Flow-Inspector: A Framework for Visualizing Network Flow Data using Current Web Technologies

Lothar Braun · Mario Volke · Johann Schlamp · Alexander von Bodisco · Georg Carle

November 8, 2012

Abstract New web technologies led to the development of browser applications for data analysis. Modern browser engines allow for building interactive real-time visualization applications that enable efficient ways to understand complex data. We present Flow-Inspector, a highly interactive open-source web framework for visualizing network flow data using latest web technologies.

Flow-Inspector includes a backend for processing and storing large-scale network flow data, as well as a JavaScript-based web application capable to display and manipulate traffic information in real-time. This work provides operators with a toolkit to analyze their networks and enables the scientific community to create new and innovative visualizations of traffic data with an extensible framework. We demonstrate the applicability of our approach by implementing several different visualization components that help to identify topological characteristics in network flows.

1 Introduction

The increasing popularity of web applications has led to numerous W3C standards that specify functionality to build dynamic and interactive web applications. Prominent representatives of such technologies include HTML5 and JavaScript. Those standards provide mechanisms for real-time rendering of 2D graphics and are favored by browser vendors, which leads to advances in rendering speed as browser implementations improve.

Lothar Braun · Mario Volke · Johann Schlamp · Alexander von Bodisco · Georg Carle
Technische Universität München
Department of Computer Science
E-mail: {braun,volke,schlamp,klein,carle}@net.in.tum.de
First IMC Workshop on Internet Visualization (WIV 2012), November 13, 2012, Boston, Massachusetts, USA.

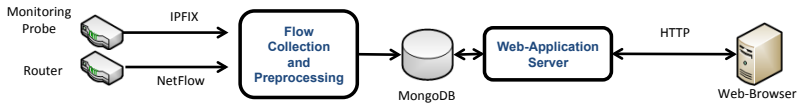

Fig. 1 Flow-Inspector data processing chain

As a result of this technological progress, JavaScript-based frameworks for data visualization emerged [1]. Such frameworks already implement a variety of algorithms useful for the data visualization community, which can be particularly applied to study network measurement data. While such algorithms are capable to display large data sets in a human-understandable way, the use of JavaScript enables highly interactive analyses by providing means to manipulate rendered images. This flexibility can lead to additional insights compared to common static visualization approaches.

We present Flow-Inspector, a JavaScript-based web application that applies modern web technologies to visualize network flow data. The systems consists of a backend for preprocessing, aggregation and storage of data, and a frontend that allows for interactive querying and rendering.

This paper targets two audiences alike: operators and researchers. First, operators benefit from our framework when confronted with analyzing traffic flows in their own networks. Several built-in visualization components are available for different use cases, including volume-based and node-based visualization of traffic.

Flow-Inspector provides users with a new traffic visualization approach: hive plots [2] enable novel analyses of network flow data. The analysis frontend supports drill down methods to filter data and an interface for interacting with rendered images. Additional pieces of information can be provided, e.g. tool tips can be requested by hovering over objects.

Second, researchers and developers can profit by Flow-Inspector's extensible framework. It is straightforward to integrate new visualization algorithms while relying on the backend to provide the necessary data. It is even possible to extend the data model without interfering with other visualization components. This technical flexibility allows for rapid development and early visualization of novel data sets.

We organize our paper as follows: Section 2 introduces the design of our framework, with focus on easy extensibility. With Section 3, we discuss built-in visualization algorithms shipped with Flow-Inspector. We compare our approach to related work in Section 4, and conclude the paper in Section 5.

2 Design and Implementation

Flow-Inspector consists of two main components. A backend that is responsible for preprocessing, aggregating and storing of flow data. While preprocessing the data, it also generates several pre-computed statistics on the data and makes all data available via an HTTP-API.

The second part is a JavaScript application that displays the pre-processed flow information using the methods of current web browsers.

2.1 Data Model

Flow-Inspector's primary goal is to visualize traffic measurements that describe how hosts (or networks) communicate with each other. Relevant characteristics of such communication patterns depend on the environment and the user's intended results, i.e. necessary data is potentially unknown from the design's point of view. Traditional flow information, such as the information transported by NetFlow v5 messages might not be sufficient for certain visualization tasks.

