Network Stack Diagnosis and Visualization Tool

Krist Wongsuphasawat, Pornpat Artornsombudh, Bao Nguyen, and Justin McCann
University of Maryland, College Park, MD 20742
kristw@cs.umd.edu, apornpat@umd.edu, baonn@cs.umd.edu, jmccann@cs.umd.edu

ABSTRACT

End users are often frustrated by unexpected problems while using networked software, leading to frustrated calls to the help desk seeking solutions. However, trying to locate the cause of these unexpected behaviors is not a simple task. The key to many network monitoring and diagnosis approaches is using cross-layer information, but the complex interaction between network layers and usually large amount of collected data prevent IT support personnel from determining the root of errors and bottlenecks. There is a need for the tools that reduce the amount of data to be processed, offer a systematic exploration of the data, and assist whole-stack performance analysis.

In this paper, we present Visty, a network stack visualization tool that allows IT support personnel to systematically explore network activities at end hosts. Visty can provide an overview picture of the network stack at any specified time, showing how errors in one layer affect the performance of others. Visty was designed as a prototype for more advanced diagnosis tools, and also may be used to assist novice users in understanding the network stack and relationships between each layer.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: Information Interfaces and Presentation

Keywords

Visty, Network Stack Visualization

1. INTRODUCTION

While using networked software, end users often encounter unexpected situations, such as stuttering and delay in video chat, disconnections from instant messaging servers, and slowly loading web pages and applications. Users often have little insight into the causes of such poor performance, leading to frustrated calls to the help desk. Network administrators and IT support personnel are left to perform labor-intensive troubleshooting, but determining the root cause of the problem is difficult because of the complexity of modern systems.

A large body of research has focused on monitoring and diagnosis of services over the network. However, determining whether errors and bottlenecks are caused by software or the network infrastructure is still a complicated problem. The key to many proposed approaches is using cross-layer information to aid analysis [5, 6].

To gather consistent cross-layer information, we created a data collection tool that captures statistical snapshots from each layer of the network stack on an end host. To capture performance problems over periods of a few network round trip times, these snapshots are taken approximately every 100 milliseconds. However, this can result in a very large amount of data in a short period of time. Even if an expert understands the significance of each protocol statistic and cross-layer interaction, analyzing this amount of data and keeping track of complex interactions across the network stack are still challenging tasks. Therefore, an analysis tool is required that can reduce the amount of data to be processed, offer a systematic exploration of the data, support cross-layer network traffic analysis and suggest interesting information.

Visualization is the use of graphical techniques to help people see, explore, and understand large amounts of information at once. Network data visualization has a long history [7, 9, 10, 14, 18, 19, 22, 24], but most approaches focus on large-scale network diagnosis tasks, and are not designed to troubleshoot problems with a particular machine, which is what IT help desks often encounter.

Hence, to support network problem analysis from a single machine perspective, we created a simple visualization tool called *Visty*, which has the following contributions:

1. Visty allows IT support personnel to explore the relationship of end host protocol statistics over time, and provides them with an overview picture of the network stack at any specified time to help locate performance problems seen by end users.
2. Visty serves as a prototype for more advanced diagnosis tools.
3. In addition, Visty can be used to assist novice users and developers in understanding the network stack and relationships between each layer.

Our design approach is according to Shneiderman's information visualization mantra: overview, zoom and filter,

CHIMIT'09, November 7–8, 2009, Baltimore, Maryland.

details on demand [23]. An abstract representation of the network stack and a timeline are used to reduce the amount of data shown on screen and to provide an overview of the network stack over time; zooming and filtering are employed to provide systematic exploration; color-coding techniques suggest interesting spots in the data; and detail information is hidden until it is requested.

Visty loads the data provided by our end host data collection tool and creates the network stack visualization. Users first select a period of time from the timeline that displays all running applications over time. By relating all in-layer and cross-layer information within the selected period of time together, Visty can build the network stack for that timespan. It allows the users to see all traffic being passed over the network stack. Users can then use zooming, filtering and color-coding mechanisms to further explore the data.

The rest of this paper is organized as the following: Section 2 covers the relevant history of network diagnosis and network visualization tool. Section 3 describes motivation. Section 4 defines terms that are used throughout the paper. Section 5 explains high-level design concepts. We introduce Visty and discuss implementation in Section 6. We describe use case example and findings from collected data in Section 7 and conclude in Section 9.

2. RELATED WORK

A large body of research on network performance monitoring and diagnosis had been done. Many researchers had presented ideas that were relevant to this study. We divided these related researches into two areas: *cross-layer network diagnosis* and *network traffic visualization tool*.

2.1 Cross-Layer Network Diagnosis

Some approaches to network diagnosis focus on performance of a single layer [16, 17, 20, 26, 27]. While such approaches are useful, they cannot give an accurate picture of overall system performance since losses at one layer may or may not cause problems at other layers. For example, backoff and retransmissions on a wireless link do not necessarily mean that higher-level protocols such as TCP are limited by the wireless link; TCP losses may be occurring somewhere else in the network.

Some enterprise network researchers use a cross-layer view to detect the causes of network defects. There are two different approaches in enterprise network diagnosis: centralized and decentralized.

On the centralized side, Cheng et al. [12, 13] developed the *Jigsaw* framework for enterprise wireless network diagnosis. From raw frame data, Jigsaw reconstructs a complete description of all link and transport-layer conversations. Using this synthesized information, Jigsaw provides a global view of the network. However, to achieve this goal, Jigsaw requires a centralized system to collect and monitor data from all network nodes. In other words, Jigsaw focuses on large scale monitoring tasks while our work here focuses on the single host viewpoint.

