

# Enhancing Visual Analysis of Network Traffic Using a Knowledge Representation

Ling Xiao\* John Gerth\* Pat Hanrahan\*

Stanford University

## ABSTRACT

This paper presents a network traffic analysis system that couples visual analysis with a declarative knowledge representation. The system supports multiple iterations of the sense-making loop of analytic reasoning by allowing users to save discoveries as they are found and to reuse them in future iterations. We will show how the knowledge representation can be used to improve both the visual representations and the basic analytical tasks of filtering and changing level of detail. We will describe how the system can be used to produce models of network patterns, and show results from classifying one day of network traffic in our laboratory.

**CR Categories and Subject Descriptors:** I.6.9 [Visualization] information visualization, H.5.0 [Information Interfaces and Presentation] general.

**Keywords:** network traffic visualization, visual analysis

## 1 INTRODUCTION

The last decade has seen a rapid growth in both the volume and variety of network traffic, while at the same time it is becoming ever more important for analysts to understand network behaviors to provide quality of service, security, and misuse monitoring. To aid analysts in these tasks, researchers are seeking better visual analysis techniques for network traffic.

The sense-making loop of information visualization is critical for analysis [6]. The loop involves a repeated sequence of hypothesis, experiment, and discovery. However, current visual analysis systems for network traffic do not support sense-making well because they provide no means for analysts to save their discoveries and build upon them. As such, it becomes the analyst's burden to remember and reason about the multitude of patterns observed during visual analysis, which quickly becomes impossible in the large datasets typical of network traffic.

In this paper we present a network traffic visualization system that enables previous visual discoveries to be used in analysis. The system accomplishes this by allowing the analyst to interactively create models of observed patterns, which are stored in a reusable knowledge base. The reuse of knowledge creates the analytical cycle as summarized in Figure 1. In addition to facilitating the sense-making loop, knowledge representations can afford more insightful visualizations that the analyst can use to discover more complex and subtle patterns.

To illustrate its effectiveness, we will present the results of using our system to analyze a day of network traffic from our laboratory.

This paper will be structured as follows: Section 2 will provide an overview of the analysis process supported by the system; Section 3 will give a sampling of related work in this area; Section 4 will describe the system's knowledge representation; Section 5 will explain the interactive model creation process; Section 6 will present a pilot study which used the system to analyze one day of

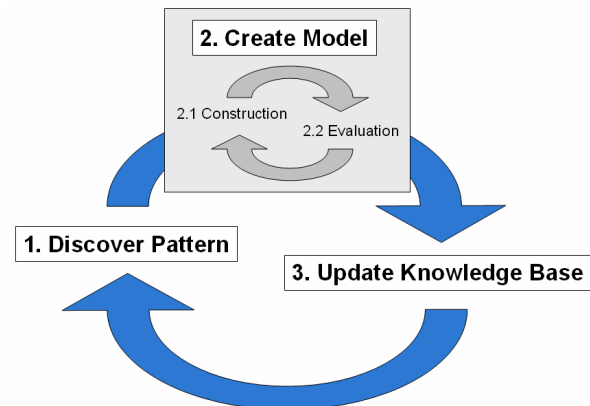


Figure 1. The analysis cycle supported by the system. (1) The analyst uses visualization enhanced with previous knowledge to discover patterns in the data. (2) Once a pattern is discovered, the analyst creates a model for the pattern. (3) The analyst commits the model into the knowledge base to reuse in future analysis.

network traffic in our laboratory; Section 7 will discuss the shortcomings of the current implementation; and Section 8 will conclude the paper and provide future research directions.

## 2 SYSTEM OVERVIEW

The system supports the analysis cycle outlined in Figure 1. In explaining the system, we will use the following terms: a *pattern* refers to an ordered sequence of events that can be described by a *model* in a declarative language; thus, a model is a general description of a pattern; and a *clause* is a logical sentence realizing a model.

