VAT4Net - a Visualization and Animation Tool for Network Simulations

Torsten Braun, Jana Krähenbühl, and Thomas Staub
Institute of Computer Science and Applied Mathematics, University of Bern, Neubrückstrasse 10, CH-3012 Bern
{braun, kraehen, staub}@iam.unibe.ch

Keywords: Discrete event simulation, simulation software, computer networks, communication systems, distance learning

Abstract

The paper describes the design, implementation, and evaluation of VAT4NET, a visualization and animation tool for network simulations. It has primarily been designed for simulation experiments performed in a distance learning environment. It supports visualization and animation of trace data obtained from network simulators, e.g., ns-2. Users can view via web browsers results from simulations that have been performed and pre-processed on remote systems. The architecture allows user-specific pre-processing of simulation data.

1. INTRODUCTION

Simulation of networks and protocols is an important subject of university teaching on computer networks. Students need to learn how protocols can be tested and evaluated using simulation techniques. As more and more distance learning systems include remote laboratories, simulation and visualization tools should be able to be integrated into such learning environments. Unfortunately, many simulation and visualization tools such as ns-2 [14] [16] [17] and the Network Animator (NAM) [5] [15] are very platform dependent. Therefore, those tools are rather difficult to use in a web-based e-learning environment, where students should access remote experimental facilities with web browsers only. Installation of specific software on a student's computer must be avoided. Our main goal was to use a visualization tool for an e-learning environment, where students could access remotely to a course infrastructure.

We have designed and implemented the Visualization and Animation Tool for Network Simulations (VAT4Net) to support the analysis step in a network simulation process. VAT4NET enables users to visualize trace files from network simulation experiments. While animating a simulation experiment, special events and features can be visualized. For example, users can detect bottlenecks, packet drops, congestion, network structure, movement behavior of wireless nodes, etc.. Another focus of VAT4Net is processing of statistical data and its graphical representation, e.g., end-to-end delay, packet loss rate and other Quality of Service characteristics to support quantitative experiments for learning and research purposes.

2. ARCHITECTURE

2.1. Design and Implementation Choices

The basic data for all visualization and animation processes is provided by trace files generated from a simulation run in ns-2. The current version of VAT4Net focuses on trace files in NAM trace format, but can be extended easily for other simulation trace file formats as produced by other simulation tools such as Omnet++ [13].

Certain processes are necessary in VAT4Net to animate and visualize the simulated scenario. When analyzing a trace file for the first time the preprocessor handles the file in an initialization process. As soon as the preprocessed trace file is available, the animation, analysis, and plug-in control engine is executed.

VAT4Net combines two different architectures - the client-server and the stand-alone design (Figure 1). Trace files can be stored locally or remote. If the trace file exists on the local computer, VAT4Net is running as stand-alone software. All processes, threads and operations are performed on the local computer system. On the other hand, when selecting a trace file from a remote server, the VAT4Net client first tries to connect to the VAT4Net server on the remote computer. Then, the preprocessing and statistics calculation tasks are performed on the remote computer and only the results of the remote operations are submitted to the client over the network.

VAT4Net has been implemented in Java. It is available as a Java Applet for the web-based infrastructure and as stand-alone software. When viewing a web page containing a VAT4NET Java applet, the Java-enabled browser downloads the applet's byte-code to the client system and executes it by the Java Virtual Machine (JVM) [6]. In general, some functions are restricted while the JVM is running the applet, e.g., file access, network connection establishment, reading system properties. However, it is possible to sign the applet with a digital security certificate and to enable the required functions. In stand-alone mode VAT4Net is started from a Java Archive (JAR) file [6] and interprets the compiled source code in the JVM. In this case there are fewer restrictions as with applets.

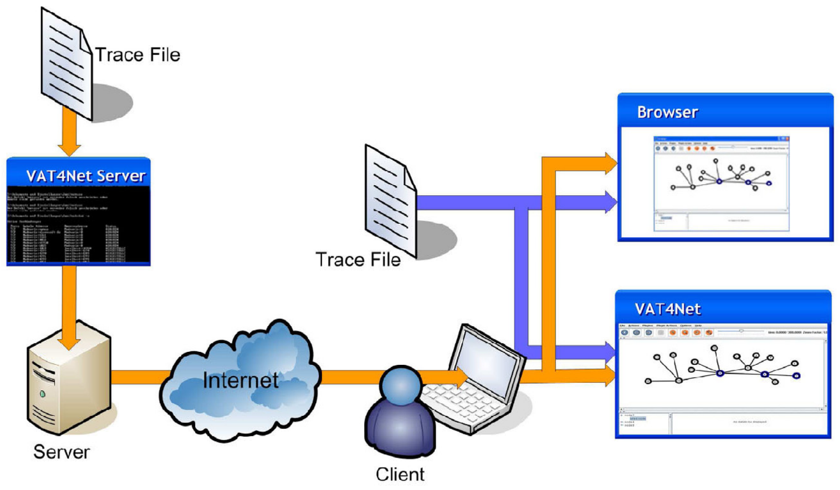


Figure 1. VAT4Net System Architecture.

2.2. Preprocessing Trace Files

Event-based network simulations such as ns-2 generate trace files containing events in a certain format, e.g. NAM format. Events of a single packet are not grouped together, but are distributed over the trace file together with events of other packets.

The upper part of Figure 2 shows a small part of a trace file as a result of an ns-2 simulation. Mixing the various events makes the animation of packets rather difficult. For the animation of a packet on a link the hop (line 587) and receive event (line 618) or a drop event should be known. Therefore, it is important to be aware of line 618 when starting to animate the event in line 587. This requires pre-parsing the file. In certain cases, the complete trace file must be considered for animating a packet.

