is network emulation. Simulators can be combined with and attached to real networks to send and receive traffic of these networks. The Figure 1 visualizes an idea on how to use network emulation to help defend against a Distributed-Denial-of-Service attack. A company network is connected to the Internet and guarded by firewalls. These firewalls are also connected to a network simulation and are able to mirror every traffic that passes through them. This simulation tries to copy the topology, attributes and behaviour of the company network. In case of a DDoS attack the firewalls could forward the traffic to the simulation and a load balancing software could apply different load-balancing strategies by reconfiguring the emulated network. Furthermore it would use the results of its tests to make a decision on what load-balancing strategy may be the most effective against the ongoing attack. This strategy can then be applied to the real company network in order to repel the attack. This is obviously are highly complex and elaborate way of using network emulation, but it demonstrates, that there are more fields of application for network simulators than research and development.

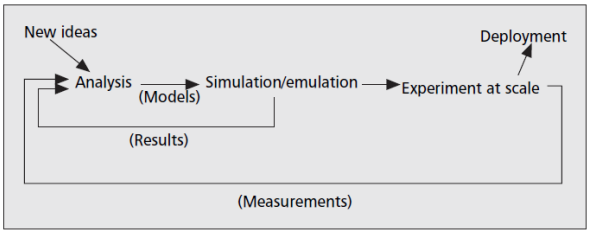

Figure 2: The cycle of network simulation [16]

The development of a technology or a product with the help of network simulation is an iterative process. As drafted in figure 2 a simulation is modelled on the ideas and concepts of the simulation designer. The results and measurements of the simulation are used to alter it, in order to create different simulation behaviour and results, until the desired outcome can be accomplished. In comparison to implementing a protocol on a real testbed simulation is in most cases more flexible and less cost-intensive. Even though creating a simulation may also become elaborate and costly.

2.2 NS3

NS3 is a discrete-event network simulator. It is open source and licensed under the GNU GPLv2 license. Since its release in 2008 it is one of the most important and widely used network simulation tools. Even though it does not offer any graphical user interface it has proven to be comprehensible and easy to handle. NS3 is intended to be used with Linux, although it is possible to run it on Windows by using cygwin or MiniGW. It was developed to replace its predecessor NS2, which was released in the mid-90s. The reason for redesigning the simulator was the "limited scalability regarding memory usage and runtime" as described in [18] . NS2 was designed to reduce compilation time by using C++ in combination with the scripting language oTcl. While the simulation components, their behaviour and the topologies are described by C++ code, oTcl scripts model the overall simulation behaviour and are used for binding. Today's simulation designers are not focused on compilation time, but on scalability and performance. This led to the development of NS3. The following subsections describes the functionality NS3 offers and explains how to work with it.

2.2.1 Design and Structure

As stated before NS3 is based on the concept of discrete simulation. This means that a point in simulation time is assigned to every event, events are initiated and triggered consecutively and "simulation time moves in discrete jumps from event to event"[8]. Computing these events in real-time is an option as long as the system running NS3 offers enough computing power and the attributes set for the simulated network do not exceed the laws of physics. Especially in network emulation this may become necessary when sending or receiving packets from real world hosts. In most cases however realtime is not an issue and the focus is more on the order of events and their consequences regarding captured data.
The NS3 project uses Mercurial [10] for source code management. Documentation of this code can be accessed via Doxygen [17], a tool for creating documentation. The Python-based build system waf [7] is recommended by the NS3 development team. Their NS3-tutorial [4] gives all necessary information for getting started with NS3. This document is also a step-by-step documentation of a couple of useful implementation examples for some network simulations.
Creating a NS3 simulation consists of four basic steps. Before describing them in detail the key abstractions of a NS3 simulation have to be explained. These basic component types of a network are nodes, applications, net devices, channels and topology helpers. They are all represented by sets of C++ classes and their functions and properties, including examples, are explained in the ns3-tutorial [4]. A node in a NS3 simulation stands for a communication point, such as an end system or a router. It is the base for any events and interaction. Functionality and properties are added to these nodes. The nodes are interconnected by channels, which represent the different forms and media of data transmission. Two of the C++ classes in NS3 that describe channels are the PointToPointChannel and the WifiChannel. As the name indicates the PointToPointChannel implements a simple wired connection from one endpoint to another endpoint. The WifiChannel represents a Wifi connection and is designed to behave like such a connection. The third key abstraction are net devices. They are attached to nodes and channels and form what in real world would be considered network interfaces. There are different types of net devices due to the diversity of channels. As in a real network a node can be attached to multiple net devices. The application is another key abstraction of every NS3 simulation. It forms the actual functionality of the nodes and is supposed to be implemented by the simulation designer. NS3 offers many different applications for all kinds of network functionality. Configuration and adaptation of these applications is the key