**Stage 1 – Discover Pattern:** The analyst uses the visualization to discover patterns in the data. For example, in a temporal visualization of HTTP connections, the analyst might see a sequence of connections bunched together in time and infer that this represents the load of a web page, since a web page often makes additional connections as it loads the URLs for images, ads, etc.

**Stage 2 - Construct Model:** Once a pattern is discovered, the analyst constructs a model for the pattern. To capture the pattern exhibited by a web page load, the analyst selects one horizontal sequence of marks from the visualization. The system then identifies predefined predicates that are true for the selected events. In this example the following predicates are identified: “from same IP”, “to same IP”, “temporal locality”, “source port locality”, “destination port HTTP”.

The analyst then engages in an interactive loop to create a clause describing the pattern from the identified predicates:

**Stage 2.1 – Construction:** The analyst constructs a candidate model of the pattern. In the web page example, the analyst may choose the single predicate “to same IP”, after which the system updates the visualization.

**Stage 2.2 – Evaluation:** The analyst evaluates the candidate model and refines it if needed. Here the analyst may see that the clause “to same IP” is too broad and needs to be further qualified, so he incrementally conjuncts other predicates such as “temporal locality”, “from same IP” and “source port locality”.

The analyst continues iterations of Construction and Evaluation until he converges on an accurate description of the pattern.

**Stage 3 – Update Knowledge Base:** The analyst commits the clause produced in Stage 2 into the knowledge base by giving the pattern a name, e.g. WebPageLoad. The system then labels all instances of the pattern in the data, and produces abstract events representing instances of the pattern. The analyst can then use the labels and abstract events in subsequent visual analysis.

Thus, in each iteration of the analysis cycle the system incrementally incorporates the analyst’s discoveries.

### 3 RELATED WORK

Visualizations of internet network traffic patterns have existed for over a decade [5, 17]. Early visualizations were aimed at understanding overall network loads and topology. These have evolved into tools for monitoring quality of service at the scale of the major network trunks [12]. More recently a variety of visualizations have been developed in both the research [8, 4, 10] and commercial worlds [2, 18] to help analysts understand traffic at different scales ranging from the corporate intranet down to individual machines.

To produce classification models that correspond to analysts’ domain knowledge, researchers have also proposed visual data mining systems. PBC [1] and PaintingClass [21] both allow the analyst to use interactive visualization to produce decision trees.

The declarative language of our system is based on event patterns, which we model using clauses in first order logic [7]. There are numerous event pattern languages such as GEM [15], RAPIDE [14], and SEL [23].

Our event structure is very similar to that found in Complex Event Processing [14], which uses the RAPIDE event pattern language to describe patterns based on events organized in a Concept Abstraction Hierarchy. This abstraction process creates a hierarchical organization of events that summarizes and correlates lower level events with more abstract higher level events for multiple levels of detail. CEP has been used for event correlation on network event streams to facilitate analysis, reduce false alarms, and detect coordinated attacks.

As a whole, our work is most closely related to that found in NVisionIP [22] which has recently added monitoring of user actions [13]. In NVisionIP the analyst’s selection actions are recorded as a pattern tree using a rule set based on the well-known Berkeley Packet Filter language [16]. These patterns may be saved in a file and subsequently applied to other network datasets, thereby permitting the analyst to capture the value of a particular set of filtering operations as reusable knowledge. We expand the notion of reuse by using a knowledge representation that goes beyond flow attributes, and is extended as the analyst leverages the knowledge base to perform visual analysis.

### 4 KNOWLEDGE REPRESENTATION

In this section we will describe the type of network traffic that the system analyzes and the knowledge representation used.

