Visualization of Node Interaction Dynamics in Network Traces

Petar Dobrev, Sorin Stancu-Mara, and Jürgen Schönwälder

Computer Science, Jacobs University Bremen, Germany
{p.dobrev,s.stancumara,j.schoenwaelder}@jacobs-university.de

Abstract. The analysis of network traces often requires to find the spots where something interesting happens. Since traces are usually very large data-sets, it is often not easy and time intensive to get an intuitive understanding of what happens within a given trace. Through the use of suitable data visualization techniques, it is possible for humans to identify noteworthy spots or characteristics of a trace much faster. Particularly interesting properties of a certain class of network traces are node interaction dynamics, that is how the traffic matrix between nodes evolves over time and the pattern of messages exchanged between nodes. This paper presents some tools visualizing node interaction dynamics that were developed to assist network trace analysts.

Keywords: Network measurement, network visualization, SNMP, NetFlow.

1 Introduction

The collection of network traces is necessary in order to understand how network protocols are used in operational networks and to validate simulation models that are frequently used to evaluate new protocols. While the collection of network traces is technically relatively straightforward, it turns out that the analysis of the collected data is challenging due to the volume of network traces and the difficulty to identify the spots of a trace where interesting observations can be made. The later problem can be tackled by generating suitable visualizations so that the perception capabilities of humans can be exploited to identify spots effectively.

Most network trace analysis tools provide graphical displays of traffic over time, showing traffic breakdowns in different time resolutions. While useful to get a first overview, such traffic plots are not sufficient in order to develop a deeper understanding of the exchanges captured in a network trace. For understanding certain traces, it is essential to develop an understanding of the interaction dynamics. A simple example are network or port scans where the scanning strategy reveals some insights about the tools an attacker might have used.

Another example are traces covering a specific type of traffic that is expected to exhibit a certain behaviour. Some of our previous work has been related to

R. Sadre and A. Pras (Eds.): AIMS 2009, LNCS 5637, pp. 147–160, 2009.

the collection and analysis of network management traces, and in particular SNMP traces [1,2]. Initial results were published in [3] where we showed some simple static visualizations of the SNMP topologies discovered in our traces. While the simple static visualizations proved to be useful (one trace showed some unexpected anomalies due to dynamic address assignments), the static topology visualizations do not provide insights about the interaction dynamics. In particular, it is not possible to determine whether a data collection engine spreads the data retrieval traffic over a polling cycle or sends a burst of polling requests at the start of each cycle. Furthermore, it is not possible to see whether there are topology changes or if there are patterns of topology changes. (Note that topology changes on the management plane are usually caused by devices or links failing or returning back to their normal operational state.) To address these questions, we need to visualize the node interaction dynamics in a way such that it is possible to observe a pattern on a trace spanning days, even though the messages are exchanged with a round-trip time measured in the order of microseconds.

We started our investigation of node interaction visualization techniques by asking the following two questions:

- To what extent can existing tools help in visualizing node interaction dynamics?
- Is it possible to develop generic solutions that can be used for different types of traces?

The rest of the paper is structured as follows. We first review in Section 2 the state of the art in SNMP trace analysis and in network data visualization. We then describe an experiment to use the network animator (NAM) for the visualization of node interaction dynamics of SNMP traces in Section 3. Section 4 discusses a second experiment where we used a Java graph drawing library to visualize topology changes in SNMP traces. We briefly describe how we applied the tools to NetFlow traces in Section 5 before we draw our conclusions in Section 6.

2 Related Work

The Simple Network Management Protocol (SNMP) [4] is widely deployed to monitor, control, and (sometimes also) configure network elements. Even though the SNMP technology is well documented, it remains relatively unclear how SNMP is used in practice and what typical SNMP usage patterns are. In order to get a better understanding how SNMP is utilized by network management applications, an effort was started to collect SNMP traces from operational networks [3]. The Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF) produced an RFC [1] explaining the motivation behind the SNMP trace collection effort and specifying two trace storage formats that ease the exchange of traces between tools.