to creating the intended network behaviour. The main functionality of the applications is the creation, processing and transmission of data. The simulation designer can create and configure nodes, channels, net devices and applications separately or this can be done by using the extensive and powerful Helper-API of NS3. These helper classes make it possible to configure a network simulation with relatively low effort. Adding a protocol stack and addresses to a set of nodes are one of the many options the helper classes offer. They also make it much easier to read and understand the code of an NS3 simulation. As mentioned before all code is documented and can be accessed via Doxygen. Having a closer look on what the NS3 helper have to offer is recommend before starting to program a simulation.

2.2.2 *NS3 Workflow*

The helper classes are a main aspect of the NS3 workflow. As mentioned in chapter 2.2.1 there are four main steps when working with NS3. The first one is the programming of the actual network with its topology, the used protocols and its applications. This is done by the abovementioned helpers and can be illustrated with the following example. In order to create a group of hosts a NodeContainer class is initiated with the corresponding attributes such as the number of hosts. Furthermore the InternetStackHelper class is used to add the network protocol stack to the nodes in order to enable Internet communication. The first step also includes setting up all addresses, such as MAC and IP addresses, and adding and writing the application classes. This is where most of the logic is implemented. Another important part is the configuration of the class attributes. NS3 offers an attribute system, which enables a more convenient way of dealing with properties, variable values and other simulation related information. As described in [8] namespaces

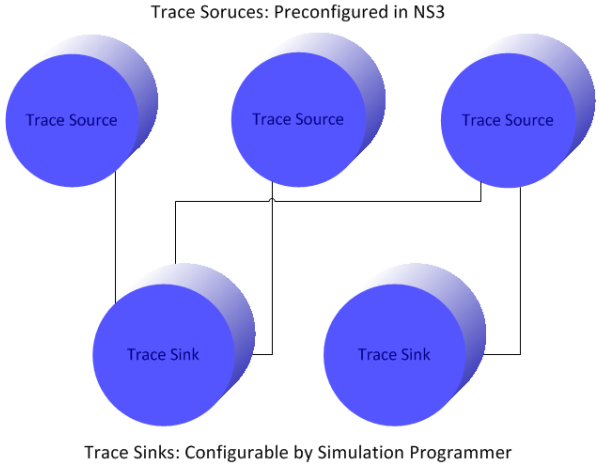

Figure 3: Tracing Sources and Sinks

and paths can be used to access and edit the values and information the attributes offer. A list and description of all attributes can be found in the official attributes manual [3]. The second step is the description of the simulation behaviour. A set of methods for initiating and stopping the simulation, as well as other control methods such as debug logging needs to be included in the implementation of the simulation. Tracing has be configured too. As depicted in Figure 3 the simulator offers trace sources and the simulation designer implements the trace sinks. These trace sinks specify what information to capture an what to ignore. Tracing can be used on different levels of abstraction. The designer can either use preconfigured sets of trace sinks with less adjustment capabilities or specify trace sinks in detail. Too little or too much information can destroy the benefit of implementing a simulation and therefore much consideration should be given to tracing in NS3. Filtering information is one key for success, because insufficient information leads to wrong conclusions regarding the simulated network.

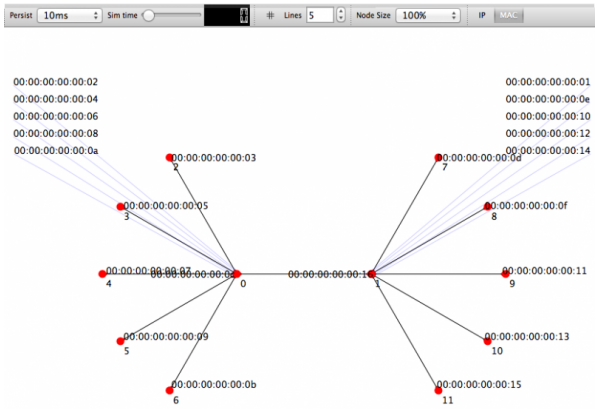

Figure 4: GUI of NetAnim visualization tool [14]

