PennAnalytics: Network Visualization and Analytics

Dept. of CIS - Senior Design 2013-2014

Patrick Wingo
pbwingo@gmail.com
Univ. of Pennsylvania

Aubrey Chase
aubchase@seas.upenn.edu
Univ. of Pennsylvania

Bill He
billshihe@gmail.com
Univ. of Pennsylvania

Daniel Ge
dange@seas.upenn.edu
Univ. of Pennsylvania

Ryan Sasson
rsasson@seas.upenn.edu
Univ. of Pennsylvania

1. ABSTRACT

Society is becoming increasingly digitized and reliant on computer networks for everything from business to leisure. Due to this reliance, it has become increasingly important for computer networks to be reachable around the clock. Meanwhile, computer networks are growing in size and complexity. With the importance, ubiquity, and growth of computer networks, issues such as congestion are becoming more prevalent and increasingly challenging to address for network administrators who are responsible for managing computer networks. As a consequence, the need for easy-to-use tools to analyze computer networks is more urgent than ever. Many current network analytics tools present data in a way that is hard for humans to understand and use effectively. PennAnalytics is a tool designed to alleviate this situation and help IT professionals better understand the condition of a computer network. It uses Simple Network Management Protocol (SNMP) to visualize and analyze computer network data in real time and display information in a user-friendly way through a web browser.

PennAnalytics differentiates itself in three dimensions: real-time analysis, cross-platform compatibility through web browsers, and a focus on user experience. Instead of manually building out a network to visualize connections between nodes, PennAnalytics automatically builds the network graph and displays traffic over time. It was designed as a tool to allow administrators to quickly understand problems at a macro level, as well as easily drill down and display potential problems at the micro level to expedite and enhance the manual network diagnostics process. In order to do this, we built the entire stack that supports a tool that visualizes the network and network metrics in a way that provides clarity on network problems such as traffic spikes and network congestion.

2. GLOSSARY

Link Layer Discovery Protocol (LLDP) is a protocol used to discover the layout of a network [2]. This protocol allows Ethernet devices such as switches and routers to broadcast information about themselves, e.g. capabilities and neighbors, as well as receive this information from other devices. Devices broadcast information about themselves and store information they learn about each other in local databases called management information bases (described later in this section), allowing network management systems access this data to build a map of the network topology.

Simple Network Management Protocol (SNMP) is a protocol that helps exchange information about a network between the devices in the network [4]. The basic setup of an SNMP-managed network consists of managed devices, agents, and network-management systems. Managed devices run SNMP agents (software) and store information which can be accessed via SNMP by the network-management system through a query. It is important to note that SNMP does not define the information that is stored on managed devices and accessed by the network-management system. Defining the management information is the job of a database known as the management information base (MIB).

3. INTRODUCTION

The modern Internet is becoming increasingly complicated, and as we become more reliant on continuous connectivity, we face a greater risk and danger stemming from both malicious and accidental outages. Akamai Technologies reported a 54% quarter-on-quarter increase in DDoS attacks during 2013, and the number of attacks only continues to rise over time [8]. Malicious outages are not the only source of network failures: our consumption of multimedia and transmission of huge amounts of data takes its toll on the network, and, as a result, congestion can be a significant problem. Given the massive economic implications of network failure, it is important to detect and diagnose problems even before they begin. We aim to contribute to this field and add to the corpus of tools currently on the market.

Many IT professionals must perform analysis of a computer network to discover information such as the topology of the network, the capacity of network links, and the capacity utilization of network links. The usage of SNMP to retrieve data has emerged as a building block to perform this analysis and tools such as NSen and other proprietary network analysis tools have used this building block to enable analytics through a variety of configurations and interfaces.

While there are some good products and open-source technologies currently available, we believe that existing tools can be further improved upon. Log files give extremely granular detail that provides rich information, but can be overwhelming for a user. Charts give a high level picture of network statistics, but sacrifice details to achieve that simplicity. Existing graph solutions offer a logical and intu-
itive way of viewing a network, but fail to incorporate both high level visualization and low level detail elegantly into one workspace.