On the other hand, Chandra et al. [11] developed a decentralized system called *WifiProfiler*. Unlike Jigsaw, WiFiProfiler relies upon peer diagnosis among clients without the involvement of system administrators. The analysis tool must be installed on every client to collect data. Those data was later exchanged between peers. The clients could then use the exchanged data to analyze across the network layers to locate the problems. Although WiFiProfiler was designed from the end user perspective, it requires cooperation between users, which could be difficult to achieve.

In this study, we also use cross-layer information to solve network diagnosis problems. However, our approach is different from the other alternatives because it is based on visualization rather than rule-based inference or data mining techniques.

2.2 Network Traffic Visualization Tool

Other approaches to network data visualization focus on network traffic diagnosis tasks. One of the earliest works in this theme was SeeNet by Becker, et al. [9]. SeeNet visually represents the amount of data being sent between two network nodes using three visualization techniques: static displays, interactive controls, and animation. Two of these use geographical relationships, while the third is a matrix arrangement that gives equal emphasis to all network links. With this tool, an administrator can easily identify overloaded nodes as well as the data flow between them.

Later work includes from Ball [7], Patwari [22] and Kim [18] describes various visualization techniques applied to support network traffic analysis and management tasks. *NetGrok* [10] employs zoomable interfaces and treemaps to make the network structure and traffic flow information more visible to network administrators.

Several authors have applied visualization techniques to network security problems. McPherson et al. [19] use histograms to summarize coarse data collected from TCP and UDP ports; activity levels are color-coded to help users easily uncover interesting security events. In a similar effort, Tehsin et al. [24] propose a visualization of packet headers to highlight the features of the network data most vulnerable to attacks.

Gerald Combs et al. [15] developed *Ethereal*, which was later renamed to *Wireshark*. Wireshark captures data from an ongoing network connection and perform offline and online analysis. It provides a graphical front-end and many information sorting and filtering options that allow the user to analyze traffic being passed over the network. Wireshark is able to display the packet encapsulation and protocol fields along with their interpretation for many different networking protocols. It also provides simple visualizations, such as simple timeline and throughput graphs. However, it does not provide a visualization for cross layer analysis. In addition, Wireshark depends on packet-level capture on a given interface. Our data collection tool collects protocol statistics across the entire host, and as such should be more scalable.

Most current network analysis tools such as WireShark[21], Nagios[8], and INAV[25], focus on traffic diagnosis tasks. Data is collected from different protocols and visualized in forms of log tables or summarized graphs. Some data analysis techniques such as sorting, filtering or color coding are applied to enhance the data readability. Those visualizations are useful in analyzing data from a single layer or identifying the data trends. However, it is not easy to interpret the relationships between particular data coming from different network layers. By organizing network data the way people think about it—as a protocol stack—and visualizing the connections between all protocol layers, Visty helps users explore and highlight the cross-layer relationships. Moreover, by taking the advantages of zoomable interfaces, Visty

users can analyze those relationships in different levels of abstraction.

The idea of visualizing data by querying data from different time slots is done in RRDTool based tools such as Cricket[3], Munin[4], and Big Brother[2]. However, these tools lack the ability to search time series, which would allows the users to effectively explore and identify the time slots of interest. In contrast, the timeline searching bar in Visty enables users to quickly navigate through the network data over time. Using this interactive interface, users can first scan the data as a whole to get some hints about problem areas, and then zoom into particular time slots to get the insights.

3. MOTIVATION

Applications which run over the network can encounter many problems. For example, an instant messaging application that was working fine could suddenly fail every outgoing message attempt, creating frustrated users. The IT help desk is then called to solve the problem. However, these kinds of problems can come from any point of the system, from the application code, network congestion, or even hardware error.

Locating the causes of the problem is a difficult task because the abstract interface provided between protocol layers does not include much diagnostic information. The main challenge is that each layer of the network architecture cannot be looked at in isolation. Errors or performance problems at each layer may be compounded by layers above and below, making it impossible for one layer to distinguish its own problems.

Furthermore, the amount of data collected via a network trace can be very large even for a very short period of time. For example, our data collection tool collects protocol statistics from the entire network stack approximately every 100 milliseconds. This amount of data requires significant time and effort to analyze without assistance from a proper analysis tool.

To the best of our knowledge, current network analysis and monitoring tools do not provide an appropriate support for cross-layer network traffic analysis, which can relate and track the cross-layer abstraction. Therefore, there is a need for an analysis tool than can assist network administrators in analyzing cross-layer information from a single machine viewpoint.

Visualization takes advantage of human visual perception to allow users to see, explore, and understand large amounts of information at once. This makes it possible to improve the understanding of the network stack captured by the underlying data and the complex interactions between network layers.

Therefore, we created Visty, a network stack visualization tool, which displays a graphical representation of the network stack and relationships between each layer, and allows users to explore the data via user interaction. Unlike other network tools, which focus on packets captured at network interfaces, Visty visualizes the flow of network traffic across layers within a host. While packet captures are useful, they provide little insight when packets are not transmitted— software bugs, slow applications, and backoffs to avoid contention are hidden to capture-based tools. By looking at the entire network stack, Visty enables users and systems administrators to diagnose which network layers are experiencing degraded performance. In the future, more features related to automated performance diagnosis can be added to assist analysis. Visty can also be used for educational purposes at the starting point of studying networks by showing examples of real network stacks to end users and developers unfamiliar with network protocols.