A look-ahead mechanism within the trace file instead of pre-processing has been proposed by [7]. A look-ahead implementation was not applicable to VAT4Net because of excessive memory consumption of the Java application when holding large parts of the trace file in the heap.

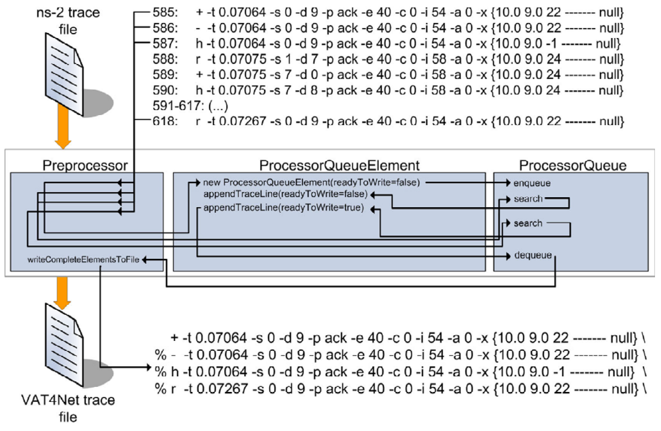
The main idea of our approach of pre-processing trace files is to sort events matching to a packet. In case of the example, matching events are at positions 585-587, and 618.

A new trace file format has been adapted from the NAM trace file format. An example of the pre-processing output format is given in the lower part of Figure 2, which represents a single line of the output file. This line provides all information required for animating a packet. No further look-ahead operations are required.

2.3. Loading Data

When VAT4Net runs as a stand-alone system on a local computer, the data required for animation and visualization operations is loaded by a separate thread from the newly generated or already existing VAT4Net trace file by accessing the local file system. The fact that each line marks one animation event allows performing read operations line by line. The read operation is used for handling either animation or statistics data.

For the distributed version with a remote data source, VAT4NET has to be divided into a client and a server part. The VAT4Net server delivers parsed trace files to the client to animate and visualize them. If necessary the trace file will be pre-processed (by plug-ins, see Section 3.2) first on the server and transported to the client afterwards. Plug-ins are located at the server, because they typically process trace files. The client part is responsible for receiving data from the server and its delivery to the VAT4Net client application, which shows the resulting data as animation, visualization and plug-in-defined view in the Graphical User Interface (GUI, see Section 3.1). The connection between the client and the server is achieved by a TCP connection and SSH (secure shell) port forwarding.


ns-2 trace file
Preprocessor
ProcessorQueueElement
ProcessorQueue
new ProcessorQueueElement(readyToWrite=false)
appendTraceLine(readyToWrite=false)
appendTraceLine(readyToWrite=true)
enqueue
search
search
dequeue
writeCompleteElementsToFile
VAT4Net trace file

```
585:    + -t 0.07064 -s 0 -d 9 -p ack -e 40 -c 0 -i 54 -a 0 -x {10.0 9.0 22 ------- null}
586:    -  -t 0.07064 -s 0 -d 9 -p ack -e 40 -c 0 -i 54 -a 0 -x {10.0 9.0 22 ------- null}
587:    h -t 0.07064 -s 0 -d 9 -p ack -e 40 -c 0 -i 54 -a 0 -x {10.0 9.0 -1 ------- null}
588:    r  -t 0.07075 -s 1 -d 7 -p ack -e 40 -c 0 -i 58 -a 0 -x {10.0 9.0 24 ------- null}
589:    + -t 0.07075 -s 7 -d 0 -p ack -e 40 -c 0 -i 58 -a 0 -x {10.0 9.0 24 ------- null}
590:    h -t 0.07075 -s 7 -d 8 -p ack -e 40 -c 0 -i 58 -a 0 -x {10.0 9.0 24 ------- null}
591-617: (...)
618:    r  -t 0.07267 -s 0 -d 9 -p ack -e 40 -c 0 -i 54 -a 0 -x {10.0 9.0 22 ------- null}
```

```
   + -t 0.07064 -s 0 -d 9 -p ack -e 40 -c 0 -i 54 -a 0 -x {10.0 9.0 22 ------- null} \
% -  -t 0.07064 -s 0 -d 9 -p ack -e 40 -c 0 -i 54 -a 0 -x {10.0 9.0 22 ------- null} \
% h -t 0.07064 -s 0 -d 9 -p ack -e 40 -c 0 -i 54 -a 0 -x {10.0 9.0 -1 ------- null}  \
% r  -t 0.07267 -s 0 -d 9 -p ack -e 40 -c 0 -i 54 -a 0 -x {10.0 9.0 22 ------- null} \
```

Figure 2. VAT4Net Preprocessor

SSH enables to connect to a server, which is behind a firewall, and which requires authentication and authorization. The example in Figure 3 depicts the details of the connection establishment from a client to a server. First, the client connects via SSH to the SSH Daemon (sshd) on a remote machine behind the firewall authenticating the client user. Second, a local port forwarding is installed from localhost:8001 to 10.1.1.x:8001, on the computer, where the VAT4Net server application is running. Finally, the client socket will be connected to the start point localhost:8001. The server socket is listening on 10.1.1.x:8001 when the server has been started. When the client submits data, it will be automatically transported via localhost:8001 through the SSH tunnel to 10.1.1.x:8001 and vice versa. As soon as the client and the server have successfully established a connection, the server can start data transmission. Without any further command the server will only deliver trace lines to the client. A possible request from the client is to receive special plug-in data objects from the server. The delivery of the trace file is halted and the plug-in data is served. When the data has been received by the client, the delivery of the trace file data can be resumed.