Static properties of several SNMP traces have been analyzed in [3] and a specific graphical representation has been introduced to visualize static SNMP flow topologies. Subsequent work to understand periodic pattern in SNMP traces, carried out at the University of Twente, made it obvious that a common terminology and a clear set of definitions are needed in order to produce results that are well defined and comparable. The common terminology resulting from this work has been published in [2]. Researchers at the Federal University of Rio Grande do Sul created an online system visualizing SNMP traces. Their system produces SNMP flow topology graphs similar to the graphs introduced in [3], plus MIB tree and time series graphs [5].

Most network data visualization work has focused on developing effective mechanisms to visualize large static data-sets. In the rest of this section we will review related work on topology visualizations and on NetFlow data visualizations.

Early work on the visualization of network topologies was done in the 1990s. The authors of [6] and [7] developed basic radial and geographic representations. More recent work often focuses on the visualization of Autonomous Systems (AS) routing topologies [8,9]. Linear IP address spaces are sometimes displayed in two dimensions using space filling curves [10,11,12], as this technique allows to show aggregations with common prefixes. To visualize network activities, three dimensional cube visualizations have been suggested, mapping source and destination addresses and port numbers [13,14]. These cube visualizations indicate interaction dynamics, although they can be hard to read due to the two-dimensional projection on computer displays.

NetFlow data is usually associated with time series displays such as those generated by RRDtool[1]. Radial layouts for visualizing NetFlow data were suggested in [15], while hierarchical network maps resembling treemaps were proposed in [16]. The work described in [17] proposes to display edge bundles on top of hierarchical network maps. The Isis system described in [18] provides visual flow data representations designed to make temporal relationships apparent and to allow for visual classification of events to reveal traffic structure. Most of these visualization systems focus on the visualization of static properties of NetFlow data. While the Isis system supports the interactive exploration of traces, it does not aim at animating node interaction dynamics.

3 Experiment #1: NAM

In our first experiment, we visualize the exchange of SNMP messages between managers and agents in order to highlight how SNMP monitoring engines distribute the polling load over time. While doing the conceptual design of the trace visualizer, it became apparent that *what* is being displayed should be decoupled from the *how* to display it. For describing what should be displayed we had to write our own tools because the SNMP trace format is relatively new and there are no powerful processing tools readily available yet. The comma separated values (CSV) SNMP trace format defined in [1] is very easy to parse in almost any programming language. Once the traces are parsed the flexibility of the programming language allows for any filter to be described and any output

[1] `http://oss.oetiker.ch/rrdtool/`

format to be used. We have used the programming language Python [19] in our experiment for it's rapid prototyping features. For visualization, we needed a tool that uses an open format and is able to display network topologies (graphs) and messages traversing the topology. We have chosen the popular network animator nam due to its ability to visualize the exchange of messages between nodes and our familiarity with this tool.

We first describe the network animator used for the experiment. We then describe how we convert SNMP traces into input files for the network animator before we present and discuss the visualization results.

3.1 Description of nam

The network animator[2] (nam) is distributed as part of the ns2 simulation software package. It is mainly used to visualize traces generated by the ns2 simulator. The nam tool is implemented in a mixture of C++, Tcl/Tk [20], and an object-oriented extension of Tcl. In general, the animation software is not easy to modify due to the high learning curve involved in understanding the interplay of the different programming languages involved.

The network animator reads a simulation trace file (also called a nam trace file), computes a graph layout for the recorded network topology, and afterwards it animates the messages passing over the links connecting the nodes. The graphical user interface provides controls to pause the animation, to change the animation speed, to zoom in and out, and to recalculate the graph layout or to manually reposition nodes.