4. DATA DICTIONARY

Before discussing our implementation in detail, we define the entities in our network stack ontology:

A *Module* is a generic type of modules in the network layer, e.g., Firefox, TCP, IP, or Ethernet. If we compare this to object-oriented programming, Module will be like a class.

A *ModuleInstance* is an instance of a Module that exists over a period of time, e.g., Firefox(PID=#1), Firefox(PID=#2), IP 10.1.1.1, IP 192.168.0.1. A ModuleInstance is analogous to an object in object-oriented programming.

A *ModuleSnapshot* is a collection of a ModuleInstance statistics at a point in time, e.g., "Firefox(PID=#1) at 4/30/2009 11:01:03". All the data (number of bytes in, numbers of bytes out, number of errors, retransmissions) is collected here.

A *Relationship* is a relationship between ModuleInstance, e.g., Firefox(PID=#1) is above Socket(fd=7), which is above "TCP conn. 192.168.0.1:42295 ↔ www.google.com:http".

5. HIGH LEVEL DESIGN

The data used by Visty is collected over a period of time, which may be hours, days, or possibly weeks. Displaying the network stack for the entire data set may be not desirable, especially when the data is very large. Users specify a time range that they are interested in. Then, the first question users might encounter is, "When is the interesting time period that I should look at?" So, we design a timeline panel to give an overview of the data by showing the lifetime of all applications. Users then can specify the duration from the timeline, using the knowledge of when applications were running to help them make their decision.

After the users specify the time period, Visty fetches all the required information from the database and builds the network stack. A major goal of Visty is to reveal insight into every part of the networking stack, from device drivers to transport protocols to applications. The module instances (e.g., an application, TCP connection, or an Ethernet interface) are represented as boxes and placed on the screen according to their layers— applications at the top, down to physical interfaces at the bottom. Lines are drawn from each module instance to all its dependencies to express the relationships between module instances.

Sometimes, module instances are unrelated to the analysis of a given stack trace, for example when a particular application is being diagnosed. In such cases, users should be able to filter out these modules. Therefore, we add filtering mechanisms that allows users to remove such module instances from the stack.

Each module instance contains detail information: number of packets in, number of packets out, number of errors, etc. It is not possible to show all the information on the screen at the same time. Therefore this information is hidden by default. Users can click on a module instance to view detail information.

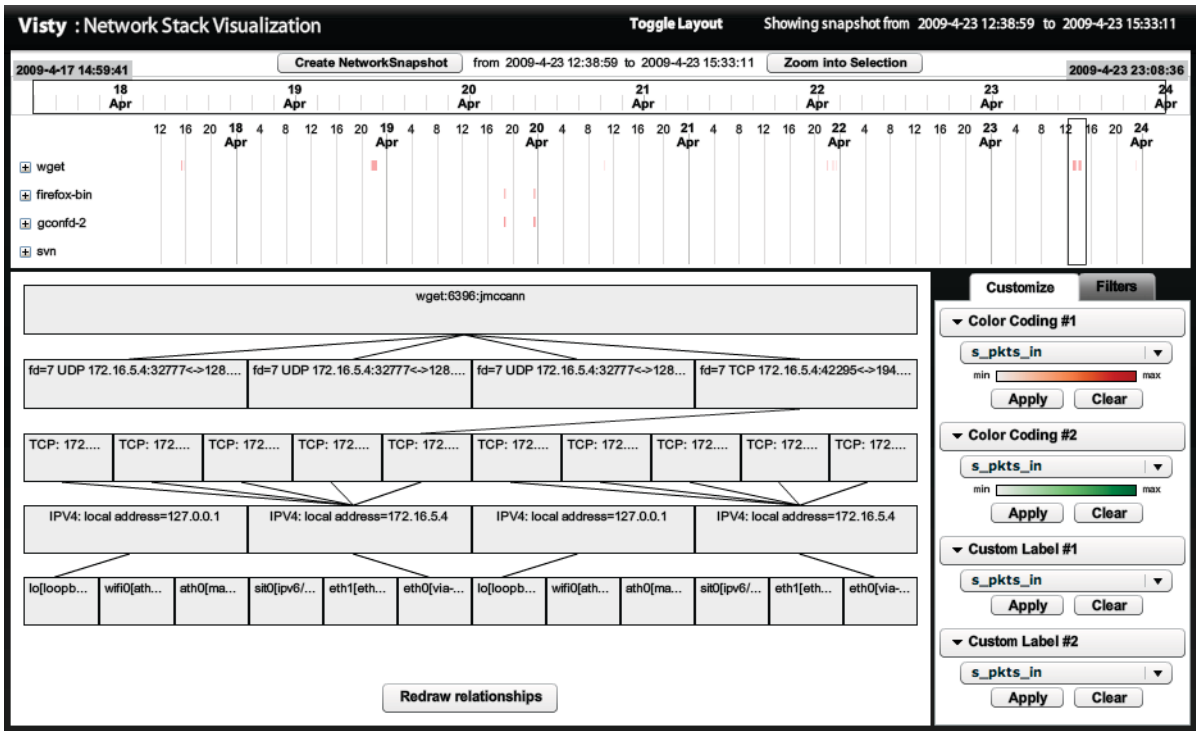

Figure 1: Visty - The timeline panel on the top displays lifetime of all applications. The user can draw a selection (black rectangle) on the timeline to specify the range of interest and click on "Create Network Snapshot". The result network stack is displayed in the bottom left part of the screen. The control panel in the bottom right can be used to specify color-coding scheme or additional labels on the module instances and filter non-relevant module instances. Please see filtered, labeled and color-coded network stack in Figure 2.