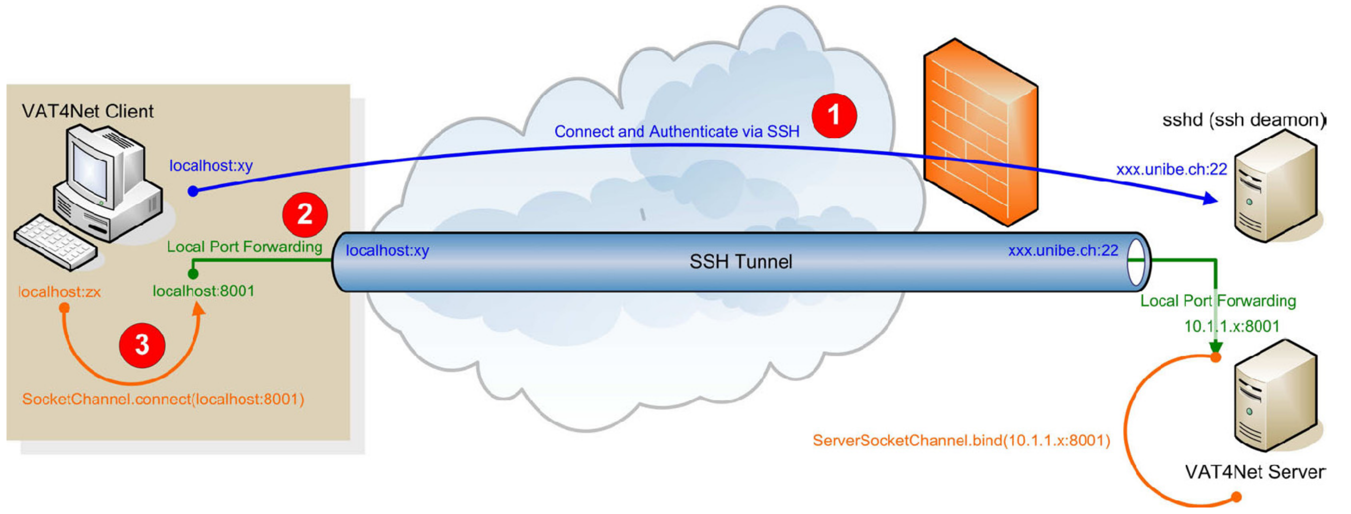


Figure 3. VAT4Net Connection Setup with SSH Port Forwarding.

3. ANIMATION, VISUALIZATION AND ANALYSIS

3.1. The Graphical User Interface (GUI)

The VAT4Net GUI (main frame) is divided into five parts (Figure 4). It is running in stand-alone or applet mode. The main menu contains all important commands for the whole application. The control panel gives access to all commands concerning animation. The animation process is displayed in the animation panel. The element info panel displays properties of nodes and links in a tree based view. All plug-in or statistics related views are shown in the statistics and plug-in panel.

Figure 5 shows the VAT4Net control panel, which enables to access all required functions provided by VAT4Net. The animation engine visualizes and animates the network topology with all its elements and events occurring in the simulation. Wired nodes, links, queues and wireless network grids are static elements, while wireless nodes and packets are moving objects.

After a part of the trace file is available in the buffer, either from a local or a remote source, VAT4NET will consume it and the animation can be initialized and started. The time controller acts as a supervisor on the animation. It further acts as a timer for the animation and invokes at the right time the necessary methods such as parsing new lines from the buffer, updating and drawing simulation elements and events. Figure 6 shows how VAT4NET processes trace files. The VAT4Net Parser converts trace lines read from the buffer into elements and events to be animated.

The network element is the main element in the animation process. It is aware of all other elements and events in the animation. It controls all updating and drawing calls. A node is a basic structure of ns-2 and stands for entities like routers, terminals, servers or mobile devices. The node types are not distinguishable from each other in the simulation. Therefore, they have the same shape in the animation, but they can get different colors.

Wired nodes are connected through links to each other. Placing wired network nodes over the animation panel is done in the initialization process of the network animation. In contrast to wireless networks, there is no location information available in the resulting trace files from almost all wired network simulations. In a first approach, the graph drawing algorithm of NAM was implemented in VAT4Net. The resulting networks drawn by this algorithm were rather unstructured. When displaying more than ten wired nodes, the resulting graph has a lot of overlapping nodes and links (Figure 7). While the algorithm from NAM tries to place the nodes randomly over the animation panel without evaluating the network structure, the newly chosen algorithm implements the spring model, where the calculation is based on forces between nodes. The new algorithm is a modification of the spring-embedder model of Eades [8][9]. It distributes the vertices evenly in the frame, minimizes edge crossings, makes edge lengths uniform, and reflects inherent symmetry, Compared to the old algorithm the new one consumes less memory, is significant faster and gives useful results of node placing (Figure 8). Wireless nodes are not affected by the node placing problem. Their position information is available from the trace file (Figure 9).