After programming the simulation in step one and two the next stage is the execution of the simulation.
The fourth step is the output analysis. This not a part of NS3, but an important aspect of network simulation. NS3 offers no tools for analyzing, visualizing or processing the data gained through network simulation. Nevertheless there are many free tools offering a broad range of features for this purpose. One of them is NetAnim [14]. It is a tool to animate data gained from tracing. Figure 4 is a screenshot of the GUI of NetAnim. Another tool for visualization is

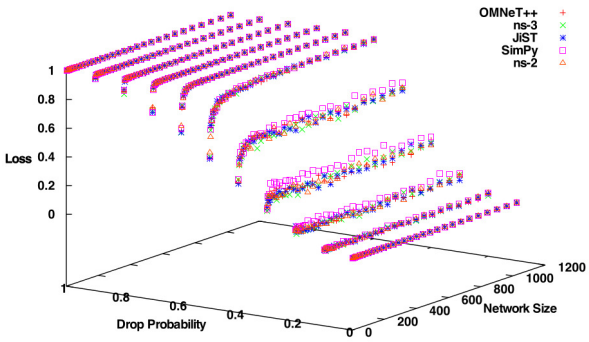

Figure 5: Gnuplot graph visualizing simulator output [18]

Gnuplot [19]. With Gnuplot static, highly customizable 2D and 3D graphs can be created to visualize large amounts of data. Figure 5 displays a Gnuplot graph. This 3D graph

doi: 10.2313/NET-2013-08-1_08

visualizes the amount of packet-loss observed in a network simulator performance comparison carried out in the reference [18]. Wireshark is also a useful tool in this context. Its purpose is filtering and displaying transmitted information by listing packets and its content. All these tools for output analysis offer a broad range of options and customizability and this is a necessity in the field of network simulation. where every output analysis needs to focus on very specific information and goals.

2.2.3 *Models*

Models are a key element of NS3. In the NS3 model library [5] models are defined as "abstract representations of real world objects, protocols, devices, etc." [5] or as stated in [4] "an abstraction of reality" . Models are written in C++ and are aggregated into modules, which are all individual software libraries. NS3 relies on an active community. It designs, implements, documents, tests and validates these models. The models are meant to form the base for simulation designers to create the functionality they have in mind. The behaviour of these models can be modified by changing the code, adding new attributes or just reconfiguring the attributes. The abovementioned helper classes provide easy access to the models. The NS3 model library [5] lists a large variety of models. Some of them implement relatively simple functionality such as the Point-to-Point model and others offer more complex functionality such as local or global routing protocols. A model needs to be validated in order to know whether and to what extent the model differs from the behaviour of real world object it is trying to simulate. This topic will be discussed in a chapter 3.1.
NS3 frameworks are groups of models, which focus on modeling coherent functionality. A framework is intended to offer a simulation of an environment, which simulation designers can build upon. An example for this is the UAN Framework[5]. It offers a variety of underwater network scenarios. The NS3 predecessor NS2 also relies on the concept of community-based model development, but NS2 models are not compatible to NS3. This means that, even though NS2 models also written in C++, integrating them into a NS3 simulation is not possible without extensive adjustments. With the amount of available models constantly growing this concept has proven to be quite powerful.

2.3 Alternatives to NS3

There are several network simulators with slightly different approaches regarding modeling of simulations, revenue model and support. One of them is OPNet. In contrast to NS3 OPNet [12] is a commercial network simulator. It offers a graphical GUI for the design of the simulation and the visualization of captured data for output analysis. Like NS3 OPNet simulations are based on discrete events. One disadvantage of OPNet is the fact that it is proprietary software and therefore limited in terms of customizability. Like NS3 an open source system can be modified in every aspect and the OPNet simulator lacks that feature. Another disadvantage is the lack of support and collaboration an active community of an open source project provides. Nevertheless OPNet has gained a big market share. Two reason for this are the fact that it has the resources to offer customer support and validation of their models. Another aspect is the graphical user interface. Figure 6 displays different levels of the OPNet GUI. Such a bundle of features increases the understandability regarding the complexity of a network simulation. These two aspects are very important for companies when choosing a network simulator for the development of their network technologies.