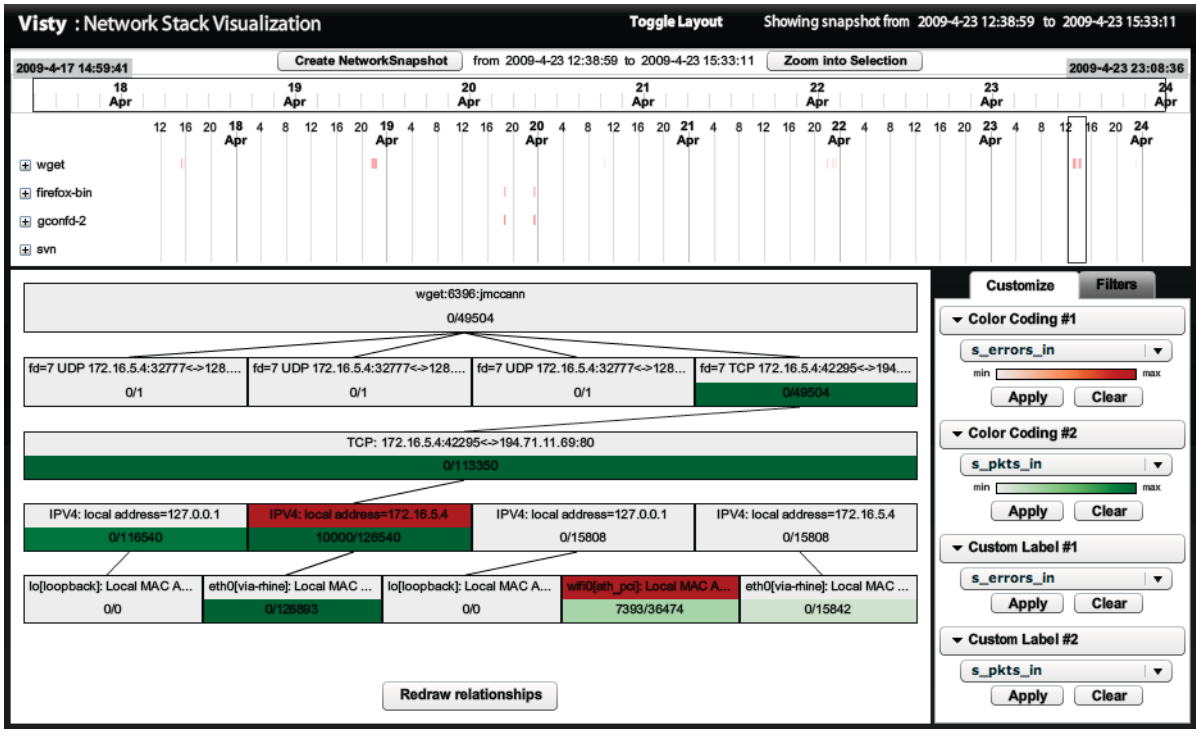

Figure 2: Network stack from 12pm to 3pm on Apr 23 - The users filtered out non-relevant instances, and use green and red to represent high numbers of incoming packets and errors, respectively. The first and second numbers under the module instance name are number of incoming errors and packets, respectively.

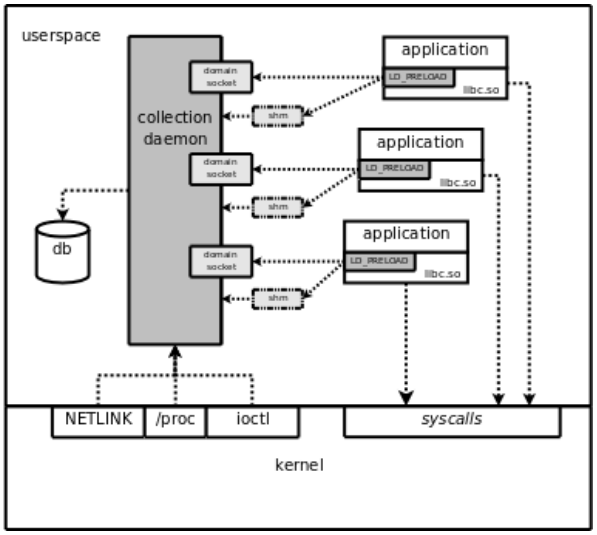

Figure 3: Data collection architecture. The collection daemon retrieves statistics from the kernel and a custom library call interposition module invoked via LD_PRELOAD, and stores them in a database for later analysis.

Moreover, we would like to provide an easy way for users to locate possible problems in the network stack. As a result, we add color-coding mechanisms to the tool. Users can select what data field (e.g., number of packets in, number of errors, etc.) they want to color-code the module instances by, and makes it easy to locate outliers.

6. IMPLEMENTATION

Visty is a web-based application developed using Adobe Flex 3. We use a library called asSQL [1] to connect to a MySQL database that contains statistical snapshots of the entire network stack, retrieved several times a second by the collector shown in Figure 3.

Visty consists of 3 panels: timeline, stack and control panel. (See Figure 1.) The timeline panel is located at the top. It shows lifetime of all the applications in the data. Users select a period of time from the timeline and click on "Create NetworkSnapshot" button to load data within that period from the database. Once the data is returned from the database, the network stack is visualized on the stack panel, which is located in the bottom left of the screen. The control panel in the bottom right corner of the screen provides filtering and color-coding mechanisms for further exploration.