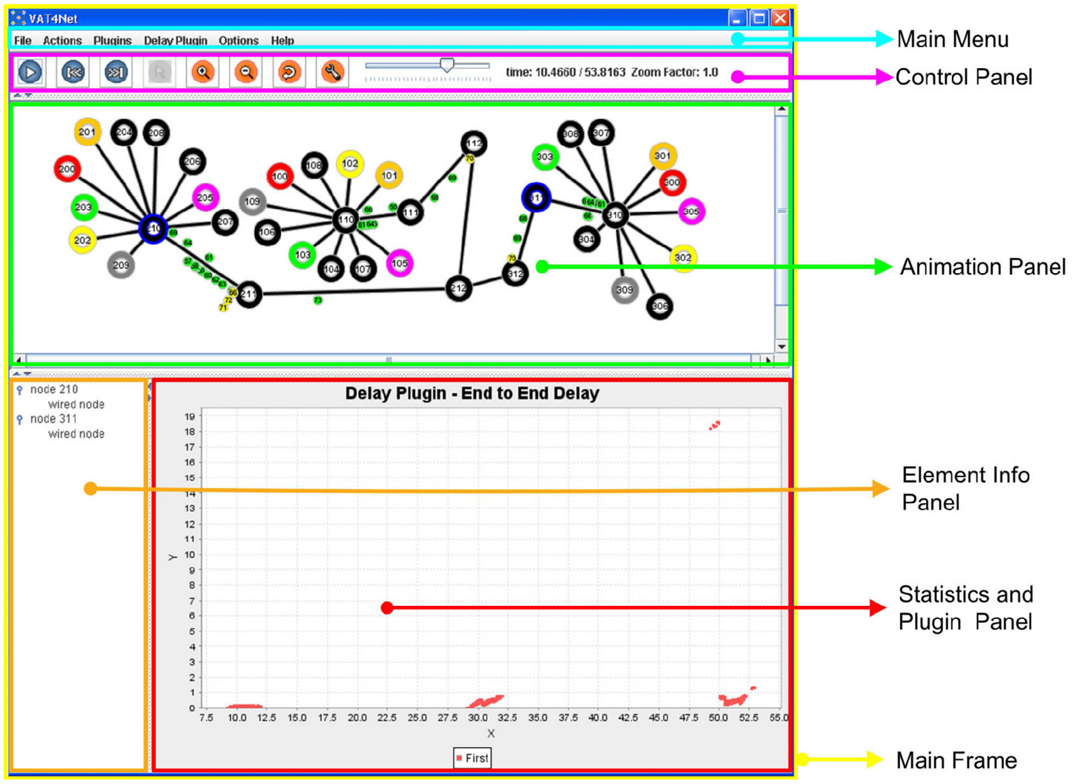

VAT4Net
File Actions Plugins Delay Plugin Options Help
time: 10.4660 / 53.8163 Zoom Factor: 1.0
node 210
wired node
node 311
wired node
Delay Plugin - End to End Delay
Y
X
First
Main Menu
Control Panel
Animation Panel
Element Info Panel
Statistics and Plugin Panel
Main Frame

Figure 4. VAT4Net GUI.

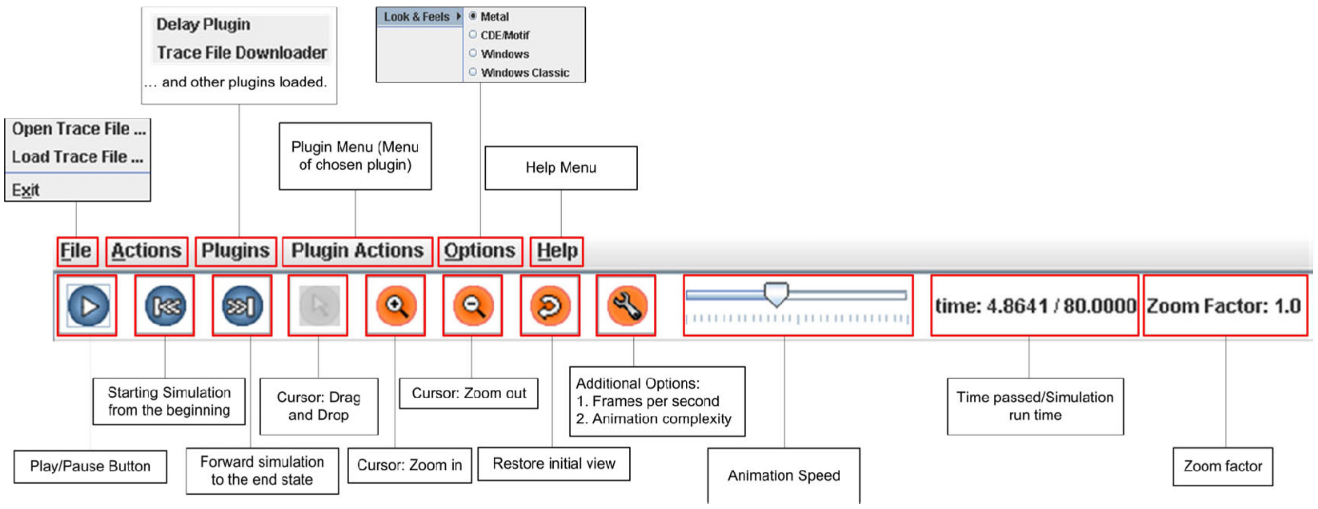


Figure 5. VAT4Net Menu and Animation Control.