Dimensions	Measures
GMT Start time	Duration
IP protocol	Source packets
Source ASN	Source bytes
Source IP address	Source application bytes
Source port	Destination packets
Destination ASN	Destination bytes
Destination IP address	Destination application bytes
Destination port	

Table 1. Flow table fields

#### 4.1 Data

The system is designed for the analysis of network flow data. Flow data is increasingly used in network analysis because it does not contain packet payloads and is therefore more scalable. Moreover, flow data also avoids many of the social and legal issues associated with packet content recording and is more robust in the face of encryption [11].

In our system we collect flows produced by the open source Argus [3] sensor; storing them with anonymized IP addresses in a relational database where the rows are events and the columns are fields. The fields of a flow table are shown in Table 1.

#### 4.2 Pattern Language

The system uses a declarative event pattern language (EPL) based on first-order logic. First-order logic is a well-studied knowledge representation that is both amenable to automated reasoning, and relatively easy for analysts to understand [9]. Patterns are described using clauses composed of logical connectives and predicates. For example, the following clause describes a special case of the web page load pattern involving only two flows.

```
SimpleWebPageLoad(x,y) =
  identical_source_IP(x,y) AND
  identical_destination_IP(x,y) AND
  time_within_2_seconds(x,y) AND
  ( destination_port_80(x) AND
    destination_port_80(y) )
```

Free variables represent network events. The clause above describes two flows from the same IP, to the same IP, within 2 seconds of each other, and both to port 80 (associated with HTTP traffic). In first-order logic, predicates cannot take parameters as arguments; therefore it is necessary to create separate predicates for each parameter value. While tedious, we have found that most parameters we desire require only a few values.

We distinguish four types of predicates. *Unary* predicates have only one argument; *n-ary* predicates have multiple arguments, but the number of arguments is fixed (currently limited to 5); *sequential* predicates accept a partially ordered set of events as arguments; and *group* predicates accept an unordered set of events as arguments.

Sequential and group predicates describe patterns composed of repetitive sequences of events with no pre-determined length. This type of pattern is very common in network traffic due to the structure of protocols. Using sequential and group predicates, we can generalize the clause above for SimpleWebPageLoad as:

```
WebPageLoad(x1,x2,...) =
  identical_source_IP(x1,x2,...) AND
  identical_destination_IP(x1,x2,...) AND
  time_sequence_2_seconds(x1,x2,...) AND
  ( destination_port_80(x1) AND
    Destination_port_80(x2) AND ... )
```

Knowledge Type	Sample Predicate Descriptions
Location	(U) Dest IP is DNS server (U) Src ASN is Google
Connection	(U) Protocol is TCP (U) Dest port is 80
Traffic measurements	(U) Total bytes sent is > 3000 (U) The duration is < 1 s
Temporal	(N) One event before another (S) In tight time sequence
Equality	(N) Two events have the same src IP (S) Has same dest AS number
Counts	(G) Number of destination IP > 20
Approximate	(U) Data is within 2KB of 20KB
Trend	(S) Amount of data is increasing (S) Dest IP number is increasing
Variability	(G) High distinct dest port usage (G) High destination IP access rate

Table 2. Selected predicates from the system. (U) unary predicate (N) n-ary predicate (S) sequential predicate (G) group predicate

To model network traffic, the system is initialized with a set of predicates, such as those shown in Table 1. The predicates describe a variety of traffic characteristics: some about the source or destination, the type of connection, the characteristics of the traffic, temporal relations and trends, variability etc. Others are provided for efficiency. The system also provides an interface for analysts to add their own predicates.

#### 4.3 Update Knowledge Base

To update the knowledge base with a pattern, the system searches for instances of the pattern in the data, which are true groundings of the pattern’s clause.

To search for true groundings of a clause, we first convert the clause into a disjunction of conjunctions, and then ground each conjunction independently. We use one of two strategies to ground a conjunction depending on the type of predicates in the conjunction. We do not currently handle conjunctions involving n-ary with sequential or group predicates; and conjunctions involving only unary and group predicates.

For conjunctions of unary and n-ary predicates, we convert the clause into a SQL query that returns a superset of the true groundings. We then evaluate the clause on each of the returned groundings to determine its truth value.