6.1 Timeline Panel

This panel is designed to show an overview of the data and help users choose the time range that they want to see the network stack. There are two timelines in this panel. (See Figure 4.) The one on the top is called *main timeline*. This timeline shows the entire range of the data. Users can draw a box on the main timeline to specify the duration of data shown in the *detail timeline* beneath it. The detail timeline displays the lifetime of all application instances, grouped by application (e.g., wget, firefox, and svn) using color-coded rectangles. The darker the color of the rectangle is, the higher number of that application instances are running at that time. Clicking on the "+" symbol in front of the application name shows all instances of that particular application. Users can draw a selection box on the detail timeline to specify a time range to build the network stack. The label on the top of the timeline changes according to the selected range. Users can zoom into the selected range by clicking on "Zoom into Selection". When users have finished choosing the time range, they can click on "Create NetworkSnapshot" button to load data from the database and build the network stack.

6.2 Stack Panel

Once users have clicked on the "Create NetworkSnapshot" button, the network stack is visualized in the panel below. (See Figure 5.) A major obstacle to representing the network stack trace is the diversity of behaviors and features exhibited in network architectures. Communication functions are grouped as layers according to their common features. Different authors have interpreted the RFCs differently regarding whether the Link Layer (and the four-layer TCP/IP model) covers physical layer issues or a "hardware layer" is assumed below the link layer. In this paper, we use the OSI model as a reference.

Each module instance is represented as a box. Relationships between module instances are visualized as lines between the boxes. The boxes are placed vertically by layers and horizontally by time. The highest layer module instances are placed on the top of the stack panel while the lowest layer module instances are placed in the bottom of the stack panel. Module instances are placed from left to right according to its creation time; oldest on the left to newest at the right. Moving the mouse pointer over a module instance triggers a tooltip that shows details of that particular module, highlights all of its relationships. Clicking on the module instance creates a popup that displays all detail information about that module instance.

6.3 Control Panel

The control panel consists of two tabs:

6.3.1 *Customize Tab*

Users can open the popup to see information in each module instance. However, there should be a way to show some part of the information in all module instances at the same time. Visty allows users to show any two specified data fields on the box using the custom label controls. (See Figure 6.) Moreover, a number alone does not help the users see the trends or spot anomalies. Therefore, we implement a customizable color-coding mechanism into Visty. Users can select up to two fields that they want to color-code the boxes by. The boxes are divided into two parts; the upper part has color corresponding to color code 1 (white to red) while the lower part has color corresponding to color code 2 (white to green). The darker color represents higher value.

6.3.2 *Filter Tab*

Visty allows users to filter out irrelevant data from the network stack. In control panel, there are checkboxes labeled by module instance names, which are grouped by layer. Users can show or hide the module instances in the network stack by checking or unchecking these checkboxes. (See Figure 7.)

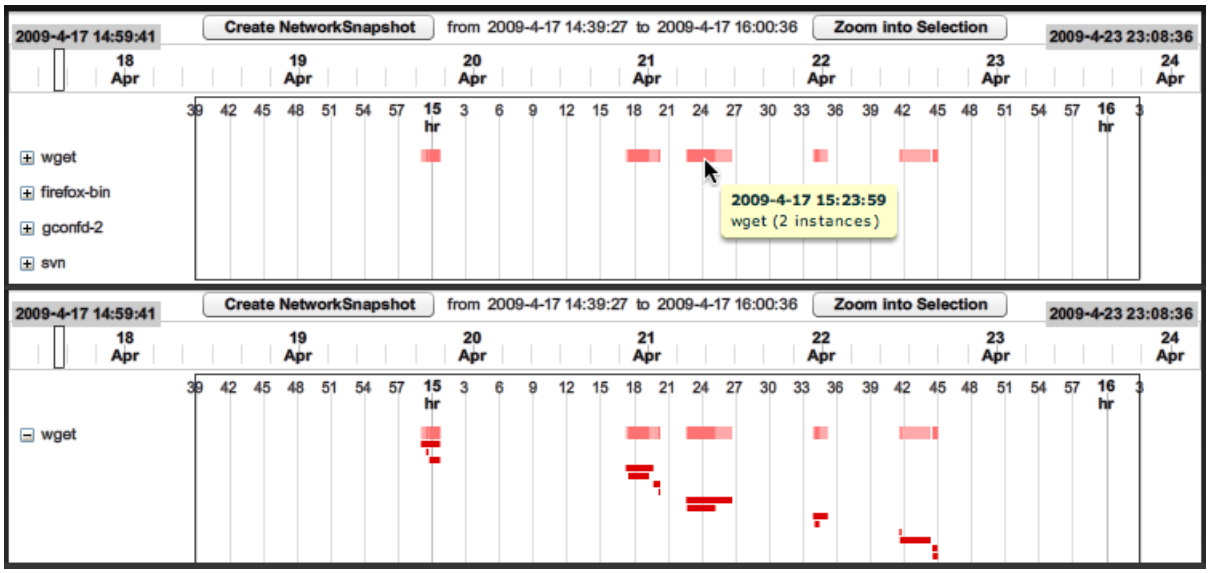

Figure 4: Timeline Panel - (above) This dataset was collected while four applications were running: wget, firefox, gconfd, svn. (below) Users can expand to see all instances of each application.