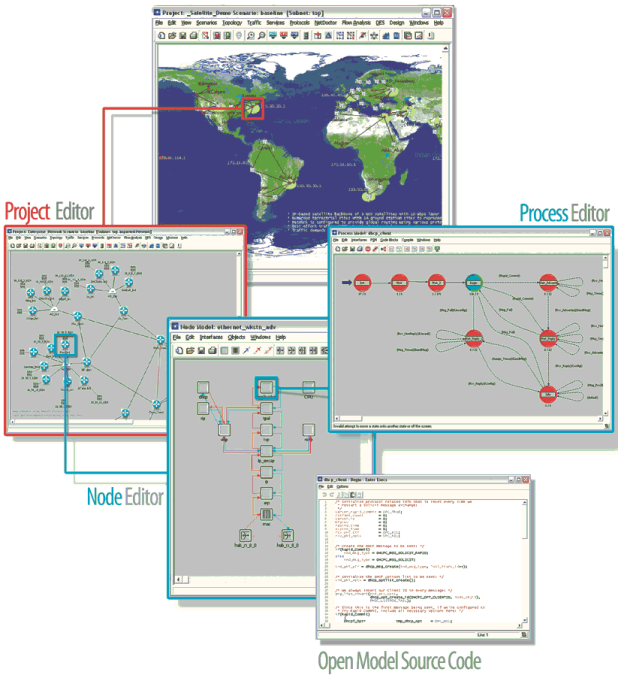

Figure 6: Different levels of the OPNET graphical user interface [12]

A second alternative is SimPy [11], a Python based open source network simulator. It is licensed under the GNU Lesser GPL, which means it is free for non-commercial use. SimPy is a "object-oriented, process-based discrete-event" simulator [11] and written in Python. It offers a GUI and even plotting for output data. The developers claim SimPy to be and very easy to use network simulator. SimPy seems to be a good alternative for those who want to work with a GUI when designing a simulation and are not willing to invest in a proprietary software system.

3. LIMITATIONS OF NS3

As stated and described in chapter two network simulation is a powerful tool for a range of possible applications. Nevertheless there are limitations to it. The history of network simulation is long and there has been many obstacles in its evolution. Although constantly increasing computing power and more powerful programming languages managed to overcome many obstacles, some of them still remain. The different network simulators use different approaches on dealing with these limitations and therefore their performance and reliability vary. This chapter focuses on the limitations of NS3, as well as its consequences.

3.1 Credibility and Validation

Simulation credibility is a challenge not only in the field of network simulation. All aspects of reality can never be implemented in a simulation and therefore compromises have

to be made. Such compromise can be a lower level of simulation detail or the absence of certain aspects of the network. A network simulation can only be useful if the behaviour it shows and the results it delivers are comparable to a real network. The simulation designer can never be one-hundred percent sure if this is the case. Hence a certain level of trust is need when working with simulation.
The authors of reference [8] name three strategies to increase trust in a simulation. The first two are the rather simple concepts of regression tests and reuse of code. These methods are useful in case of fault prevention and quality of code and therefore have an indirect impact on the increase in credibility. The third strategy is the validation of models, in this case the NS3 models described in the last main chapter, by using testbeds. The main purpose of a validation is to proof, that the behaviour of a network simulation model is comparable to the behaviour of a testbed with the same configuration as the simulated network. This is needed by simulation designers who want to use these models to create new simulations, because they are relying on the correctness of these models. The Reference [1] is such a validation. The authors of [1] are comparing the performance and behaviour of the IEEE 802.11 MAC model of NS3 to a testbed providing the same functionality. Furthermore they draw conclusion from their results. Therefore they use different scenarios with different focuses and objectives and then compare their measurements. They come to the conclusion that the IEEE 802.11 MAC model is a rather good representation of reality, although the authors observed some "noticeable quantitative differences" [1] regarding the measurements. In their opinion this is not necessarily the fault of the model but may also be a flaw in the implementation of the testbed. The conclusion of the authors demonstrates another problem regarding the credibility of network simulation models. Even if the model someone is trying to validate is flawless in terms of behaviour and attributes, the testbed with its real world hardware and implementations is probably not. Thereby the results of measurements may differ significantly. Simulation designer always have to consider that they are creating a simulation to estimate the real world behaviour of the network technology and do not want to create a perfect simulation implementation. In summary it can be said, therefore, that validation is a potent method to build up credibility, if the fact, that real world devices not always behave like they should be, is considered.

3.2 Simulation of upper Layer Functionality

As described in chapter 2.2.3 simulation designers do not build up all functionality from scratch. They rely heavily on the usage of models. Every model, even if validated, are a possible source of wrong behaviour and failure. Validation is an improvement, but it does not guarantees the absence of malfunction or the presence of all expected features. The designer always has to consider what a model does not implement. It may be written for one context in which it functions flawlessly, but, if used in a different context, malfunctions completely. The problem of overrating the capabilities of models is described in [4]. Some models implement every component of a network protocol, but this does not mean that every aspect of its behaviour does not differ from how the protocol would behave in the real world given the same preconditions and circumstances. The higher in the protocol stack the designer operates the more likely the occurrence of these problems becomes. None of the sources describes any solution to this problem.