Our goal was to create a network visualization tool that addresses these problems. Our tool combines log, chart, and graph visualization methods to simultaneously offer users a high-level picture of the network and link-level detail in one workspace. To further enhance the analytic capabilities of users, we detect traffic spikes and high network usage and perform this analysis in real-time. Furthermore, PennAnalytics is built on top of SNMP, which is a protocol that is supported by many network device vendors, can be hosted on any server, and is accessed via a web browser. All of this means that PennAnalytics can be run in a variety of IT environments.

4. RELATED WORK

Many pieces of software have been written to improve the utilization of NetFlow for network analysis and visualization, and there are a variety of open- and closed-source tools available. The same applies for SNMP and LLDP tools. In fact, LLDP is specifically designed to provide information on the topology of a network, the very thing that we want to build out, but we find that the tools are widely disaggregated, and often unintuitive to use. For example, LLDP is a command line based tool: the results output is completely text-based, and it is difficult to understand the output even after reading manuals.

One of the better software packages available is SolarWinds, a commercial NetFlow analysis package that breaks down network nodes in tree form, on a map, by CPU usage, node health, and other metrics [1]. We find that it is a great way to diagnose potential problems with the network, but based on our test of the free trial, it is not an effective tool for isolating a problem. The user interface is also somewhat confusing: the relationship between different metrics and graphs is not highlighted. In addition, it appears that LLDP and SNMP are not incorporated into the SolarWinds software; we believe that by integrating multiple data sets, we can provide a more holistic picture of network health.

Another related tool is NfSen, an open-source web-based NetFlow aggregator and analyzer [3]. It takes raw NetFlow data from another open-source tool, NfHump, and visualizes it onto a web interface. It solves the problem of cross-platform visualization, but the graphs are too high-level to be useful for anything other than establishing the existence of a basic problem. We spoke with the Penn system administrators, and they found NfSen to be useful only in detecting potential anomalies. From there, they used command tools and frequently needed a multitude of other software platforms in order to understand the problem in detail.

Work by Minarik and Dymacek [7] demonstrates the need for system administrators to be able to intuitively recognize a problem using a graph-based, high-level examination, and then to be able to drill down on those nodes to understand statistics for those nodes as part of their investigation. They include programmatic DNS, port, and WHOIS lookups in order to reduce the mechanical work of an analyst, but we found that they do not use LLDP or SNMP as a means to create and understand topology. Their user interface also could be improved to make it more intuitive.

Kotz and Essien [6] used SNMP in order to understand macro-scale campus networks as wireless was emerging in 2005. Although the research primarily investigated roaming across the network, which is not entirely relevant to our system, their method of breaking down traffic across the network using SNMP is relevant. The researchers polled their access points every 5 minutes using the SNMP software installed across the network, and got inbound and outbound byte counts as well as recent MAC addresses. From there, they were able to graph traffic over the course of an average day as well as traffic patterns over an extended period of time.

Yin et al. [9] developed a tool called VisFlowConnect, a visualization tool that uses a parallel axes representation of NetFlow to help users envisage network traffic. The development of VisFlowConnect was motivated by the insufficiency of traditional tools to present hostile attack patterns in an intuitive way for the human mind. VisFlowConnect has various features that have served as inspiration for our system, including interface views of different granularity (e.g. global, intra-network) and filtering of data on various parameters (e.g. protocol, packet-size). However, a shortcoming of VisFlowConnect is the inability to drill down on the flow data and not only filter on, but observe information such as protocol and transfer rate. In addition, a network can be represented as parallel axes (as it is in VisFlowConnect), but we wanted to experiment with an alternative graph representation.

Ball et al. [5] developed a tool called VISUAL Information Security Utility for Administration Live (VISUAL) which allows users to see communication patterns among internal networks and external hosts in a graphical representation. After surveying IT professionals, Ball, Fink, and North found that users found it difficult to quickly assess the security state of networks with text-based tools. They found that with VISUAL, users could develop insights from the traffic data without any training, demonstrating the power of visualization tools. VISUAL’s main features include markers for external and internal hosts, color-coded links between external and internal hosts, interactive filtering, and time lines. We attempted to improve on this by including more proactive displays and processing network data closer to real-time.