Additional information like the results of QoS measurements, application specific information, or additional structural information could be interesting for a user to visualize. Since we cannot anticipate future use cases and do not want to restrict users, we focus our design on extensibility. We take the definition of a flow given by the IPFIX standard as a base for our data model:

A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties. [3]

The IPFIX flow definition allows for different types of flows and arbitrary supplemental information. Properties that define a flow could be the IP five tuple but are not limited to these keys.

We provide implementational flexibility by allowing the developer to individually specify flows based on a set of arbitrary keys. Supplemental flow properties can be specified as well. Adding new properties does not require any changes to the core code for implementing new visualization components.

2.2 Backend

The previous section outlined the need for a flexible data model that needs to handle potentially unknown data types. Flow-Inspector's backend supports such indetermined data objects with a document-based database called MongoDB [4]. Documents thereby consist of key-value pairs of arbitrary types, and allow to import any flow information from NetFlow or IPFIX messages into the database. Interaction between the frontend application and the database backend is realized using a JSON data model, that allows the frontend to query any types of data as lists of key-value pairs.

The choice of MongoDB was also motivated by some of its built-in functionality: Mongo supports so-called *sharded* databases which allows to distribute the database onto multiple machines. Furthermore, the database includes built-in support for the MapReduce programming model which allows to parallelize complex database requests onto a sharded database. Besides MapReduce, MongoDB also provides a system called "Aggregation Framework", which provides simple ways to perform aggregation on the stored data.

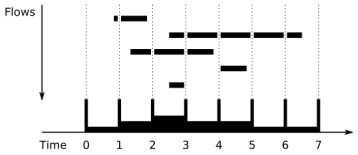

Fig. 2 Slicing flows into buckets

The aggregation framework is compatible with sharded databases which allows to execute the aggregation in parallel on multiple machines.

All flow data is pre-processed before it is stored in the database, which results in an overall architecture as shown in Figure 1. This pre-processing step includes temporal and spatial flow data aggregation as described in [5], and flow indexing for quick data access. Aggregation techniques are applied based on a given backend configuration, where operators can specify aggregation flow keys for spatial aggregation, and time intervals for temporal aggregation.

Time aggregation is essential in order to allow users to choose visualization intervals and analyze traffic over time. Flow-Inspector thereby integrates mechanisms for temporal aggregation and interval distribution [5]. Each flow is associated with a start and end time, i.e. the time of the first and last packet observed within a flow. The system configuration contains time intervals that define the visualization granularity (e.g. 5/10/30/60 minutes). Flows are sliced to match those intervals, called buckets throughout the rest of this paper, as shown in Figure 2. Flows that share the same aggregation flow keys in the same interval on the other handy aggregated into a single flow.

The stored data can be queried by an HTTP API provided by the server. This API further provides filtering mechanisms to purge data sets based on a client's request. Data rendering is exclusively performed on the client.

2.3 Frontend

Flow-inspector's frontend is implemented as an interactive JavaScript application that is automatically delivered when loading the website in a browser.

The core of this application utilizes D3.js [6], a library that allows data-driven manipulations of the website's document object model (DOM). With D3.js, complex interactive visualizations of arbitrary data can be efficiently build using HTML5, SVG and CSS. Additional JavaScript libraries, e.g. from the projects listed in [1], that provide specialized types of visualization can be integrated into the application as well.

Flow-Inspector also relies on BackBone.js [7], which provides a model-view-controller design architecture that facilitates extending the system with new visualizations. While providing new view classes that perform data rendering, model classes responsible for fetching data can be re-used. We created several visualization components based on this approach, which will be discussed in the following section.