Each sequential predicate is defined with an ordering function. For example, a temporal predicate would have a time based ordering. To ground a conjunction involving sequential predicates, we compute an ordering of the events from the sequential predicates’ order functions, such that true groundings of the clause are a run. We then perform a scan through the ordered events for runs, and evaluate the clause on each run. For efficiency, we do not consider subsequences of runs.

For each pattern instance found, the system associates the name of the pattern with the constituent events of the pattern instance.

The system will then create a higher level event representing each pattern instance in a new event table. For example, if the sets of flows {a,b,c} and {d,e} both satisfied the WebPageLoad clause, we will create a higher level event *abc* that corresponds to {a,b,c}, and another event *de* that corresponds to {d,e}. Intuitively the event *abc* represents the web access event that caused the flows {a,b,c} to be observed, and similarly for {d,e}. The most primitive events are the Argus flows themselves.

The attributes of higher level events are computed from their constituent lower level events. The names of attributes are

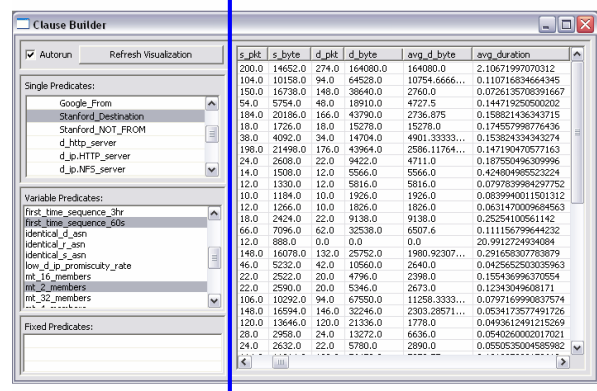


Figure 2. The interface that is used to create candidate clauses (Left) List of identified predicates that can be used in the pattern description clause (Right) The details of the selected events

semantically unique in the system. For example, the “first time” of a higher level event is the earliest “first time” of its constituent events, and therefore is semantically equivalent. Predicates are defined in terms of attributes, and are applicable to event types that have the required attributes.

## 5 MODEL CREATION

Once a visual pattern is discovered, the analyst constructs a model by selecting the pattern directly on the visualization, and then exploring the model space by iteratively constructing and evaluating candidate clauses; eventually converging to a clause that he believes captures the pattern.

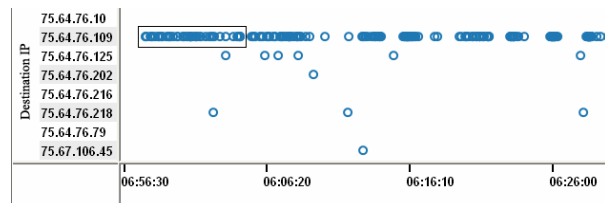
In this section, we demonstrate this approach by extending the event hierarchy we started with WebPageLoad events in Section 2 to encompass web crawls made by search engines.

### 5.1 Selecting Patterns

To discover patterns, the analysts may employ a variety of visualizations, including scatter plots, Gantt charts, and parallel plots. Due to the strong temporal nature of network event analysis, we have found that event diagrams are especially useful. An event diagram is a plot that maps time on the X-axis and a dimension or measure on the Y-axis using a circular mark for each event.

Once a pattern is found, the analyst selects an instance of the pattern by directly indicating the set of marks on the visualization with a mouse, either individually or by capturing groups in boxes.

In the web crawl scenario, an analyst might be looking at an event diagram showing WebPageLoad events to local IP’s (on the Y-axis). The analyst observes that there are numerous horizontal strings of events, corresponding to rapid bursts of WebPageLoad events, each of which might be an example of a web crawl. He selects one string of events as shown below.