5. SYSTEM MODEL

Our goal was to create a system that aggregates, visualizes, and analyzes network data to extrapolate meaningful results. We aimed to display the network data in a simultaneously user friendly and descriptive way. The architecture of the application can be broken down into two main components as shown in Figure 1. First, we have a module that aggregates, parses, and processes network data. Then, the processed network data is input into a user-friendly, interactive front end on a web platform. We were able to make our system process a constant stream of data from an experimental network, making the system real-time.

To retrieve the data from the routers, our system first has a server attached to one of the network’s main switches, periodically sending SNMP queries to each switch in the network, with this process shown in Figure 2. This lets the system obtain both network statistics and LLDP data, allowing us to infer the network’s load and topology. The output of these queries is stored in memory. The application is then able to perform a series of computations in order to reduce the data down to the minimum needed for display to the client. When a network administrator properly logs in and
requests to view the system, her browser will send a request to the server, which then dynamically fetches the necessary data and statistics. The client then renders the statistics and visualizations using a standard graphing library, while asynchronously checking for updates.

The rationale behind this architecture is that the front end does not need to be exposed to back end implementation details (e.g. SNMP queries to each switch) and vice versa. This enables a clean line of development and lends itself towards natural interfaces between each module. In addition, future modifications of the system could add new sources of data to the analysis, such as SNMP requests for additional data fields, without requiring the front end to parse different data formats. This separation of concerns allowed us to continue moving forward without being hindered by forward changes in other components of the system. Thus, this architecture made it possible for us to specialize in implementing different components, therefore allowing us to accomplish more in our implementation of PennAnalytics.

The back end is implemented in Python, and thus can run on any computer or server with Python support. For our setup, we chose to use a Raspberry Pi, a small, inexpensive low-power computer, that we connected via Ethernet to one of the switches in our switch setup.

We chose to use Python because the programming language is particularly well-suited for text processing and string manipulation. Although Python is not the most efficient language, we found that not only was it much easier to make modifications to in the initial prototyping stages, but also that the majority of execution time is spent on network I/O.

We divided the back end into two components: a daemon-like program and a server program. The daemon is a back-ground process that is responsible for polling the switches at a fixed interval. It sends a series of SNMP requests to each switch that it is configured to monitor, processes the incoming data, and outputs the data into a JavaScript Object Notation (JSON) file. JSON was an ideal format because it is not only human-readable, but also widely supported in many languages, making the API interface (described later) cross-platform.

The background Python script delegates the task of sending SNMP requests to the command line tools `snmpget` and `snmpwalk`, both of which are part of the Net-SNMP suite of tools. These tools return SNMP data in the form of key-value pairs, in which each key is a series of numbers (e.g. `1.0.8802.1.1.2.1.3.3.0`) whose definitions are usually standardized (`lldpLocSysName` for the previous example) or are proprietary. The outputs of these tools are parsed and organized into data structures that represent the nodes in a network.

Since SNMP responses tend to contain a lot of data, a majority of execution time is spent waiting on network I/O. To mitigate this issue, the daemon creates a thread for each switch. Each thread sends SNMP requests to and reads responses from only one switch. All threads synchronize when modifying the common data structure storing information about each switch. Thus, we were able to parallelize all SNMP traffic, mitigating the issue of network latency.

The server component is implemented with Flask, a lightweight Python web framework. It is responsible for providing the API that allows clients to retrieve switch data in JSON. It runs independently of the daemon process, allowing us to serve multiple requests for data simultaneously without requiring additional SNMP traffic. This application server is run behind the popular reverse proxy server nginx, which forwards requests to Flask.

Each switch stores LLDP data and link statistics in its MIB, which can be queried by issuing SNMP GET requests for those respective fields. Data generated from LLDP, a protocol over which switches advertise themselves to their neighbors, allows us to construct the topology of the network.

On the other hand, link statistics, such as the capacity and number of bytes sent and received for each link, can be used to infer metrics such as link utilization.