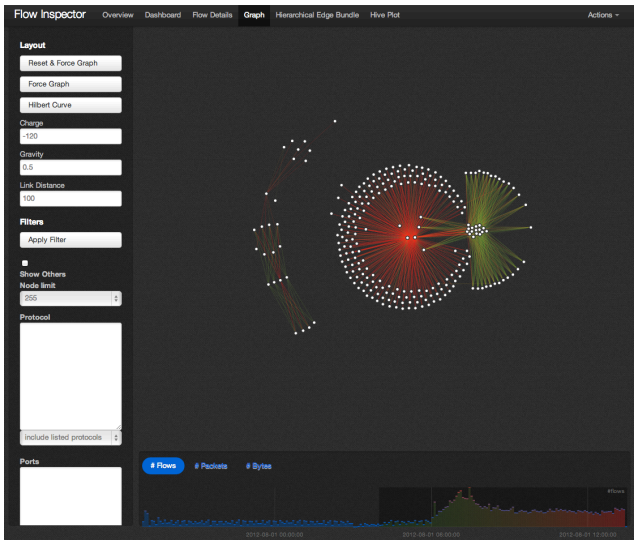

Fig. 3 Flow-Inspector: Control Interface

3 Visualizations

One of the most important steps for a visualization process is the proper selection of the parts of the data that should be displayed to the user. Figure 3 shows the control interface of Flow-Inspector that is shared by most of the views of the system. Each view includes three major regions.

The first region is the side bar on the left, and contains the filtering and control options of the view. These include common controls shared by all visualizations, such as fields for filtering for ports, protocols, or IP addresses. Furthermore, each visualization can add it's own controls. The example shows the Force Graph visualization, presented in Section 3.2, which provides some view specific operations that control the graph layout. Each control field provides information on its proper usage with tool tips that are displayed when the component is highlighted.

Time-based selection of traffic is performed in the bottom field of the view. This field contains an time-based overview of the traffic volumes. A time-interval can be selected by clicking on a start-interval and dragging the mouse over the available intervals.

The final and major component displays the flows that have been selected by the controls using one of the implemented visualization techniques. These are described in the remainder of this section.

3.1 Volume-based Visualization

Volume-based representations are the most common technique for presenting the current and past state of network traffic to an operator. They can be

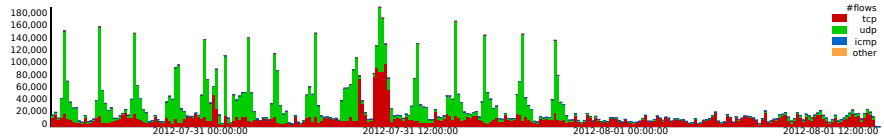

Fig. 4 Time series: number of flows over time

used to obtain an initial understanding of the time-dependent dynamics in a network. Time-series-based graphs that show the overall volume of traffic divided per transport protocols are generally available in systems that show network flows. Flow-Inspector supports such standard time-series-based views as well as views that allow to identify dominant hosts or services in the traffic data.

Typical time-series volume graphs, as shown in Figure 4, provide an overview about the number of flows, packets, and bytes observed in the user-defined buckets. They provide timeline views on the x-axes and display the amounts of flows, packets, or bytes split by transport layer protocol on the y-axes.

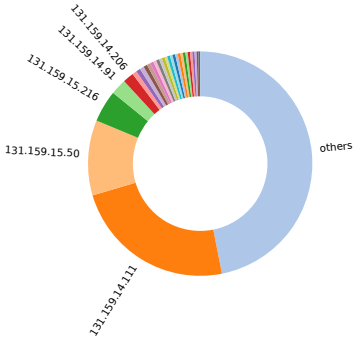

Fig. 5 Distributions of host with highest flow counts

Other pieces of helpful information are the global distributions of flows, packets or bytes among IP addresses (hosts) and ports (services). These distributions show the most active hosts and most heavily-used services in the network. Flow-Inspector uses donut charts to represent the share of individual IP addresses or ports in the total amount of traffic, as shown in Figure 5. All IP addresses or ports are sorted by one of the sums available in the precomputed buckets (flows, packets or bytes). The most active IPs or ports are then shown as individual segments, while all others are summarized into an "others" segment in order to avoid an overcrowded visualization.

An additional host overview graph displays the most active hosts sorted by bytes, packets or flows in table form. Figure 6 shows an example of a flow-based host overview graph. The host overview graph complements the donut view by breaking down its visualization into the different transport layer protocols and comparing only the most active IP addresses or ports.

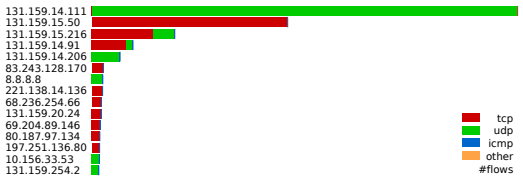

Fig. 6 TOP 15 hosts (most flows)

All views are associated with additional information that is only provided upon interaction between the user and the graphs. The graphs are connected to buttons that allow the user to choose a metric he wants to observe, e.g. distributions by number of flows, packets or bytes. Furthermore, every bar in the time-series and host overview graphs is provided with a tool tip that holds additional information. It is shown as soon as the user hovers over a portion of the graph.