The nam input file uses a relatively simple textual line-oriented format. We only describe a subset of the features here that we have used in our experiment. A nam trace file starts with a header containing version information, the list of nodes, and the list of links connecting the nodes. The version line has the following format:

```
V -t * -v 1.0a5 -a 0
```

The character V indicates that this is a version definition and the version number 1.0a5 is passed as the argument to the -v option. Nodes are defined using lines of the following format:

```
n -t * -a %d -s %d -S UP -v circle -c black -i black
```

The character n indicates that this is a node definition. The -t * parameter defines that this event does not have a time attached. The -a and -s parameters define the address and the id of the node. The -S parameter defines the node status while the -v parameter specifies the shape of the node and the -c and -i parameters specify the color of the node. Links connecting nodes are defined using lines of the following format:

```
l -t * -s %d -d %d -S UP -r %d -D %f -c black
```

[2] http://isi.edu/nsnam/nam/

The character `l` indicates that this is a link definition while the `-t *` parameter again indicates that this definition does not have a time attached. The `-s` and `-d` parameters specify the id of the source and destination nodes. The `-S` parameter defines the link status and the `-c` parameter the link color. The `-r` and `-D` parameters specify the data transmission rate of the link and the delay associated with the link. Finally, the color header lines specify the color coding of message types:

```
c -t * -i %d -n %s
```

The character `c` indicates that this is a color definition. The parameter `-t *` again indicates the time of the event. The parameter `-i` specifies the id of the color while the `-n` parameter carries a color name (according to the X11 color database). Following the `nam` file header, network message events are encoded using the following format:

```
h -t %f -s %d -d %d -e %d -c %s -i %d -a %d
```

The character `h` indicates a hop event. The parameter `-t` specifies the time of the event while the parameters `-s` and `-d` specify the source and destination node id of the message. The parameter `-e` carries the message size while the `-c` parameter carries a conversation identifier and `-i` a message id. The color of the message is specified using the `-a` parameter. Next to hop events, we use receive events. They have the same format in the `nam` trace file, except that the character `h` is replaced by the character `r`.

3.2 Conversion Algorithm for SNMP Traces

In order to use the network animator, we wrote a program to convert SNMP traces into the `nam` trace format. The program reads SNMP traces in the CSV format defined in [1]. In order to generate the `nam` header with the complete node and link list, our program needs to read the whole CSV file before `nam` output can be produced. In order to be able to share visualizations with other researchers and the public, we took care that our converter hides information such as IP addresses and absolute time-stamps.

For each transport address in the SNMP trace file, a node with a unique id is generated. The id is not derived from the transport address contained in the trace file. The links are inferred by determining all pairs of transport addresses that exchanged SNMP messages and translating this to source/destination node ids. For SNMP traces, the number of nodes and links encountered is usually small [3]. Finally, we generate for each SNMP message two nam trace events, namely a hop event and a receive event. The SNMP message events are color coded, allowing network analysts to distinguish easily between different request message types and responses.

An important issue is the time scale. Our visualization is designed to display traces covering weeks. However, in a real scenario, packets travel the network in fractions of a second. Any decent time scale that will allow a user to view the entire data-set will make the SNMP messages almost invisible. For this reason we tweak the network properties in the following way: we change the delay assigned to a link to 60 seconds. This is unrealistic, but it helps to make packets visible for longer time periods since it allows users to select faster than real-time timescales and still be able to see SNMP messages being sent. This change also helps to obtain a general overview over the network because the user actually sees all the data in the last 60 seconds. Another issue is the data rate of the links. It is used by the network animator to display the size of the message (in conjunction with the delay). For this reason, a 150 byte message on a 1Mbps link is too small to be displayed. Therefore we have also decreased the rate of the links down to 1kbps.

3.3 Results and Discussion

The generation of topology layouts is left to the network animator. Our conversion program does not provide any hints to the network animator concerning the graph layout.

Figure 1 shows a typical layout calculated by NAM. In this trace, two manager applications monitor an almost disjoint set of network devices. (Note that this layout is different from the layouts described in [3] since it is based on the network addresses of the nodes involved and does not distinguish different SNMP flows to the same endpoints.)