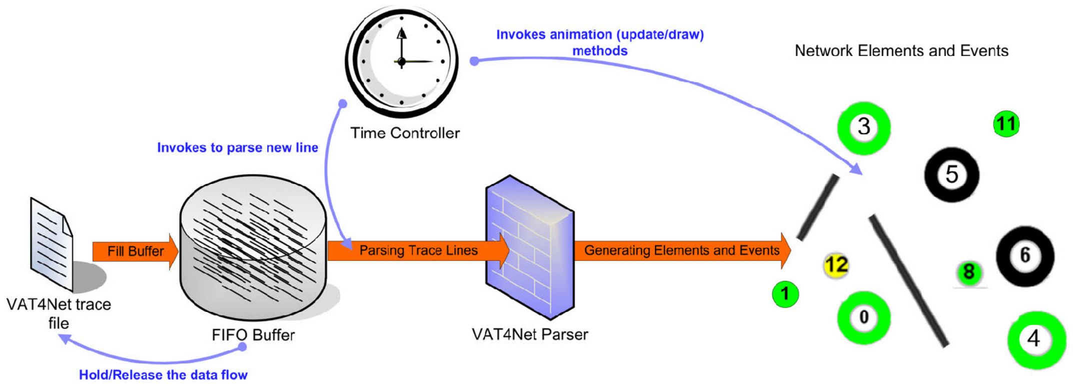

Invokes animation (update/draw) methods
Network Elements and Events
Time Controller
Invokes to parse new line
Fill Buffer
Parsing Trace Lines
Generating Elements and Events
VAT4Net trace file
FIFO Buffer
VAT4Net Parser
Hold/Release the data flow

Figure 6. VAT4Net Animation Engine Workflow.

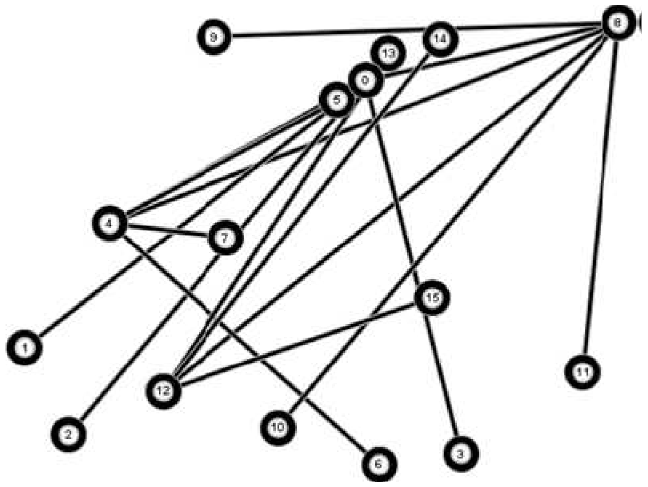


Figure 7. Node Placement by NAM.

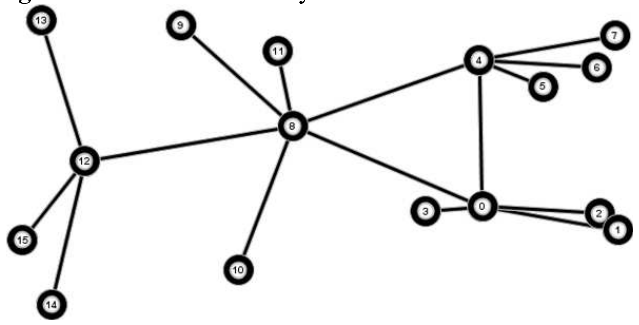


Figure 8. Node Placement by Spring-based Algorithm.

3.2. Plug-ins

The analysis and add-on engine has been implemented to support modular implementation of plug-ins. Those plug-ins are implemented as modular packages with predefined interfaces to the VAT4Net application. This allows easy enhancements of new functionalities required by users. The plug-ins are responsible for all actions and functions in addition to the animation of network simulations.

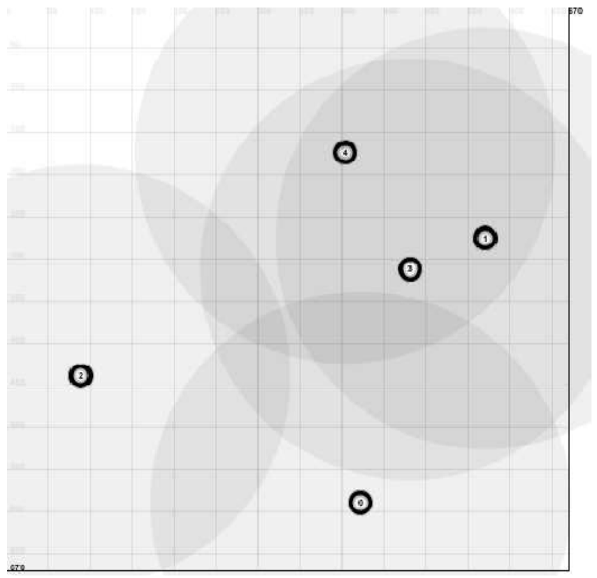


Figure 9. Wireless Network in VAT4Net.

Primarily this part is able to process statistical data, e.g., end-to-end delay, packet loss rate and other Quality of Service characteristics. It is also possible to add further functionalities, e.g., a file downloader for remote trace files, a file uploader, a trace file generator.

The data flow in case of plug-ins is shown in Figure 10. First, the user has to select a plug-in from the plug-in menu. The plug-in and the corresponding plug-in menu are activated (step 1). As soon as the plug-in is activated, the user has to apply settings required by the plug-in and to select the preferred action the plug-in provides (step 2). Then, the plug-in starts processing and delivers the data to the user. If the application runs as a stand-alone system the plug-in process is executed on the local computer. Otherwise, if VAT4Net is running in client/server mode, the plug-in calculation process is executed on the server. For statistics plug-ins the VAT4Net trace file is in most cases the data source for calculating and preparing the output (step 3). As soon as the calculation and preparation process has been completed, the plug-in delivers the results to the VAT4Net application (step 4). The data is visualized in the statistics and plug-in panel or the action can be completed by the user, e.g., saving the trace file on the local computer (step 5).