To represent and perform analysis on the information retrieved from SNMP queries, we created data structures to represent network nodes and network links. Network nodes are dictionaries whose keys are port numbers and values are network link data structures. Each network node also contains the name of the node and its physical address. Within

---

**Figure 1: Basic architecture of the project**

**Figure 2: SNMP Data Pipeline**

### 6. SYSTEM IMPLEMENTATION

#### 6.1 Back End
each node, one can get the total capacity of all adjacent edges, total bytes received, total bytes sent, total bytes received since last SNMP query, and total bytes sent since last SNMP query.

Network links contain information about the system name of the node on the other side of the link, the capacity of the link (in bytes per second), total bytes received, total bytes sent, total bytes received since last SNMP query, total bytes sent since last SNMP query, total number of SNMP queries, a moving average of input utilization in every SNMP query interval, and a moving average of output utilization in every SNMP query interval. To calculate bytes received/sent since last SNMP query, each link maintains queues to keep track of bytes received/sent for the current and previous SNMP queries. To calculate input utilization since the last SNMP query, we calculate

\[
\text{Bytes Received Since Last SNMP Query} \times \frac{8}{\text{Seconds Between Each SNMP Query} \times \text{Link Capacity}}
\]

Similarly, to calculate output utilization since the last SNMP query, we calculate

\[
\text{Bytes Sent Since Last SNMP Query} \times \frac{8}{\text{Seconds Between Each SNMP Query} \times \text{Link Capacity}}
\]

At each SNMP query, we update the moving average of input utilization as follows

\[
\text{Input Utilization Average} = \frac{\text{Last Input Utilization} - \text{Input Utilization Average}}{\text{Total Number of SNMP Queries}} + 1
\]

Similarly, we update output utilization as follows

\[
\text{Output Utilization Average} = \frac{\text{Last Output Utilization} - \text{Output Utilization Average}}{\text{Total Number of SNMP Queries}} + 1
\]

### 6.2 Front End

With a server that now stores all the relevant router node information in memory in a structured format, a Node.js server running within the Penn network will serve this information to the client. The server will send data as well as HTML, CSS, and Javascript resources to the client; node and link state will be updated in a continuous fashion through a socket connection between a client and a server. This allows the server to push updates to the browser client instantaneously. The benefit of this is that a network administrator could leave the dashboard open on a large monitor and not have to continuously refresh it. The server will retrieve information by polling the network server periodically and will push that data back to the client to maintain a realtime stream of data. The client will store this data within an in memory data structure until the data structure reaches a maximum size of elements. At this point the data structure will evict the oldest elements to make room for the new incoming data points.

The other role that the Node.js server serves is that it is another network data transforming layer. In order to render the data visualizations in the client, the server must further process the data in order to fit the data format required by the d3.js data visualization library. d3.js requires the network graph represented as a list of all nodes and a list of all adjacencies between them. Additionally the time series graphs per link in the network (one for bytes in/out, one for download utilization and upload utilization) require data points to have a timestamp, a value, and an ID corresponding to the link the data points correspond to. The server will also run the calculation per node to determine the load on the router node through approximating the sum of all data going in from the links going into the node. This data will be used to determine the health of the node. The server will run the necessary transformations and calculations on the data so the client has clean and useful data.

By this point we have all the network data parsed into a clean JSON format and the server now serves this data in real time. The next step is to take the JSON data and to transform it into intuitive visuals that give the user insight into the ongoings of the network. As mentioned earlier, the tool that enables the application to display visualizations of the data is d3.js, a Javascript framework with a large set of built in visualizations such as graphs or charts. With this framework we utilize two key visualizations: a directed graph and time series charts.

The directed graph is a visual tied to the overall topology of the router network the router server is querying. Given all of the adjacencies between router nodes, we render a graph based on the information to provide the global view of all of the nodes. The user can see the names of the nodes and links connected by hovering the mouse over the node or link in question. If a particular node sees an increasing load, a corresponding color transition will happen. If a node is under light load then it will have a green color indicating it is working fine. Every progressing increase in load will correspond to a color change from yellow, orange, then to red, which indicates a node is under heavy load. This lets the user view the health of the network in one glance.