Figure 2 shows screenshots from a visualization of trace l06t01 (see [3] for more details about this trace). The left part (a) shows a manager polling devices by sending requests at about the same time to many devices. The devices respond with roughly the same delay causing subsequent requests to be sent out

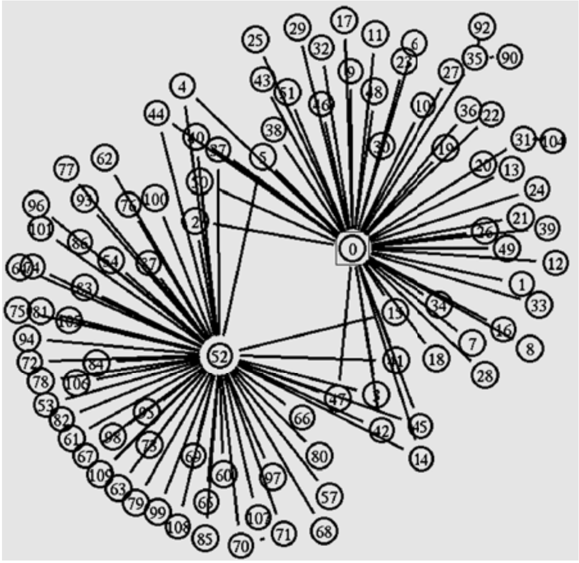
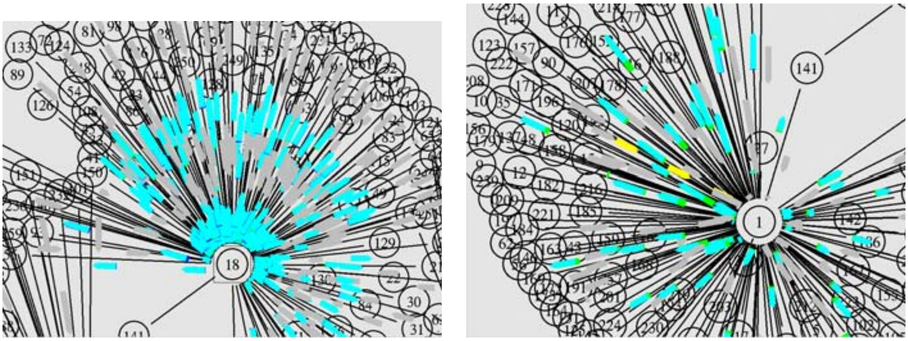


Fig. 1. Network layout calculated by `nam`



(a) Message waves

(b) Dispersed messages

Fig. 2. SNMP interaction dynamics visualized in `nam` (trace l06t01)

at roughly the same time as well. This leads to message "waves" in the visualization. The right part (b) shows a manager distributing the messages well over time, thereby avoiding bulky request/response waves. Part (b) also shows notification messages originating from one agent to inform the manager of some events.

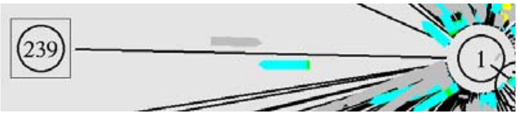


Fig. 3. SNMP message interaction details

Figure 3 shows some details highlighting the visualization of request/response interactions over a single link. Due to the adaptation of the link properties (delay and data rate), a request and the corresponding response usually appear traveling on a link at the same time.

4 Experiment #2: NetViz/JUNG

In our second experiment, we aim at visualizing the changes of the manager / agent topology over time by calculating topology-change dynamics graphs. A topology-change dynamics graph is a undirected graph showing the relation between nodes based on the recorded activity. If a message between node X and node Y has been seen in the last t_{link} seconds, there is an edge between the nodes in the graph. If no activity has been seen on the link for a period of t_{link} seconds, the edge between the nodes is removed. Once a node remains with no active links attached to it (no edges on the graph), a timeout timer is fired and if no link attaches to it for t_{node} seconds, the node is removed from the graph. In general, one can expect that the relationship between SNMP managers and agents is rather static. Changes usually occur if there is a (notable) event in the network (devices powering up / down, notifications emitted that do not happen regularly). To understand a trace, it is therefore a good approach to find such topology changes and to further analyze them.