## 5.2 Identifying Predicates

After selecting the pattern, the analyst is shown (1) a list of predicates that describe the pattern along with (2) the details of the selected events (see Figure 2)

To identify predicates that describe the pattern, each predicate in the knowledge base is tested on the selected events. Different strategies are used depending on predicate type.

The system returns a unary predicate if it is satisfied by over 90% of the selected events.

The system returns an n-ary, sequential, or group predicate if it can be satisfied by the selected events. The truth value of predicates with multiple arguments depends on the assignment order of events to arguments. Therefore, to determine if a predicate can be satisfied by a set of events, we would need to evaluate the predicate on all permutations of the events. To reduce computation cost and maintain interactive response time, the system imposes constraints for identifying n-ary, sequential and group predicates.

To test an n-ary predicate, all possible permutations of the events are evaluated; however, at most 120 permutations are required, since the number of arguments for n-ary predicates is limited to 5.

To test a sequential predicate, we sort the events using the predicate's order function, and evaluate the predicate on the ordered events.

To test a group predicate, whose truth value is independent of the arguments' assignment order, only one evaluation is needed.

In the web crawl scenario, the identified predicates include:

```
destination_port_80,
destination_Stanford,
identical_source_AS_number,
time_sequence_30s,
time_sequence_60s,
more_than_4_events,
more_than_32_events
```

## 5.3 Creating a Clause

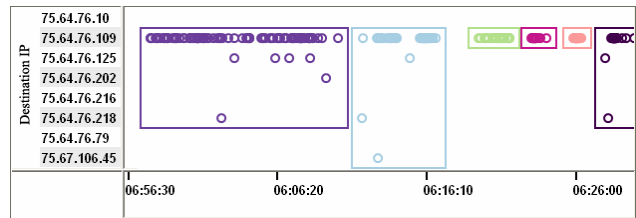
To create a clause describing the pattern, the analyst interactively constructs and evaluates candidate clauses, which are currently limited to conjunctions. To generate a candidate conjunction, the analyst selects a combination of identified predicates using the interface shown in Figure 2. The system then updates the visualization to display groups of events that satisfy the current clause. The analyst uses the visualization to evaluate the clause, and decide what, if any, refinement is required. The process of refinement and visual evaluation is repeated until the analyst converges on a satisfactory model of the pattern.

This interactive process is inspired by dynamic queries, which allow users to explore the query space by manipulating the query parameters with widgets that correspond to the schema of the query in a tightly coupled manner. The tight coupling provides value in a number of ways: consistent affordances; rapid, incremental, and reversible interactions; continuous display of the information space; progressive refinements; and others [20].

In our system, the model space consists of all conjunctions that can be formed from the identified predicates. The interface allows users to add and remove predicates from the candidate clause through selection and de-selection. Each change is incremental and reversible, and every change to the candidate clause is reflected on the visualization canvas.

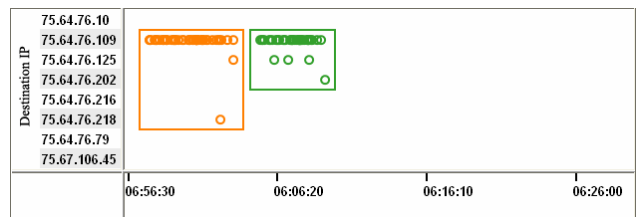
In the web crawl scenario, the analyst knows that a web crawl accesses numerous pages possibly from multiple web servers in quick succession. Therefore, he selects the predicates “time

sequence 60 seconds” and “more than 4 events”. The groups of events satisfying the conjunction are shown below, where each group is a uniform color surrounded by a bounding box.



Unfortunately, this simple conjunction is satisfied by several spurious event groups. In particular, the purple rectangle has captured multiple bursts that are more likely to be two web crawls, and the small boxes on the right are such short sequences of WebPageLoad events that they are more likely due to human navigation. Therefore, the analyst refines the clause by selecting “more than 32 events” and “time sequence 30 seconds”.