3.3 Scalability Limits

One of the benefits of simulating a network is the fact that adding or removing components, devices and channels is easy and fast in comparison to a testbed where devices have to be installed and connected. A simulation is in theory not limited by the amount of devices or transmitted data. In reference [6] the topic of network simulation scalability is covered by "quantitatively [characterizing] the capability of parallel simulation tools to simulate large-scale networks". The authors identify two main limiting factors for scalability of network simulators: the memory usage and the required computation time. Every node, channel and the other components require memory space and therefore their number is limited by the available memory. As in a discrete event simulator, such as NS3, events are processed in a certain time and the amount of events that can be processed is limited assuming that the simulation designer defines a maximum time for the computation of the simulation.
The authors of [6] explain how to improve simulation scalability. Their approach is the usage of parallel computing for processing events. Although this paper is ten years old and hence references to NS2, the main concept of parallel computation of network simulations is applicable to NS3. The NS3 model library [5] describes how to integrate the Message Passing Interface [9] in a NS3 simulation. MPI is a standard which formulates rules for the parallelization of programs regarding the exchange of information. Its purpose is to enable high performance computing on large clusters of processors. In addition to parallelization the use of distributed network simulation can improve simulation performance and therefore alleviate scalability problems. This concept describes the usage of multiple network simulators deployed on geographically distributed machines. In contrast to the parallel execution of one network simulation on a high performance computation center a distributed simulation needs to focus more on reducing the amount of information sent between the distributed network simulators in order to increase simulation performance. The authors of [15] describe two different methods to split up the functionality of simulated devices.

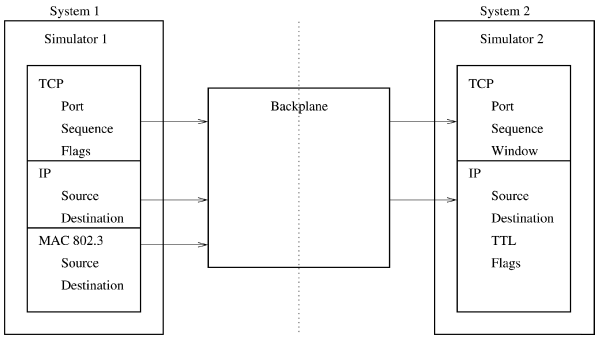

Figure 7: Cross-protocol stack method [15]

The first one is the cross-protocol stack method. The main concept behind it is the computation of complete devices or groups of devices of the simulated network on only one network simulator. This means that the network stack of

doi: 10.2313/NET-2013-08-1_08

one simulated device is located at only one network simulator. Therefore only messages sent from devices within the simulation are exchanged between the distributed network simulators. Figure 7 visualizes this concept. Contrary to the cross-protocol stack method the split-protocol stack method distributes the different layers of the protocol stack to different network simulators. In this case the information exchanged between the network simulators are not the messages sent between the simulated devices but the information passed through the different layers of the protocol stack. One simple example for this is shown in figure 8. The effect of both methods on the amount of message sent between the distributed network simulators highly depends on the topology and the functionality implemented in the simulation [15].

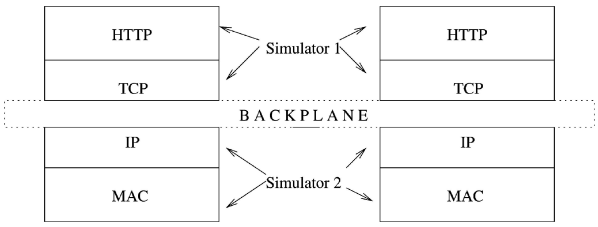

Figure 8: Split-protocol stack method [15]

Reference [13] is a performance analysis of a NS3 network simulation using the abovementioned MPI model to establish distributed and parallel computation. Their results lead the authors to the conclusion that "nearly optimal linear speedup is achievable"[13]. Parallelization and the distribution of network simulation are effective approaches for increasing scalability, but fail to overcome the following aspect of scalability. Often simulation designers want to test their network technology in an Internet-like environment. This is barely possible due to its size and complexity. As claimed in [6] the reason for this is the fact that the Internet is not understood very well and therefore hard to simulate. This is not exclusively a scalability problem but it is a limitation to network simulation.