4.1 Description of NetViz/JUNG

In order to create topology-change dynamics graphs for traces that can be very large in size, we decided to take a two step approach similar to the way the network animator works. We have implemented a graph animation tool called NetViz that reads an ordered timestamped list of events as input (addition of a vertex, addition of an edge, removal of a vertex, removal of an edge) and runs the animation. An additional program is used to read trace files and to produce the intermediate file containing a timestamped list of events. One benefit of this approach (besides increased modularity) is that animation scripts are usually several orders of magnitude smaller than the original trace files they are derived from.

The animation file format is very simple. Each line has the following basic format:

```
<timestamp> <action> <identifier>
```

The `<timestamp>` of the first line contains an absolute timestamp while all subsequent timestamps are relative. The `<action>` is one of `Va`, `Vr`, `Ea`, `Er` (vertex add/remove, edge add/remove) and the `<identifier>` is a unique identifier for the vertex / edge to be added / removed.

For the visualization of the topology, we used the Java Universal Network / Graph Framework[3] (JUNG), a software library for the modeling, analysis, and visualization of data that can be represented as a graph or network [21]. The library supports a number of different graph layout algorithms, clustering algorithms, and user interface controls such as lenses. The library makes it easy to write Java [22] programs that dynamically update the graph by adding nodes and edges while the displayed layout is dynamically recomputed. We added a simple graphical user interface (GUI) with a few controls around the JUNG framework.

4.2 Conversion Algorithm for SNMP Traces

The conversion program reads an SNMP trace in the CSV format defined in [1] and internally constructs a graph. Nodes are identified by their IP addresses and edges are identified by the IP addresses of the endpoints, independent of the direction of message exchanges. To each node and edge, there is a time-to-live counter attached, initialized with the values t_{node} and t_{edge}, passed as command line arguments. The timer for an edge starts running from the moment it is added to the graph, while the timer of a vertex starts running when there are no more edges connecting this vertex to other vertexes. The default value for t_{node} and t_{edge} was set to 600 seconds.

[3] `http://jung.sourceforge.net/`

4.3 Results and Discussion

The graph visualizer reads the animation file and draws the graph as it changes over time. The placement of the nodes on the canvas is done according to a relaxation algorithm, which considers each edge as a spring and finds the node configuration with minimum potential energy. When vertexes or edges are inserted or removed, the new configuration is relaxed until it reaches a minimum again.

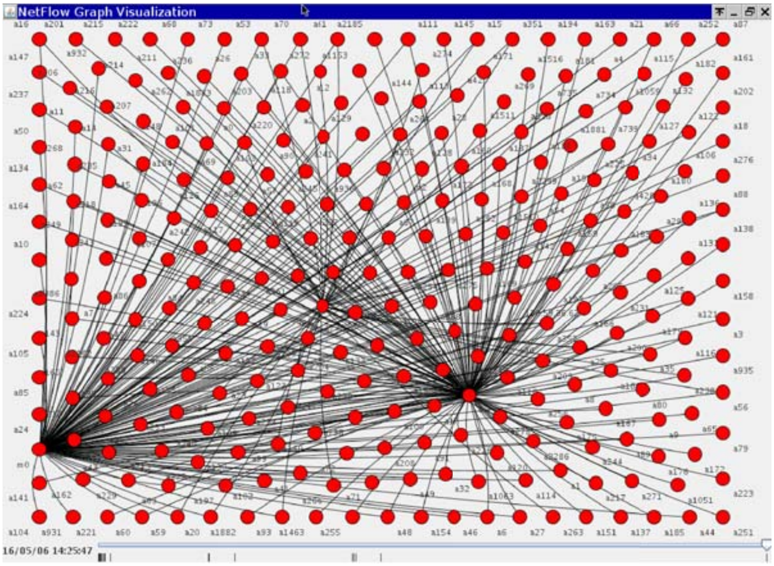


Fig. 4. Dynamic network layout calculated by NetViz/JUNG (trace l06t01)

The GUI of the application, shown in Figure 4, provides a label with human readable time of the replay, based on the timestamps, which are in seconds since the Epoch, and a timeline with the regions of activity marked on it. Since in most of the cases the configuration of the graph remains intact for most of the time and just has several peaks of activity over the whole time period, running the animation at a constant speed is somewhat boring as nothing is happening most of the time. To deal with this, we introduced an idle speed at which the animation runs when there is no activity. Once it gets close to some activity, the animation switches back to normal speed, so that the user can see what changes exactly are taking place on the graph. The normal and idle speed can be passed as command line arguments, or otherwise default to 300 real-world seconds per 1 second animation time for normal speed and 6000 real-world seconds for idle speed.