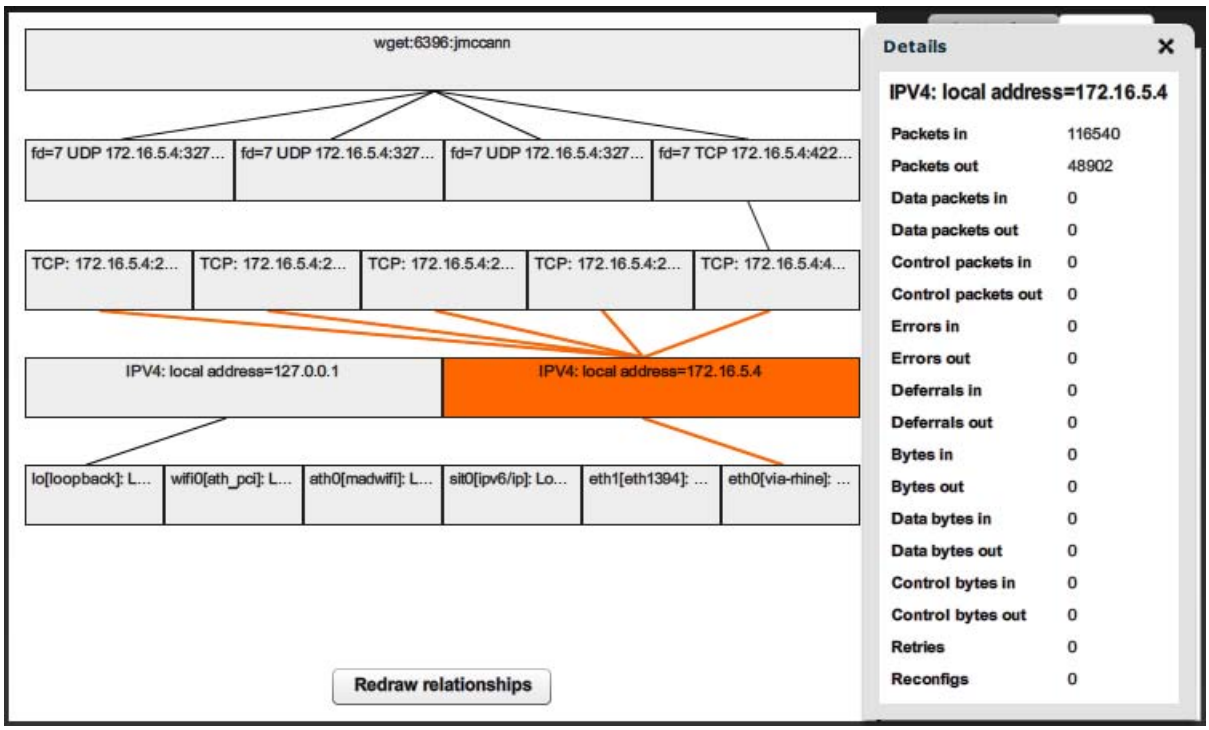

Figure 5: Stack Panel - The network stack is shown on the left. Placing a cursor over a box highlights it and all of its relationships. The user clicks on a box the bring up the popup that shows more details.

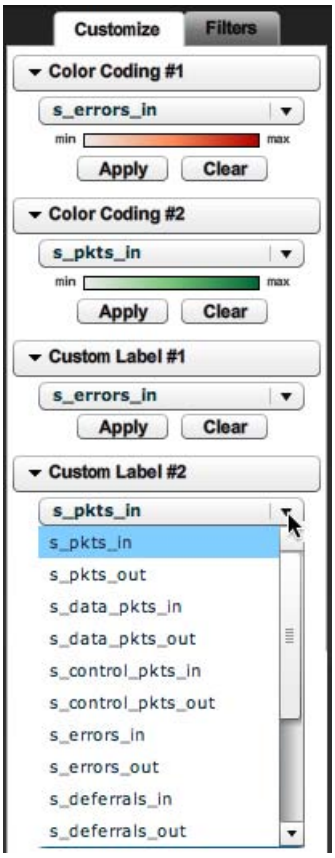

Figure 6: Control Panel - Customize Tab: Visty allows users to select up to two fields that they want to use to color-code the boxes. Also, the users can choose to display any two specified data fields on the box by selecting the field names from the "custom label" drop-down lists.

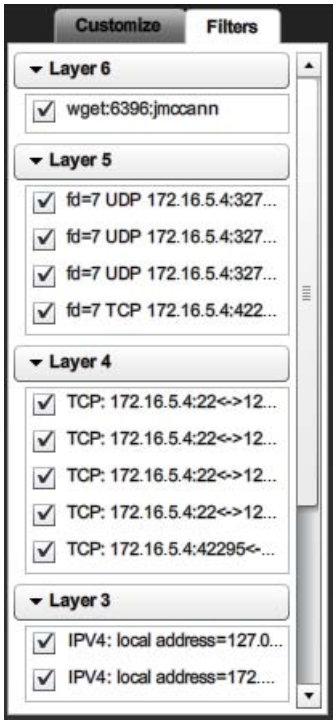

Figure 7: Control Panel - Filter Tab: Users can show or hide the module instances in the network stack by checking or unchecking these checkboxes. The module instances are grouped by layers.

7. RESULTS

7.1 Use Case Examples

This section describes two example scenarios when Visty may be used.