Volume-based graphs can help to find time spans with unusual traffic, e.g. the time of service outages or network attacks. The views are especially helpful in order to identify time intervals that should be further investigated in other views. While such graphs provide oversight of overall traffic volumes, they do not provide insight into communication patterns between nodes.

3.2 Force Graphs

Node-flow graphs show these patterns by representing hosts or networks as nodes, and the communication flows between them as lines that connect the nodes. By adopting the line color depending on the time a given flow was observed, communication patterns and changes in those patterns over time can be made visible.

The human eye requires a meaningful layout in order to derive useful information from such a flow graph. Corresponding layouts should minimize line crossings by grouping the nodes in a way such that connections overlap as little as possible.

One of the most popular approaches to generate graph layouts with respect to visualizations of relationships between entities with little overlap are force-directed layouts. There are various algorithms available, but most of them share the idea of a physics-based, iterative simulation until a power equilibrium is reached.

Flow-Inspector uses the force-graph layout algorithm provided by D3.js, which uses position Verlet integration [8]. Links between nodes are considered as a weak geometric constraint that have a desired length. Initially, nodes are randomly positioned on the canvas resulting in a non-optimal layout. The force-graph algorithm then tries to iteratively optimize the position of all nodes under their geometric constraints, computes new positions and then optimizes them again. An example of a node-flow graph with a force-directed layout can be seen in Figure 7.

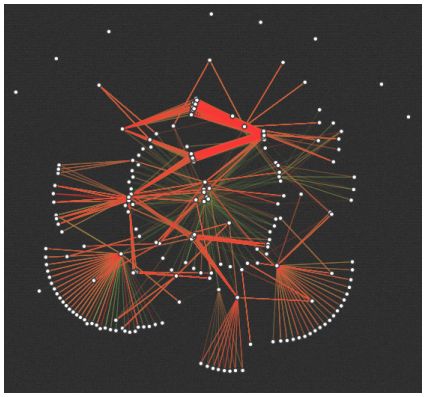

Fig. 7 Force Graph

In Flow-Inspector, rendering and calculating of the force graph layout is carried out after a user selects a time interval. All observed flows during this time interval define connections between nodes. The graph is calculated and drawn in multiple iterations, until a steady state is reached or the user aborts the iterative algorithm by clicking onto the graph.

If the user modifies the selected time range changes, which selects other flows between the nodes, all new flows are drawn into the previously calculated force-graph image. This enhances the comparability between the time ranges. By clicking the force button in the control area, the user can derive a new force-layout from the newly loaded flows. This new force-graph can be created based on the previous layout, or based on a new randomized placement of the nodes.

Force-directed layouts can often reveal structures in a graph which might be hard to recognize in trivial layouts. They minimize the length of connections and line crossings and usually lead to graph drawings with a natural look. In contrast to static visualizations of such graphs, Flow-Inspector can attach additional information to its nodes: Hovering over nodes in interesting communication patterns reveals their IP address which can be used for further investigations.

Filtering of data to an interesting subset is nevertheless a crucial part of this visualization technique. If the number of rendered nodes and flows grows to large, the graph will most likely contain many overlapping connections, which decreases the visibility and readability of communication patterns.

3.3 Hierarchical Edge Bundles

A drawback of force graphs is that two nodes are generally connected using a straight line, which can result in unwanted line crossings.

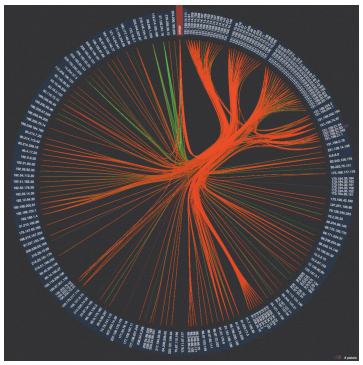

Fig. 8 Hierarchical Edge Bundle

Large-scale data sets with many nodes and connections tend to lead to massive line crossings. Bundle graphs are one way to group the connections between entities in a more comprehensible way.