![Figure 3: Topology information retrieved from a three-node setup](image-url)
information about what is going on within it (Figure 3). Upon clicking a link, the front end renders two time series: one of packet traffic in and packet traffic out, and one of download utilization percentages and upload utilization percentages. The time series types are superimposed on each other in order to let the user do a direct comparison (Figure 4). This provides a granular view of the node if the user requires more specific data on the link. The user can also shift the time scale of the time series by dragging the ends of a bar underneath the time series. This allows the user to get an overall view of the time series and to get a granular view of the time series.

Figure 4: Time series chart for a single link

7. RESULTS AND SYSTEM PERFORMANCE

Given that our project is to visualize network traffic in a way that allows humans to better interpret and analyze conditions, it is difficult to provide statistics on system performance. We measure our success in three ways: accuracy of our statistics and visualizations, usefulness of the information and display, as well as internal code quality.

Both the low-level and high-level parts of the stack work successfully and accurately in describing the structure of our analyzed network. The front end takes parsed data from the SNMP queries and displays the network topology. The topology we have extracted is the same topology as our setup and changes based on differing router connections. We have tested this by comparing our human-readable data format to the known network topology and verifying the accuracy of our initial outputs. In addition, our visualization topology is interactive; a user can browse through the network and see which nodes are connected to each other and check their names. These names and topology are exactly as described in our human-readable format, implying accuracy along the entire pipeline for our graph. One caveat to this performance is the fact that we are missing certain switches within the network; they simply do not appear when we parse through the data. Upon investigation, we have found that not all switches in the network run the necessary software to perform LLDP, and that this causes some of them to drop from the graph. However, LLDP is very common, and it is possible to install the tool on routers and switches, so that a complete network will form. We tested the majority of our system using spare routers given to us by CETS and we are able to extract all of the necessary information from them.

In terms of utility of our tool, it is able to detect network activity in real time. It also alerts users to the presence of congestion, and abnormal levels of traffic. It is interesting to look at and users can intuitively understand the topology of the network. We were able to meet with CETS administrators in order to test our software and see if it was of use to them. Their feedback was largely positive. They enjoyed the interface and stated that it would be a useful interface for monitoring Penn’s network.

Our code so far is entirely concentrated in one directory, and it is subdivided into different folders that separate it into distinctive parts. Because of the large amount of highly stratified work, and the architecture we planned since the beginning, we have been able to perform separation of concerns. This means that our low-level modules perform independently of our server and front-end. Only necessary data is transmitted between different modules, and we believe that this helps our development process. Although we have a varied codebase with many different languages and technologies, our codebase is quite intuitive to understand, and the internal quality of the project is quite high.

8. FUTURE WORK

We can easily think of three useful additions that could be made to PennAnalytics. Firstly, much of our work was only on the small network of switches given to us by Penn CETS. We only received single timestamp, static dumps of their network to analyze with the system. This was not enough to test the almost real time calculations we wished to maintain. We were able to see that our system left only a small footprint on the system, and that we could gather the large amount of data necessary, analyze it and relay it to our front end, but a larger network might require a different method for data analysis. If necessary, it would be simple to parallelize the data pipeline using a tool such as hadoop or twitter storm.

By implementing machine learning algorithms, we could add anomaly detection to the system. This would allow the tool to be truly proactive notifying network administrators of potential attacks or problems before they crippled the network. If network traffic increased too suddenly, appeared abnormal when compared to a normal routine, the user would be able to tell. The more information provided, and the larger the history of data stored, the better these detection algorithms would be able to do. More of the abilities of anomaly detection are explained below.

Netflow would also be a great addition to the tool and would help expand upon its uses. IP sources and destinations would allow for a more complete map of a network down to individual machines. It would also allow for tools like a “packet map” showing where traffic is being sent and where it being received from. This would be helpful in de-
terminating the source of large traffic users or even attacks on
the network. By adding port information on top of this, we
would hopefully be able to imply the service in use as well.
We might be able to tell whether a packet was, for example,
a Skype call, and with the addition of filters on this data,
possibly where the call was being made. This is only a small
sampling of what NetFlow could add.

NetFlow would also be the most questionable addition
due to security concerns. Even though it consists of sim-
ple packet metadata, it enables network tools to infer much
about a computers activity. This raises questions about user
privacy, and is why Penn CETS would not allow us to use
any of their network’s NetFlow data. Based on locations and
habits, owners of a stated IP address could be identified
and monitored. As mentioned above, we could see the des-
tination of aperson’s Skype call destination was. And that
is one of the most naïve abilities of the data.