The application allows for interaction with the canvas by moving it with the mouse (pulling with the hand tool) and zooming in and out with the mouse wheel. Further controls to move the timeline and to pause the animation are possible but have not been realized yet at the time of this writing.

Figure 4 shows a typical situation of a few SNMP managers polling periodically a set of SNMP agents. The nodes are identified by labels indicating whether they act as a manager (labels starting with m) or an agent (labels starting with a). The timeline shown at the bottom of the window indicates when changes happen in the topology. Figure 5 shows a zoom-in on the timeline of the configuration of Figure 4. The topology changes at the beginning of the trace are usually caused by adding nodes and edges until a stable situation has been established and as such usually do not indicate events of special interest. (An option to address this could be to suppress topology change events in the first $t_{startup} = max(t_{node}, t_{edge})$ seconds).

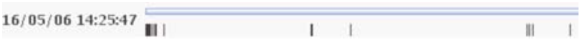


Fig. 5. Enlarged timeline display of Figure 4

Our experience using this tool and, in particular, the adaptive replay mechanism has been very positive. In fact, the activity timeline itself gives valuable insight into the traces since changes to the topology-change dynamic graphs likely indicate events of some significance. A shortcoming of the current implementation is, however, that the removal of nodes and edges can lead to rather drastic changes of the graph layout. To deal with this, we are experimenting with strategies where nodes and edges first become invisible for a while so that a node / edge disappearing and reappearing shortly later does not cause instabilities in the graph layout.

5 Application to NetFlow Traces

The tools described in Sections 3 and 4 were originally developed to visualize SNMP traces. It turned out that it was straightforward to modify the translators so that they can parse NetFlow [23] traces and produce intermediate files for the visualization tools. Of course, flow traces collected at backbone routers need to be filtered and / or aggregated appropriately for the tools to produce meaningful node interaction visualizations.

For the `nam` conversion, some adaptations are required since flow traces only record the start and end of a flow, plus the number of bytes and packets exchanged. In order to enhance the visualization, we decided to choose the size and density of the visualized packets solely based on the duration of a flow. The source emits packets as long as the flow is present. The packets travel slowly in order to enhance the visualization. This also has the nice effect that very short flows will still be visible, so it is possible to go faster through an animation without losing a lot of details.

We are currently interested to understand flow patterns generated by personal hosts and specific applications. In particular, we like to answer the question to what extend flow patterns can be used to identify machines or even users. We assume that flow pattern generated by a certain host or user can serve as a fuzzy fingerprint and we are trying to explore techniques to identify hosts or users based on such fuzzy fingerprints. In order to evaluate whether this is feasible, we started an effort to collect flow records on end user devices such as notebooks. These traces are reasonably small in size and with some limited filtering one can achieve useful visualizations of interaction dynamics in order to identify characteristic patterns.

Figure 6 shows a screenshot of `nam` visualizing the traffic generated by a notebook during one day. The screenshot shows some Web browsing activity of the user of the notebook at a certain point of time. By moving through the timeline, it is possible to easily identify periods of high activity and sites visited frequently. To further improve the visualization results, it is useful to pre-process the NetFlow traces by aggregating IP addresses so that, for example, different IP addresses of Google servers appear as a single node.