The basic idea behind edge bundle graphs is to create bundles of similar connections to increase the visibility of connection structures between end systems. An example for those edge bundles is shown in Figure 8, where entities are organized on a circle. Flow-Inspectors supports this technique with entities that are either IP addresses or networks in CIDR notation.

The layout on the circle is defined by a radial tree layout that groups IP addresses based on membership of a network. IP addresses in the same networks are mapped close to each other with additional space between networks.

Flows between these IPs are drawn as bundled lines using hierarchical edge bundling. The lines are interpolated with a piecewise cubic B-spline. A tension parameter between zero and one allows to control the attraction of the line to its control points, which affects the tightness of the flow bundles.

The graph itself is an interactive graph with several ways to filter and display additional information upon user request: A timeline at the bottom of the page allows a user to select the time range that should be visualized. If multiple buckets are selected, the color of the lines represent the buckets in which corresponding flows have been observed. When rendering the image, all available information for an entity (i.e. number of flows, packets, bytes) is loaded from the database and provided as a tooltip. In addition, if the user hovers over a node, only flows that involve the corresponding IP address are rendered.

3.4 Hive Plots

A relatively new approach to visualize large-scale data are Hive Plots [2]. An example for a hive plot is shown in Figure 9. To the best of our knowledge, no one has previously used hive plots for visualizing Netflow or IPFIX data.

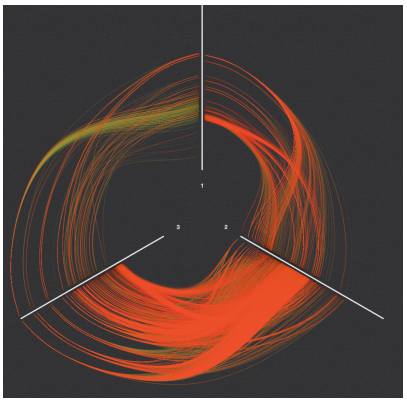

Fig. 9 Hive Plot

The developers of the hive plot concept criticize force-directed methods for visualizing large data sets as "hairballs", which do not provide much insight into the data. Unreasonable positioning of nodes is criticized as well: the resulting graph is often confusing. Hive plots are a completely different and highly customizable way to make trends visible in a huge set of relations.

A hive plot consists of a predefined number of axes arranged on a radial layout. Flow-Inspector uses three axes for mapping IP addresses or networks. When rendering data, each node needs to be assigned to one of the axes (node-to-axis mapping). Nodes are thereby positioned on axes using some feasible heuristic (position mapping). The links of the graph are drawn as lines between two axes starting and ending at the corresponding node positions. This approach is especially useful in analysis scenarios that aim at getting insight into large communication structures.

Mapping and position of nodes to axes is crucial for the insight that a Hive Plot can provide. Furthermore, any interpretation heavily depends on the specific node-to-axis and position mapping. In Flow-Inspector, those mappings are therefore configurable by the user.

The configuration sidebar contains one input field for each axis in the hive plot. Those input fields expect lists of IP addresses/networks in CIDR notation.

The flows between the specified nodes are drawn as B-splines between two adjacent axes, starting and ending at the position of the nodes. Hive plots can be used to display the traffic relationship between networks mapped to axes.

Operators can asses the amount of traffic flows exchanged between networks. The visualization can also be used to determine whether there are unexpected or interesting traffic flows between parts of the networks. For example, flows can be unexpected due to access rules that should not allow those traffic streams. This allows to identify errors in firewall configurations.

The plots can also help to understand the traffic flows in complex applications with frontend and backend traffic: By mapping clients to one axis,

frontend servers to the second axis, and backend servers to the third axis, communication patterns that are caused by frontend traffic can be observed.

Hive plots can also be used to analyze complex load-balancing environments, which often use multiple layers of load-balancing. Clients can be assigned to one or more load-balancing servers by a DNS-based round-robin scheme. The load-balancing services can relay the incoming connections to a large number of backend servers, which then respond to the actual client requests. By visually analyzing corresponding scenarios, the quality of a load-balancing process can be estimated.

4 Related Work

The need for visualization of traffic data resulted in many systems with different analysis purposes. Some of them display information on traffic volumes and constituencies. NfSen [9] and FlowScan [10] provide mechanisms for visualizing Netflow data that is generated by monitoring probes, routers, or switches. Their visualization methods concentrate on presenting traffic volumes including information about the used transport protocols.