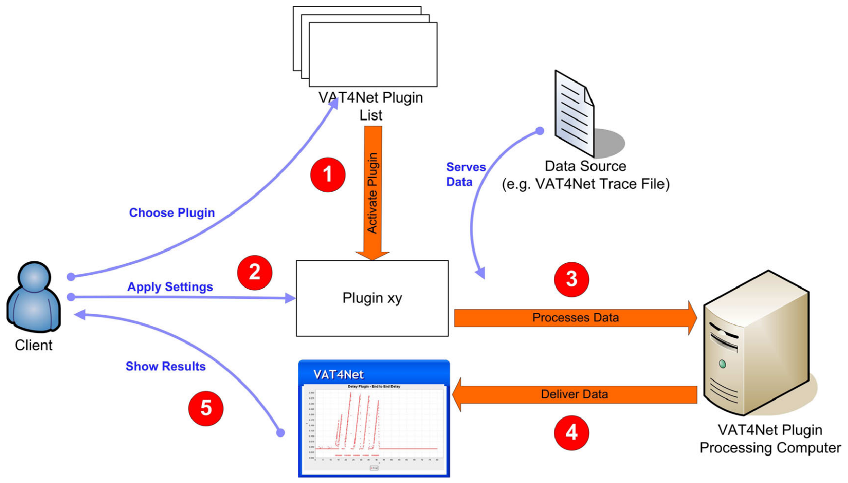


Figure 10. VAT4Net Plug-in Architecture

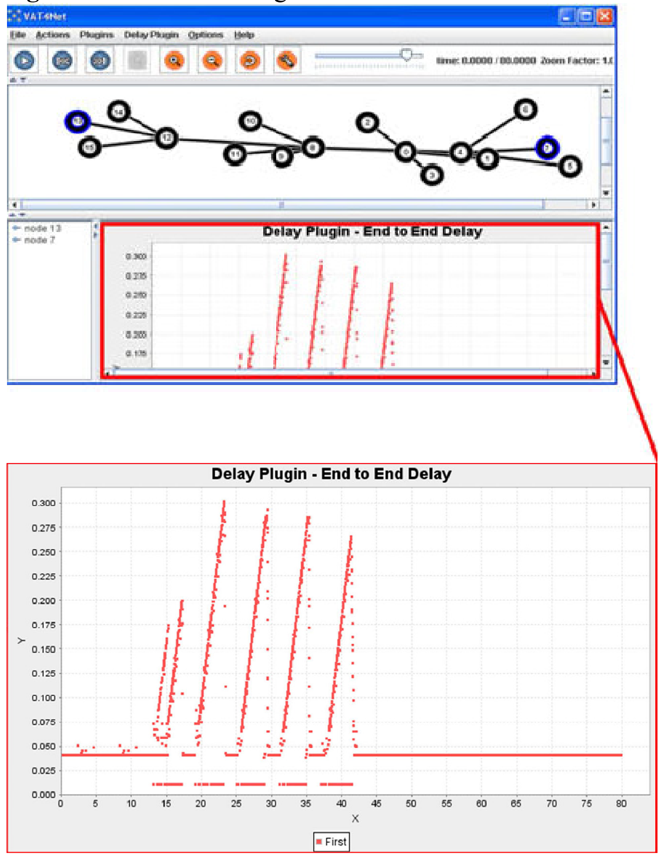

VAT4Net
File Actions Plugins Delay Plugin Options Help
time: 0.0000 / 00.0000 Zoom Factor: 1.0
node 13
node 7
Delay Plugin - End to End Delay
Y
X
First

Figure 11. VAT4Net Delay Plug-in.

An example for a plug-in is the delay plug-in, which calculates the end-to-end delay between two selected nodes. After loading a trace file into the animation engine, this trace file is also available for the delay plug-in. The user selects two nodes in the animation panel, for which he prefers to calculate the end-to-end delay. The plug-in calculates the statistical values and delivers it to the statistics and plug-in panel (Figure 11). The data is then shown in a chart with the simulation time as x-coordinate and the delay as y-coordinate. The chart is drawn with the help of the additional framework JFreeChart [10].

4. PERFORMANCE EVALUATION

Loading a trace file completely requires a lot of memory depending on the duration and level of detail of the chosen simulation. Therefore, instead of loading all trace files at once, trace files are “streamed” by buffering currently required data and releasing old data.

To test the effects of the buffer size on the whole animation process the occurrence of delays were measured. If the time used to calculate and to redraw one animation step is larger than the specified period of time, the step is considered as delayed. The tests have been executed on an Intel Pentium M 1.6 GHz, 512 MB RAM with a test VAT4Net trace file of 69 MB and 172’018 lines. The simulation has been performed with 16 wired nodes and the animation rate has been 10 frames per second.

The resulting delays of each test are shown in Figure 12. On the y-axis the number of delays (animation steps) occurring at an animation of a simulation is counted. The test scenario with a time step of 1 second (upper line) shows that large time steps cause a high amount of delays. For large time steps there is a lot of simulated data to be processed between two steps, independent of the size of the buffer. Large time steps require more computation time than smaller steps. Otherwise when doing small animation steps (0.5 seconds – middle line, 0.1 seconds – lower line), the buffer size has an impact on the delay. The animation is better for fewer delays of the animation process. Therefore, a buffer size of 37'000 KB would be the best for the test scenario.

5. VIRTUAL NETWORK SIMULATION

This section shows how VAT4NET and ns-2 have been integrated into a distance learning environment [2] allowing students to perform simulations and to visualize the simulation results remotely.

We provide a fully web accessible laboratory for network simulations including reservation system, laboratory portal server and laboratory computers. A reservation system [3] allows students the reservation of required (virtual) hardware in advance. The laboratory portal protects the laboratory equipment from unauthorized access via the Internet. The laboratory is based on a system, where multiple entities of an operating system can be run (host virtualization by User-Mode Linux). Each of these entities acts as a laboratory seat used by one user, who has reserved a time-slot in advance. This facilitates a completely separated, stand-alone working environment for each user with the possibility to resume work in a later time-slot or all to reset the laboratory to initial state.