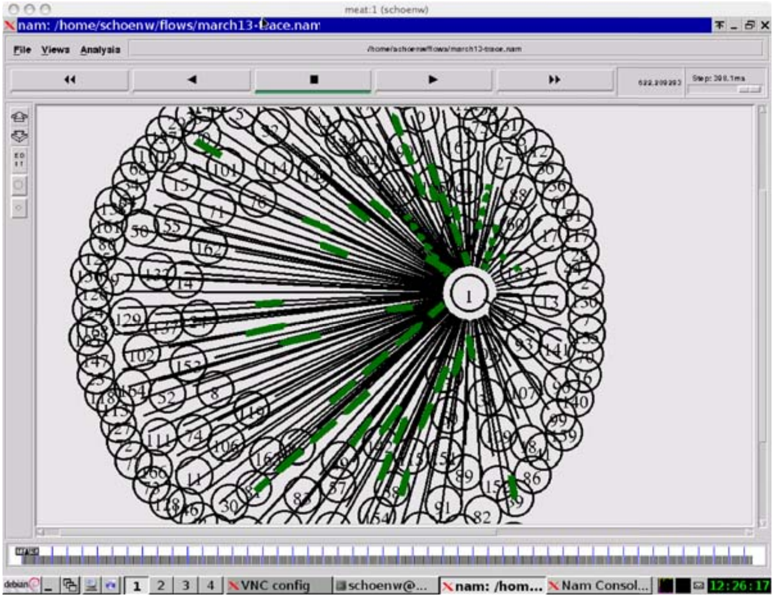


Fig. 6. Visualization of NetFlow traces using `nam`

6 Conclusions

Most existing visualizations of network traces focus on the static properties of the traces. Since our goal is to understand the dynamics recorded in network traces, we did some experiments to create visualizations of node interaction dynamics using readily available tools or libraries.

We found that with relatively little effort, it is possible to adapt tools or libraries to produce meaningful visualizations of interaction dynamics. However, effective integration of the tools / libraries remains a major problem. To be effective, a network analyst must be supported by multiple different visualization views of the same data and user interactions must be reflected on all views. To achieve this, much more development work is needed.

Our experience with `nam` is two-fold: On the one hand, it was very easy to get started since the `nam` intermediate file format is easy to generate and debug. However, the graph layout algorithm used by `nam` is not producing very satisfying results for larger graphs (e.g., ineffective use of the canvas space and many overlapping nodes) and the zooming capabilities help to to deal with this shortcoming only to some extend. Furthermore, we found that the speed of the animation does not scale well for larger trace files. As the `nam` tool is written in Tcl/Tk (with some parts in C/C++), it is not easy to extend it since the effort of learning how the `nam` tool has been implemented is relatively high.

Our experience with the JUNG graph drawing framework has been mixed as well. The graphs produced by the JUNG library generally look nice and using the Java library is rather straightforward due to good documentation and many readily available examples. While the performance of the graph layout algorithms is generally good, we did experience performance difficulties when we tried to animate more complex graphs. This is mainly a Java limitation since drawing on a canvas is relatively costly and does not exploit the capabilities of modern graphics hardware.

While we started with SNMP traces with very specific properties (dominant regular polling traffic with a relative stable communication matrix), it turned out that some of the tools we created could be adapted easily to deal with other trace formats. The decoupling of visualization software from specific data sources through intermediate file formats has proven to be a big win here. Unfortunately, some other openly available visualization tools we looked at do not support such intermediate formats and integrating their visualization capabilities or adapting these tools to different trace formats requires much more involved programming efforts.

Based on the experience we have gained by using `nam` and NetViz/JUNG for the visualization of node interaction dynamics, we are currently implementing a new visualization tool called `snam` that utilizes the Open Graphics Library (OpenGL) [24] and is able to take advantage of the capabilities of modern graphics hardware. We plan to further develop our tool towards an integrated trace visualization environment that consists of a loosely coupled set of visualization and data conversion tools that can be orchestrated as needed for a given trace analysis task.

Acknowledgement

The work reported in this paper is supported by the EC IST-EMANICS Network of Excellence (#26854). Thanks to Nikolay Melnikov for his comments that helped to improve the presentation of this work.