Since this conjunction is more restrictive, only two groups of events satisfy the clause, as shown below where each box encloses a rapid and lengthy sequence of WebPageLoad events characteristic of a web crawl. Note that a web crawl may involve several web servers (circles at different Y values). This is because web pages frequently reference pages from other web servers.



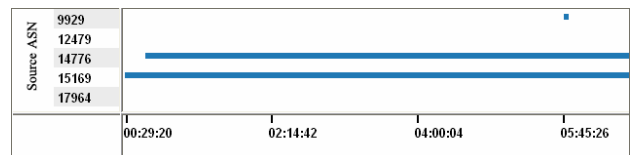
Once the analyst has converged on a clause that describes the pattern, he names the pattern, and commits the clause to the knowledge base. In the web crawl scenario, the final clause is:

```
WebCrawl(x1, x2, ...) =
time_sequence_30s(x1, x2, ...) AND
more_than_32_events(x1, x2, ...) AND
identical_source_AS_number(x1, x2, ...) AND
( is_web_access_event(x1) AND
is_web_access_event(x2) AND ...)
```

The predicate “identical source AS number” generalizes the pattern to other source ASN’s of HTTP requests beside Google.

## 5.4 Updating the Knowledge Base

Once the analyst creates a model, the system updates the knowledge base as described in Section 4.3. The Gantt chart below shows the clause capturing web crawls from Google (ASN 15169) and Inktomi (ASN 14776).



## 6 PILOT STUDY

To investigate the effectiveness of enhancing visual analysis using a knowledge representation, we conducted a pilot study using real network traffic. The goal of the pilot study was to visually identify network event patterns and produce pattern models that can be used to classify traffic. We were interested in the accuracy of the resulting classification and the overall usability of the system for this task.

### 6.1 Input Data

The data for the study consisted of all network flows in our laboratory over a 24 hour workday. We chose this sample to have a variety of network behaviors with different machines and a time span that is familiar but interesting for an analyst. Some summary statistics for the dataset are shown in Table 3.

### 6.2 Traffic Classification

Before we started the pilot study, we discussed the types of network behaviors that are likely to be observed with our network staff. Based on their input, we defined an initial set of a hundred predicates that captured the characteristics of those behaviors (a subset was shown earlier in Table 2).

Using these predicates and the interactive analysis cycle of Figure 1, we produced the 21 models listed in Table 4. These models were able to classify approximately 80% of the flows in the sample dataset. Most of the models were simple conjunctions, two of which are shown here:

```
FastScan(x1,x2,...) =
  identical_src_ip(x1,x2,...) AND
  dest_ip_in_sequence(x1,x2,...) AND
  more_than_50_events(x1,x2,...) AND
  high_dest_ip_access_rate(x1,x2,...) AND
  (is_tcp(x1) AND is_tcp(x2) AND ...) AND
  ( low_duration(x1) AND
    low_duration(x2) AND ...) AND
  ( low_total_data(x1) AND
    low_total_data(x2) AND ...)
```

```
Portmap(x1,x2,...) =
  first_dest_port_111(x1,x2,...) AND
  identical_src_ip(x1,x2,...) AND
  identical_dest_ip(x1,x2,...) AND
  time_sequence_0.5s(x1,x2,...)
```

We found that the process of interactive analysis stimulated the analyst to recall additional facts about network behaviors from their visual context. For example, while modeling the SSH login pattern, the analyst would create a candidate clause, see some residual traffic, perhaps an NFS auto-mount, and then realize that it too should be part of the login. We believe many of the models could not easily have been produced without an interactive analysis system.

To quantify the accuracy of the classification precisely, we would require an independent way of classifying the traffic on our network, which we did not have. Instead, we performed extensive visual inspection on the classified traffic, and hand-verified patterns by examining the raw data they were derived from.