4. CONCLUSION

With being easily accessible and at the same time highly customizable NS3 has proven to be a potent network simulator. The combination of helper classes and the option to configure the classes individually turned out to be a very useful concept. The community-based model concept of NS3 is a strong point and a disadvantage at the same time. The fact that NS3 is a non-commercial software results in a larger and therefore more active community, which constantly helps to improve, extend and upgrade NS3. On the other hand this makes it impossible to guarantee reliable customer support and bug fixes. This gap is successfully filled by competitors such as OPNet.

There are of course some limitations to network simulation that even NS3 cannot overcome. One of them is credibility. This will always be an issue, because it is clearly impossible to guarantee flawless real world behaviour of a simulation. One approach to partially solve this problem could be a far more detailed formalization of the validation process.

The limitation regarding upper layer functionality is in comparison to validation rather simple. It is more of a limitation of the human mind in terms of the ability to deal with high complexity results than a limitation for network simulation. Some measures regarding structured planning of simulations and a more formalized documentation of models may help reduce this problem

Scalability also remains to be an obstacle. However in contrast to credibility this problem is alleviated to some degree. Real world networks do not grow as fast as the networks that can be simulated. The Internet may be highly complex and not well understood, but progress is being made and therefore scalability may become less of a limitation than it used to be.

5. REFERENCES

[1] Nicola Baldo, Manuel Requena, Jose Nunez, Marc Portoles, Jaume Nin, Paolo Dini, and Josep Mangues. Validation of the ns-3 ieee 802.11 model using the extreme testbed. In *Proceedings of SIMUTools Conference, 2010*, March 2010.

[2] NS-3 development team. Ns-3 network simulator. `http://www.nsnam.org/`.

[3] NS-3 development team. Ns-3 network simulator, ns-3 attributes. `http://www.nsnam.org/docs/release/3.10/manual/html/attributes.html`.

[4] NS-3 development team. Ns-3 network simulator, ns-3 tutorial, Decenmber 2012. `http://www.nsnam.org/docs/release/3.16/tutorial/ns-3-tutorial.pdf`.

[5] NS-3 development team. Ns-3 network simulator, ns-3 model library, March 2013. `http://www.nsnam.org/docs/models/ns-3-model-library.pdf`.

[6] Richard M. Fujimoto, Kalyan Perumalla, Alfred Park, Hao Wu, Mostafa H. Ammar, and George F. Riley. Large-scale network simulation: how big? how fast. In *In Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS)*, 2003.

[7] Google. Waf build tool. `https://code.google.com/p/waf/`.

[8] Tom Henderson, George Riley, Felipe Perrone, and Mathieu Lacage. ns-3 tutorial. `http://www.nsnam.org/tutorials/geni-tutorial-part1.pdf`.

[9] Argonne National Laboratory. Mpi message passing interface. `http://www.mcs.anl.gov/research/projects/mpi/`.

[10] Matt Mackall. Mercurial source code management tool. `http://mercurial.selenic.com/`.

[11] Klaus Mueller and Tony Vignaux. Simpy network simulator. `http://simpy.sourceforge.net/`.

[12] Inc. OPNET Technologies. Opnet network simulator. `http://www.opnet.com/solutions/network_rd/modeler.html`.

[13] Joshua Pelkey and George Riley. Distributed simulation with mpi in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 410–414, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[14] George F Riley and John Abraham. Netanim. `http://www.nsnam.org/wiki/index.php/NetAnim`.

[15] George F. Riley, Mostafa H. Ammar, Richard M. Fujimoto, Alfred Park, Kalyan Perumalla, and

Donghua Xu. A federated approach to distributed network simulation. *ACM Trans. Model. Comput. Simul.*, 14(2):116–148, April 2004.

[16] I. Stojmenovic. Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions. *Communications Magazine, IEEE*, 46(12):102–107, December.

[17] Dimitri van Heesch. Doxygen tool for creation of documentation. `http://www.stack.nl/~dimitri/doxygen/`.

[18] Elias Weingärtner, Hendrik vom Lehn, and Klaus Wehrle. A performance comparison of recent network simulators. In *Proceedings of the IEEE International Conference on Communications 2009 (ICC 2009)*, Dresden, Germany, 2009. IEEE.

[19] Thomas Williams and Colin Kelley. Gnuplot. `http://www.gnuplot.info/`.

doi: 10.2313/NET-2013-08-1_08