Figure 13 shows the access to the simulation and visualization platform. Different clients (students, 1) want to work on their simulation experiments. After the clients have gained access rights to the Laboratory Portal Server (2), they are able to start with their work on the laboratory. The laboratory is running on the test-bed computer (3), which provides laboratory seats implemented by Use Mode Linux (UML, 4). UML allows multiple sand-boxed virtual instances of Linux to run as a stand-alone application on the Linux host system. When accessing the hands-on session for the first time, a standard predefined image of an UML entity is loaded and the client gets a blank and clean laboratory seat.

To allow resuming of laboratory work from earlier sessions, a so-called Copy-On-Write (COW) file is stored for each user. The COW file contains all changes performed by the user, such as newly generated files, system settings and changes to software. The COW file and the write protected pre-defined UML image form the current state of one user's laboratory work. Each user can reset the own laboratory seat to a previously stored state.

To access the laboratory, the user logs into a Laboratory Portal Server (Figure 14, 1), where he is forwarded to the command line interface of the assigned UML entity and logged in as root by automatic SSH key exchange. The Mindterm SSH applet [11] implementing SSH protocols in Java is used to connect the user with the laboratory seat. It provides the command line of the UML entity to which the user is assigned. The login process to the laboratory portal is accomplished by the Mindterm applet and the fact that the whole system works with single sign-on enabled by the so-called Authentication and Authorization Infrastructure [1] based on Shibboleth / SAML [4] protocols.

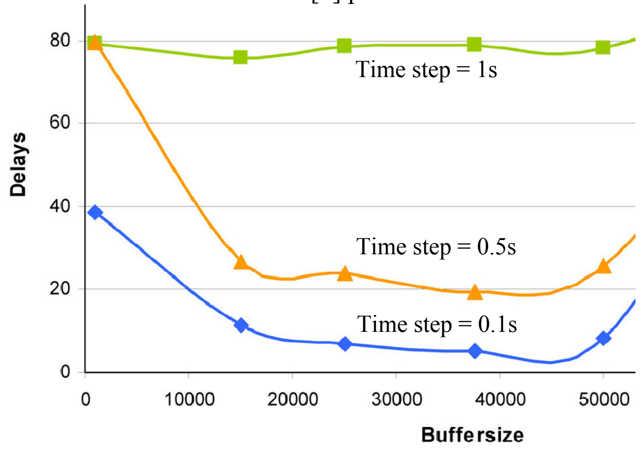


Figure 12. VAT4Net - Delay Dependent on Buffer Size [KB].

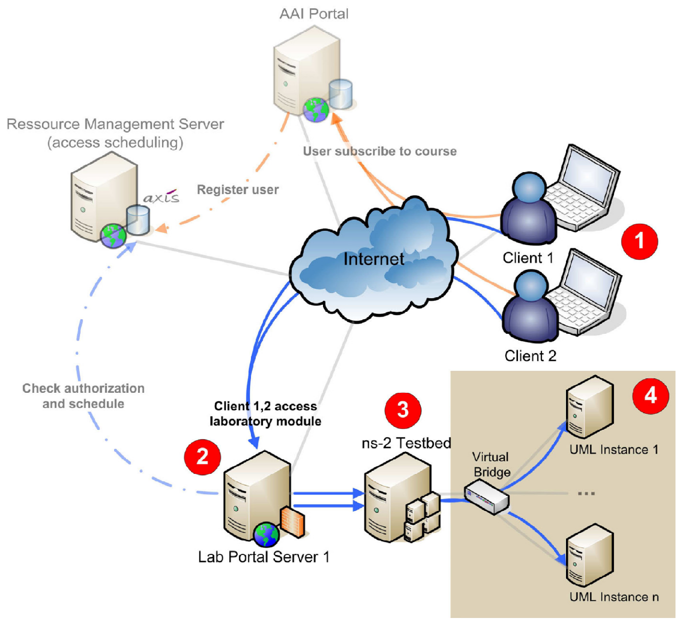
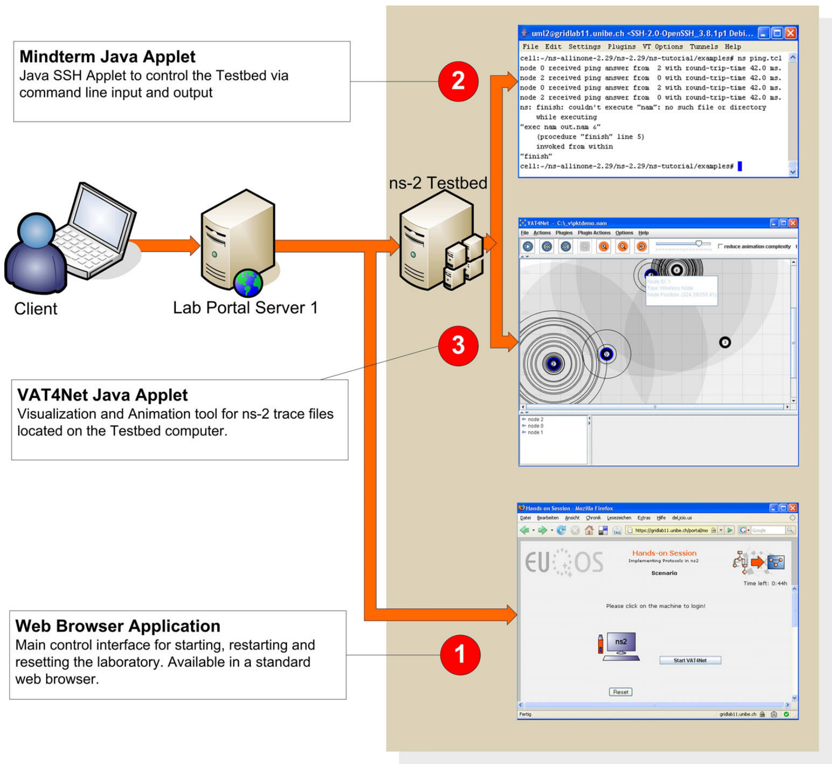