7.1.1 IT Support

Steve, a typical user, is running an alpha version of an instant messaging application. He runs into network problems while using the software, and is not sure whether it is caused by the bugs in this software, a network connection problem, or something else. To find out what really happened, Steve contacts the instant messaging support team to send someone over. Jane, a technician, visits Steve and starts the data collection tool. She then asks Steve to use the instant messaging software the same way as he did before until the same problems occur again. Jane stops the data collection tool and loads the collected data into Visty. She specifies the time when the problem occurred from the timeline panel on the top and creates network snapshot for that period. Visty generates and displays the network stack for that period of time. Jane filters out irrelevant module instances on the generated network stack so using the controls on the "filters" tab on the right. She then sets the color coding scheme to help her locate module instances with high error or retransmission rates. There is a red spot in several of the TCP socket module instances. From the stack visualization, Jane notices that these TCP sockets were created by several different applications, and the instant messaging software appears to be waiting for TCP to transmit outgo-

ing segments. She also sees that the Ethernet interface is displaying errors, and suspects the messaging application is not at fault. Further diagnosis reveals that auto-negotiation between the host and Ethernet switch had malfunctioned, resulting in mismatched Ethernet duplex settings.

7.1.2 Learning

Wayne is an instructor of Introduction to Computer Networks course. After going over the slides and explaining the details of each network layer, he wants to show the real example of how these layers are tied together to the students in addition to illustrated graphics from the textbooks, so he starts the data collection tool and opens his web browser and Twitter client. He visits the department website and tweets a few short messages into Twitter, then stops the data collection tool and opens Visty. He specifies the time range of the data that was just collected and creates network snapshot. The students now can see the network stack of Wayne's recent activities. He can use the visualization to explain more about how data flows from his web browser through the layers to the web server, and vice versa.

7.2 Analysis of Sample Data

This sample data was a modification of collected data on one machine from Apr 17, 2009 to Apr 23, 2009. The analysts loaded data into Visty and saw from the timeline that the user was running *wget* from 12:50 to 15:15, approximately, on Apr 23. They wanted to see what was happening during that period so they specified that range and created a network snapshot. (See Figure 2.) From the links representing dependencies between modules, they found that wget created four sockets: three services were in connectionless mode using ports number 32777 and one service was in connection-oriented mode using port number 42285. With the connection-oriented mode, the wget application established a connection to the other end by requesting a service from the TCP/IP protocol. A connection was created between host IP address 172.16.5.4 on port number 42295 and the server IP address 194.71.11.69 on port number 80. By using IPv4 (i.e., Internet Protocol version 4), packets were sent to the eth0 module, an Ethernet card, which was connected to an ISP.

The color coding method was used to locate module instances with high errors. They used green to represent high numbers of incoming packets and red to represent high number of incoming errors for each module instance. A green spot in connection-oriented socket showed the highest usage of this module among all four sockets, while a red spot in the wifi0 module pointed out high numbers of incoming errors, indicating some problems which this interface. They discussed with the data collector who explained that this network card was not used during that time, but not disabled. This concluded that the high number of incoming errors was caused by wireless sensing from this device. There was also another red spot in the IP layer which suggested a possible problem. This could be caused by the fault in IP layer on the machine or from the medium. They made an assumption that IP layer on that machine was working properly and therefore looked for other possible problems from the source. They discussed with the data collector again and finally found that the router malfunctioned and corrupted a large number of packets.

8. FUTURE WORK

The timeline can still be improved by showing not only the application layer information, but also information from lower layers.

The network stack visualization could be improved by making better use of the available space. We tried to use the width of the boxes to encode another dimension of the data, e.g. number of incoming or outgoing links, however, we found that the labels for instances which had low number of links could not be displayed because the boxes were too small. An appropriate trade-off decision has to be made. Scalability is another important issue. One interesting problem would be how to display the module instances when the number of instances is very large, e.g. for servers with large numbers of simultaneous connections. Also, the current design supports only data that follows the OSI model strictly. However, real networks can be more complex, with tunneling, VPN, proxies, etc. which can be more difficult to visualize.

More features related to automatic performance diagnosis can be added to assist analysis. The tool may provide suggestions of possible problems in the data and allows the users to examine the details.

For educational purposes, it may be useful to be able to have a video playback feature to show how the network stack was constructed and changed over time. This will allow the students to see changes and understand the data flow step-by-step.

9. CONCLUSION

Communication across the network is complicated. Communication functions are grouped into related and manageable layers, but each layer hides information such as loss and retransmissions from layers that depend upon it. Even if an expert understands how each layer of network architecture is actually related, locating errors and bottlenecks caused by these layers is still a challenging task. Combining with the fact that the amount of data which the network data collection tool can generate is usually large, this kind of analysis can take an unreasonable amount of time and effort.

In this paper, we describe a network stack visualization tool called *Visty* that assists in cross-layer network performance analysis. Visty provides users with an overview of the network stack over time, allows them to systematically explore the network stack within any specified period of time, and to use their knowledge to locate errors or bottlenecks hidden shown by the data. Visty is designed to be simple but can be enhanced into a more advanced diagnosis tool for advanced users. Moreover, Visty may be used as an educational tool to help students understand the network stack and relationships between each layer.

10. ACKNOWLEDGEMENT

The authors would like to thank Dr. Neil Spring for his guidance and all the reviewers for their thoughtful comments.

11. REFERENCES

[1] assql. *http://code.google.com/p/assql/* (Jul 2009).
[2] Big brother. *http://bb4.com* (Sep 2009).
[3] Cricket. *http://cricket.sourceforge.net* (Sep 2009).