Property	Value
Number of flows	1,297,635
Distinct local IP addresses	1,065
Distinct remote IP addresses	39,589
Total bytes sent and received	168,012,562,973
Total number of packets	207,121,399

Table 3. Statistics of flows collected for the pilot study.

Model Name	# Flows	% of Traffic
Chat	12,527	1.0 %
DNS lookup	131,764	10.2 %
Fast scan	19,322	1.5 %
IMAP mail	15,885	1.2 %
LDAP query	121,365	9.4 %
Microsoft file access	57,461	4.4 %
Multicast	24,858	1.9 %
Printer	23,246	1.8 %
NFS file access	62,103	4.8 %
NTP time sync	35,027	2.7 %
Pop mail	1,763	0.1 %
Port scan	336	0.02 %
SUN portmap	45,273	3.5 %
Wrong direction	25,824	2.0 %
Sendmail	19,322	1.5 %
Ganglia cluster monitor	85,936	6.6 %
Slow scan	41,639	3.2 %
SSH dictionary attack	34,353	2.6 %
SSH successful login	11,479	0.9 %
Successful login	23,605	1.8 %
Web crawl	72,622	5.6 %
Web page load	386,652	29.9 %

Table 4. Models and percentage of traffic associated with each

Type	Flows	Events	Flows/Event
Mail	10,178	8,355	1.22
Port map	45,139	15,538	2.91
HTTP	375,563	58,019	6.47
Chat	12,537	237	52.90
Port scan	336	6	112.00
SSH Attack	34,353	89	385.99
SSH Login	11,479	23	499.09
Web Crawl	72,622	125	580.98
Scan	60,691	52	1167.13

Table 5. The number of flows per high level event, by type.

The errors in our classification can be divided into the following three categories, of which the first contained the majority of cases:

1. The underlying data for the flows was faulty. This is caused by the sensor dropping data during overloaded sessions; and Argus occasionally confusing the orientation of the flows.
2. Some traffic was misclassified by errors in our models. We were often able to correct these mistakes by refining the models.
3. Some traffic could not be classified because our EPL was insufficiently expressive to describe their characteristics.

The knowledge produced is evidence that analysts can use the system to identify network behaviors and produce accurate models of the patterns exhibited by these behaviors.

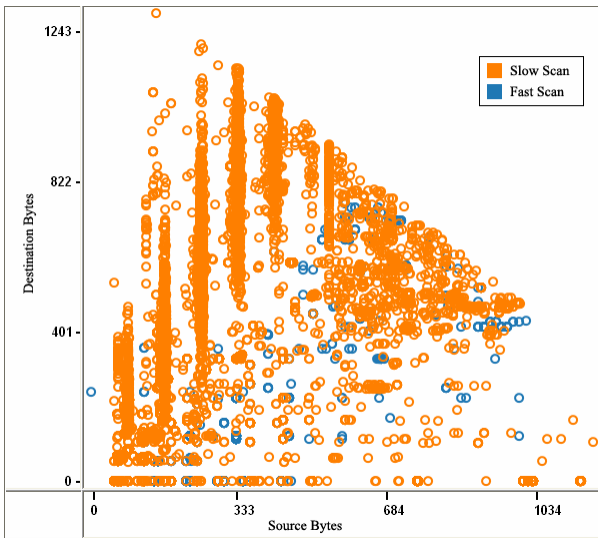


Figure 4. Relationship between the number of bytes from the source vs. the destination of an individual flow. Only flows from fast and slow scans are shown