Obviously a combination of all three of these additions
would be optimal. A system capable of grand scale calcu-
lations over a large array of packet and transfer data would
allow for the best display of network health. With anomaly
detection layered on top of this, the system would be able
to detect any number of anomalies from DDoS to simple
increases in requests to Facebook. Using packet metadata,
anomaly detection algorithms would be able to determine
fluctuations in regional traffic to and from their local net-
work as well as countering large influxes of packets from a
single address. All this would be possible while still keeping
the simplified and intuitive interface operational.

9. ETHICS

Ethics are a significant concern for the PennAnalytics tool
due to the power of network visualization in understanding
network problems and human activities. Because the Penn-
Analytics project requires full network access, and is able to
understand patterns of traffic over time, there is potential
for ethical concerns both at the level of the individual and
at the level of the network. Because administrators will be
able to see traffic from routers over time, they will be able
to record and understand all traffic from a given router. If
an unethical voyeur is able to derive which users tend to use
a particular router, they can make inferences on user activity
based on traffic patterns over time. This is a significant
concern, but it is partially mitigated through the aggrega-
tion of individual-level data on routers. Very few routers
service only one user, so it would be difficult to understand
which user has a particular traffic pattern. One strategy to
address and further alleviate this ethical and privacy issue
is to add a given amount of noise to the data in order to
obscure traffic patterns.

At the network level, there are also significant privacy
concerns. Even displaying aggregated network data is a se-
curity risk, as attackers with full knowledge of aggregated
network traffic will better understand positions of weak-
ness within a network. Disgruntled employees, or attackers
who manage to break in and spend time working with the
dashboard will be able to understand traffic patterns over
time across different network nodes and links. This knowl-
edge would allow such an attacker to understand weak links
within the network; in order to flood the networks with traf-
cic and shut them down through an intelligence-augmented
denial-of-service attack. Given that the PennAnalytics tool
was created to help network administrators to maintain the
health of a network through better network intelligence, this
is a significant concern. The best way to protect against this
sort of ethical breach is to securely maintain access to the
network analysis dashboard and application programming
interface, and to firewall off any external access to them.

In the production system of a large, highly sensitive net-
work, it would be important to ensure the security of the
PennAnalytics tool. Furthermore, it would be important to
stress test the software to ensure that it never malfunctions
and overburdens the network with query requests, as gather-
ing network topology and traffic data requires some queries
on the network. If these queries overburden the network
traffic and cause failure of an important network, this would
be a serious problem for PennAnalytics, and would violate
the ethical concern of doing no harm. The primary meth-
ods to avoid such a failure include significant testing within
production systems for unknown software bugs, as well as
a fail-safe system that cuts back on the process of querying
network traffic when the traffic burden is already significant.

10. CONCLUSION

The increasing importance of computer networks had led
to many approaches in managing networks. Removing the
human part of the equation and automating network diag-
nostics is one approach. However, machine learning tech-
niques are not infallible in real world situations and may fail
in tasks that are intuitive for humans. Thus, we believe that
human judgement is still essential to network diagnostics.

To improve the capabilities of a human that is managing a
network and help him or her effectively diagnose a network, a
tool that allows them to look at a computer network at both
a macro and micro level through an intuitive user interface is
invaluable. While there are several existing tools that enable
network visualization and analytics, few are able to do it
real-time, deliver the service in a platform agnostic way, and
through a friendly user interface. PennAnalytics delivers on
all three dimensions and can serve as a foundation on which
network administrators monitor their networks.

11. REFERENCES

[1] It management and monitoring software by solarwinds.
discovering and managing converged network devices.
http://www.extremenetworks.com/libraries/
2013-12-10.
Home-centric visualization of network traffic for
security administration. In Proceedings of the 2004
ACM workshop on Visualization and data mining for
computer security. VizSEC/DMSEC ’04, pages 55–64,
New York, NY, USA, 2004. ACM.
campus-wide wireless network. Wireless Networks,