[4] Munin. *http://munin.projects.linpro.no* (Sep 2009).

[5] Adya, A., Bahl, P., Chandra, R., and Qiu, L. Architecture and techniques for diagnosing faults in ieee 802.11 infrastructure networks. In *MobiCom '04: Proc. 10th Annual Int. Conf. Mobile computing and networking* (New York, NY, USA, 2004), ACM, pp. 30–44.

[6] Bahl, P., Padhye, J., Ravindranath, L., Singh, M., Wolman, A., and Zill, B. DAIR: A framework for managing enterprise wireless networks using desktop infrastructure. In *Proc. Annual ACM Workshop on Hot Topics in Networks (HotNets)* (2005).

[7] Ball, R., Fink, G. A., and North, C. Home-centric visualization of network traffic for security administration. In *VizSEC/DMSEC '04: Proc. ACM workshop on Visualization and data mining for computer security* (New York, NY, USA, 2004), ACM, pp. 55–64.

[8] Barth, W. *Nagios: System and network monitoring*. No Starch Press San Francisco, CA, USA, 2006.

[9] Becker, R., Eick, S., and Wilks, A. Visualizing network data. *IEEE Trans. Visualization and Computer Graphics 1*, 1 (1995), 16–28.

[10] Blue, R., Dunne, C., Fuchs, A., King, K., and Schulman, A. Visualizing Real-Time Network Resource Usage. In *Proc. Visualization for Computer Security: 5th Int. Workshop, Vizsec 2008* (Cambridge, MA, USA, September 2008), Springer, p. 119.

[11] Chandra, R., Padmanabhan, V., and Zhang, M. WiFiProfiler: cooperative diagnosis in wireless LANs. In *Proc. 4th Int. Conf. Mobile systems, applications and services* (2006), ACM.

[12] Cheng, Y.-C., Afanasyev, M., Verkaik, P., Benkö, P., Chiang, J., Snoeren, A. C., Savage, S., and Voelker, G. M. Automating cross-layer diagnosis of enterprise wireless networks. *SIGCOMM Comput. Commun. Rev. 37*, 4 (2007), 25–36.

[13] Cheng, Y.-C., Bellardo, J., Benkö, P., Snoeren, A. C., Voelker, G. M., and Savage, S. Jigsaw: solving the puzzle of enterprise 802.11 analysis. *SIGCOMM Comput. Commun. Rev. 36*, 4 (2006), 39–50.

[14] Cheswick, B., Burch, H., and Branigan, S. Mapping and visualizing the internet. In *USENIX Ann. Tech. Conf.* (2000), pp. 1–12.

[15] Combs, G., et al. Wireshark. *http://www.wireshark.org* (July 2009).

[16] Dunn, J., Neufeld, M., Sheth, A., Grunwald, D., and Bennett, J. A practical cross-layer mechanism for fairness in 802.11 networks. *Mobile Networks and Applications 11*, 1 (2006), 37–45.

[17] Jardosh, A. P., Ramachandran, K. N., Almeroth, K. C., and Belding-Royer, E. M. Understanding link-layer behavior in highly congested ieee 802.11b wireless networks. In *E-WIND '05: Proc. ACM SIGCOMM workshop on Experimental approaches to wireless network design and analysis* (New York, NY, USA, 2005), ACM, pp. 11–16.

[18] Kim, S., and Reddy, A. Netviewer: a network traffic visualization and analysis tool. In *Proc. 19th Conf. Large Installation System Administration* (2005), vol. 19, USENIX Association Berkeley, CA, USA, pp. 18–18.

[19] McPherson, J., Ma, K.-L., Krystosk, P., Bartoletti, T., and Christensen, M. Portvis: a tool for port-based detection of security events. In *VizSEC/DMSEC '04: Proc. ACM workshop on Visualization and data mining for computer security* (New York, NY, USA, 2004), ACM, pp. 73–81.

[20] Mishra, A., Shin, M., and Arbaugh, W. An empirical analysis of the IEEE 802. 11 MAC layer handoff process. *ACM SIGCOMM Computer Communication Review 33*, 2 (2003), 93–102.

[21] Orebaugh, A., Ramirez, G., and Burke, J. *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress Publishing, 2007.

[22] Patwari, N., Hero, III, A. O., and Pacholski, A. Manifold learning visualization of network traffic data. In *MineNet '05: Proc. ACM SIGCOMM workshop on Mining network data* (New York, NY, USA, 2005), ACM, pp. 191–196.

[23] Shneiderman, B. The eyes have it: a task by data type taxonomy for information visualizations. In *Proc. IEEE Symp. Visual Languages* (Sep 1996), pp. 336–343.

[24] Tehsin, S., Khan, S., and Khattak, N. Visualizing Network Traffic as Images for Network Anomaly Detection. In *International Conference on Informatics* (2007).

[25] Wagner, I., Carlsson, G., Ekstrand, K., Odman, P., and Schneider, N. A comparative study of assessment of dental appearance by dentists, dental technicians, and laymen using computer-aided image manipulation. *Journal of Esthetic and Restorative Dentistry 8*, 5 (1996), 199–205.

[26] Yang, H., Shu, J., Meng, X., and Lu, S. SCAN: self-organized network-layer security in mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications 24*, 2 (2006), 261–273.

[27] Zhang, M. Understanding internet routing anomalies and building robust transport layer protocols. Tech. rep., Princeton University, 2005.