Ntop [11] is a web-based system for analysing and visualizing traffic information and also focuses on flow traffic analysis. It provides tools for collecting and generating flow information for its visualization process. In addition to standard volume-based visualizations, ntop provides tools for combining flow information with other data sources such as BGP data or IP-based geographical information.

Our system differs from these approaches: NfSen, FlowScan and ntop perform the visualization at server side, providing fixed images to the user. Flow-Inspector on the other hand renders its images in a client's browser. This allows for providing users with a higher level of interaction.

Other approaches do not provide web interfaces for visualizations tasks. Instead, they created full-fledged client applications that perform the rendering. These, usually platform-depended applications, can make use of 3D capabilities of the client systems' graphic cards.

In [12], the authors present the client-based visualization tool FlowVis, which uses the SiLK tools for processing NetFlow data. They provide proof-of-concept visualizations like activity plots, flow edge bundles, and network bytes viewer. Lakkaraju et al. presented NVisionIP [13], a tool for displaying traffic patterns in class-B networks using scatter plots and volume-based visualizations. Yin et al. presented an animated link analysis tool for Netflow data [14], which leverages a parallel coordinate plot to highlight dependencies in the network.

Flow-Inspector has several advantages compared to those approaches. Due to its web-based nature, any computer with a modern browser can be used to interact with Flow-Inspector without any additional software installations required. Another major advantage is Flow-Inspector's extensibility. An active community of JavaScript developers is working on visualization libraries for

various purposes. Although such libraries might aim at implementing new visualization techniques for tasks beyond network flow analysis, corresponding approaches can often be adopted for displaying traffic data. Flow-Inspector users can benefit from such developments due to its extensible framework that allows to easily integrate these new visualization libraries.

5 Conclusion

In this paper we introduced Flow-Inspector, an interactive web application for dynamic network flow visualization. For network operators, visualization of traffic data is an important tool that can help to understand network characteristics and to identify immediate and long-term problems. We therefore encourage network operators to use our framework.

Our paper further aims at making Flow-Inspector available to the research community. Researchers can use it to build new and innovative visualization tools for network traffic data. The code is available for download at http://flow-inspector.net.in.tum.de, and we invite others to use and extend the system.

References

1. "Tools for Data Visualization," http://selection.datavisualization.ch, Visited: Nov. 2012.
2. M. Krzywinski, I. Birol, S. J. Jones, and M. Marra, "Hive Plots - Rational Approach to Visualizing Networks," *Briefings in Bioinformatics*, Dec. 2011.
3. B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," RFC 5101 (Proposed Standard), Internet Engineering Task Force, Jan. 2008.
4. "Mongo DB website," http://www.mongodb.org, Visited: Nov. 2012.
5. B. Trammell, A. Wagner, and B. Claise, "Flow Aggregation for the IP Flow Information Export (IPFIX) Protocol," Work-in-progress document, http://tools.ietf.org/html/draft-ietf-ipfix-a9n-05, Jul. 2012.
6. "D3 JS – Data-Driven Documents, Website," http://d3js.org, Visited: Nov. 2012.
7. "Backbone.js Website," http://backbonejs.org, Visited: Nov. 2012.
8. L. Verlet, ""Computer Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules," *Physical Review*, vol. 159, pp. 98–103, 1967.
9. "NfSen homepage," http://nfsen.sourceforge.net, Visited: Nov. 2012.
10. D. Plonka, "Flowscan: A Network Traffic Flow Reporting and Visualization Tool," in *USENIX LISA'00*, New Orleans,LA, Dec. 2000.
11. "Ntop Website," http://www.ntop.org, Visited: Nov. 2012.
12. T. Taylor, D. Paterson, J. Glanfield, C. Gates, S. Brooks, and J. McHugh, "Flovis: Flow Visualization System," in *Conference For Homeland Security 2009. CATCH'09*, Mar. 2009.
13. K. Lakkaraju, W. Yurcik, and A. J. Lee, "NVisionIP: NetFlow Visualizations of System State for Security Situational Awareness," in *Proceedings of VizSEC/DMSEC '04*, Washington, DC, Oct. 2004.
14. X. Yin, W. Yurcik, and A. Slagell, "VisFlowConnect-IP: An Animated Link Analysis Tool for Visualizing Netflows," *FLOCON 2005*.