Figure 13. Integration of ns-2 and VAT4NET into Distance Learning Environment


Mindterm Java Applet
Java SSH Applet to control the Testbed via command line input and output
2
ns-2 Testbed
Client
Lab Portal Server 1
3
VAT4Net Java Applet
Visualization and Animation tool for ns-2 trace files located on the Testbed computer.
Web Browser Application
Main control interface for starting, restarting and resetting the laboratory. Available in a standard web browser.
1
uml2@gridlab11.unibe.ch <SSH-2.0-OpenSSH_3.8.1p1 Debi...
File Edit Settings Plugins VT Options Tunnels Help
VAT4Net - C:_v\pktdemo.nam
File Actions Plugins Plugin Actions Options Help
reduce animation complexity
Hands-on Session
Scenario
Please click on the machine to login!
ns2
Start VAT4Net
Reset

```
cell:~/ns-allinone-2.29/ns-2.29/ns-tutorial/examples# ns ping.tcl
node 0 received ping answer from  2 with round-trip-time 42.0 ms.
node 2 received ping answer from  0 with round-trip-time 42.0 ms.
node 0 received ping answer from  2 with round-trip-time 42.0 ms.
node 2 received ping answer from  0 with round-trip-time 42.0 ms.
ns: finish: couldn't execute "nam": no such file or directory
    while executing
"exec nam out.nam &"
    (procedure "finish" line 5)
    invoked from within
"finish"
cell:~/ns-allinone-2.29/ns-2.29/ns-tutorial/examples#
```

Figure 14. Interaction with Simulation Platform for Distance Learning.

6. SUMMARY

VAT4Net is a new animation platform for network simulators such as ns-2. Although it currently supports ns-2 only, other simulators can easily be supported due to the modular architecture by adapting the trace file pre-processor. An appropriate node placing algorithm has been designed and implemented as well. Implementing an efficient memory management has been one of the main challenges. This could be solved by a streaming solution. The solution allows supporting large trace files (up to 100 MB). The limit is mainly set by transmission delay limits. The implementation supports two modes, a stand-alone and a distributed mode. For the latter, VAT4Net has been split into a client and a server part. The architecture based on plug-ins allows changing the required functionality in a convenient manner. VAT4Net is available under GPL [18].

7. REFERENCES

[1] M. Steinemann, Ch. Graf, T. Braun, M. Sutter: Realization of a Vision: Authentication and Authorization Infrastructure for the Swiss Higher Education Community, Educause 2003 , November 7, 2003

[2] T. Braun, M. Steinemann: The Virtual Internet and Telecommunications Laboratory of Switzerland, ACM SIGCOMM 2003 Workshop on Networking Education , Karlsruhe, Germany, August 25 - 29, 2003, pp. 2-3

[3] S. Zimmerli, M. Steinemann, T. Braun: Resource management portal for laboratories using real devices on the Internet, ACM SIGCOMM Computer Communication Review , Vol. 33, July, 2003, pp. 145-151

[4] Shibboleth - an Internet2 middleware project, http://shibboleth.internet2.edu

[5] D. Estrin, M. Handley, J. Heidemann, S. McCanne, Y. Xu, and H. Yu, “Network visualization with the VINT network animator nam,” University of Southern California, Tech. Rep. 99-703, 1999.

[6] Java – Sun Developer Network, http://java.sun.com

[7] B. Scheuermann, H. Fuessler, M. Transier, M. Busse, M. Mauve, andW. Effelsberg, “Huginn: a 3d visualizer for wireless ns-2 traces,” in MSWiM ’05, 8th ACM International symposium on Modeling, analysis and simulation of wireless and mobile systems. New York, NY, USA: ACM Press, 2005, pp. 143–150.

[8] Eades, P., 1984. A Heuristic for Graph Drawing Congressus Numerantium, vol. 42, pp. 149-160.

[9] T. M. J. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement,” Software - Practice and Experience, vol. 21, no. 11, pp. 1129–1164, 1991.

[10] “JFreeChart - a free java chart library,” http://www.jfree.org/.

[11] “Mindterm Java Applet,” http://www.appgate.com/mindterm.

[12] K. Fall, “Network emulation in the VINT/NS simulator,” Proceedings of the fourth IEEE Symposium on Computers and Communications, 1999.

[13] A. Varga, “The OMNeT++ discrete event simulation system,” in European Simulation Multiconference (ESM’2001), June 2001.

[14] “ns-2,” http://www.isi.edu/nsnam/ns/index.html.

[15] “VINT - virtual internetwork testbed project,” http://www.isi.edu/nsnam/vint/index.html.

[16] K. Fall and K. Varadhan, “The ns manual (formerly ns notes and documentation,” VINT - Virtual InterNetwork Testbed Project, 2002.

[17] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, “Advances in network simulation,” Computer, vol. 33, no. 5, pp. 59–67, 2000.

[18] “VAT4Net - Visualization and Animation Tool for Network Simulations”, http://vat4net.sourceforge.net.