References

1. Schönwälder, J.: SNMP Traffic Measurements and Trace Exchange Formats. RFC 5345, Jacobs University Bremen (October 2008)
2. van den Broek, J., Schönwälder, J., Pras, A., Harvan, M.: SNMP trace analysis definitions. In: Hausheer, D., Schönwälder, J. (eds.) AIMS 2008. LNCS, vol. 5127, pp. 134–147. Springer, Heidelberg (2008)
3. Schönwälder, J., Pras, A., Harvan, M., Schippers, J., van de Meent, R.: SNMP Traffic Analysis: Approaches, Tools, and First Results. In: Proc. 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007), May 2007, pp. 323–332 (2007)
4. Case, J., Mundy, R., Partain, D., Stewart, B.: Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, SNMP Research, Network Associates Laboratories, Ericsson (December 2002)
5. Salvador, E., Granville, L.: Using Visualization Techniques for SNMP Traffic Analyses. In: Proc. of the IEEE Symposium on Computers and Communications (ISCC 2008). IEEE, Los Alamitos (2008)
6. Schulze, M., Benko, G., Farrell, C.: Homebrew network monitoring: A prelude to network management. In: Proc. SANS II (March 1993)
7. Pleitner, S., Brown, D.: Geotraceman: A Visual Traceroute. In: Proc. Australian Unix User Group (AUUG 1995) (September 1995)
8. Huffaker, B., Plummer, D., Moore, D., Claffy, K.: Topology discovery by active probing. In: Proc. Symposium on Applications and the Internet (SAINT 2002), pp. 90–96. IEEE, Los Alamitos (2002)
9. Boitmanis, K., Brandes, U., Pich, C.: Visualizing internet evolution on the autonomous systems level. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 365–376. Springer, Heidelberg (2008)
10. Munroe, R.: Map of the Internet. xkdc (2006)
11. Irwin, B., Pilkington, N.: High Level Internet Scale Traffic Visualization Using Hilbert Curve Mapping. In: Proc. of the Workshop on Visualization for Computer Security. Mathematics and Visualization. Springer, Heidelberg (2008)
12. Samak, T., Ghanem, S., Ismail, M.A.: On the Efficiency of using Space-Filling Curves in Network Traffic Representation. In: Proc. 11th IEEE Global Internet Symposium. IEEE, Los Alamitos (2008)
13. Lau, S.: The Spinning Cube of Potential Doom. Communications of the ACM 47(6), 25–26 (2004)
14. Oberheide, J., Goff, M., Karir, M.: Flamingo: Visualizing Internet Traffic. In: Proc. 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), pp. 150–161. IEEE, Los Alamitos (2006)
15. Keim, D.A., Mansmann, F., Schneidewind, J., Schreck, T.: Monitoring Network Traffic with Radial Traffic Analyzer. In: Proc. IEEE Symposium on Visual Analytics Science and Technology (VAST). IEEE Computer Society, Los Alamitos (2006)
16. Mansmann, F., Vinnik, S.: Interactive Exploration of Data Traffic with Hierarchical Network Maps. IEEE Transactions on Visualization and Computer Graphics 12(6) (November 2006)
17. Mansmann, F., Fischer, F., Keim, D.A., North, S.C.: Visualizing large-scale IP traffic flows. In: Proc. 12th International Workshop Vision, Modeling, and Visualization (November 2007)

18. Phan, D., Gerth, J., Lee, M., Paepcke, A., Winograd, T.: Visual Analysis of Network Flow Data with Timelines and Event Plots. In: Proc. Workshop on Visualization for Computer Security (VizSec 2007). Mathematics and Visualization, pp. 85–99. Springer, Heidelberg (2007)
19. van Rossum, G., Drake, F.: The Python Language Reference Manual. Network Theory Limited (2003)
20. Ousterhout, J.K.: Tcl and the Tk Toolkit. Addison-Wesley, Reading (1994)
21. O'Madadhain, J., Fisher, D., Smyth, P., White, S., Boey, Y.B.: Analysis and Visualization of Network Data using JUNG. Journal paper under preparation
22. Campione, M., Walrath, K.: The Java Tutorial: Object-Oriented Programming for the Internet, 2nd edn. Addison Wesley, Reading (1998)
23. Cisco Systems: NetFlow Services Solution Guide (January 2007)
24. Segal, M., Akeley, K.: The OpenGL Graphics System: A Specification. Version 3.1, The Khronos Group Inc. (March 2009)