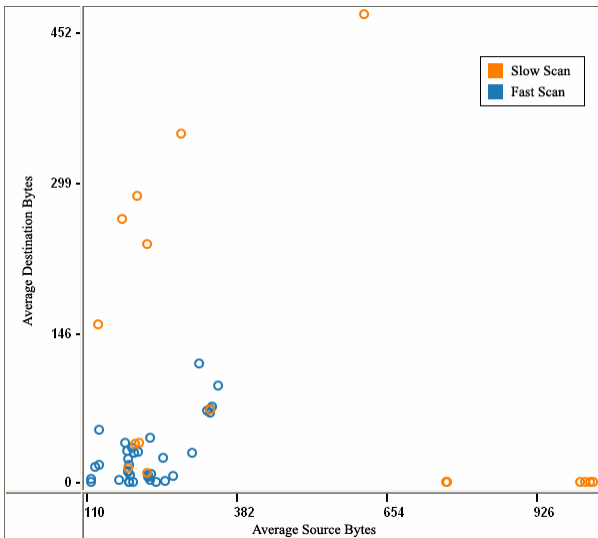


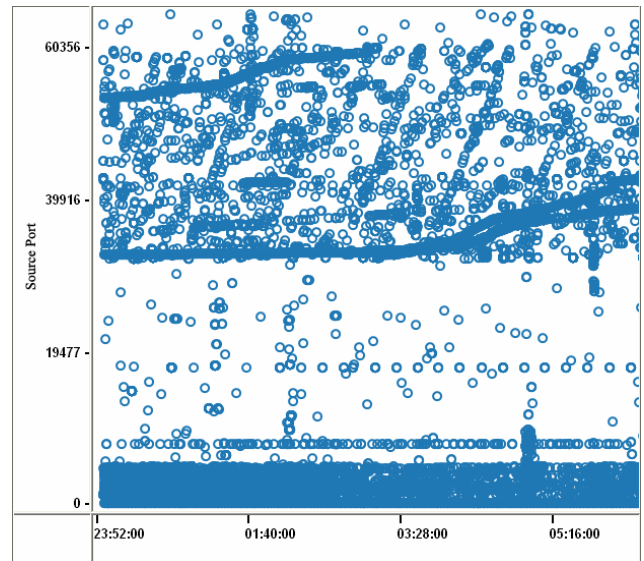
Figure 5. Average destination and source bytes for high-level fast and slow scan events computed over all the flows comprising each scan event. Observe that the traffic from slow scans is asymmetric, tending to have either high destination bytes with low source bytes, or vice versa. In contrast, fast scans tend to transfer fewer bytes and the average number of bytes in the source and destination are more balanced.

### 6.3 Visual Analytics

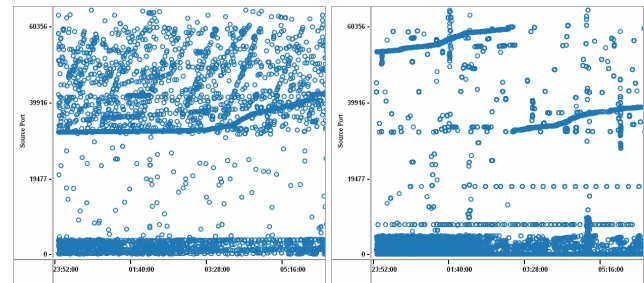
During the pilot study, we identified network event patterns using visual analytic techniques that benefited from the knowledge representation.

#### 6.3.1 Changing Level of Detail by Abstraction

We found that a better understanding of network behavior can be gained by focusing on high-level events instead of raw flow events. We believe this is because high-level events have less detail and reduce the complexity and noise characteristic of the voluminous low level events.

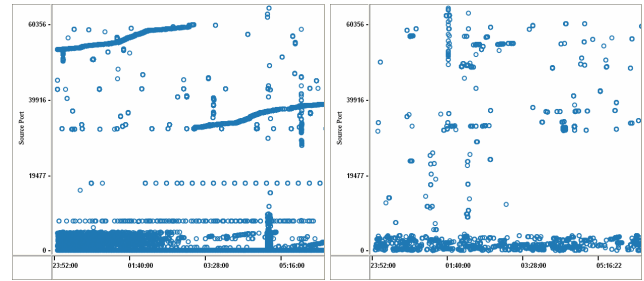


(a)



(b)

(c)



(d)

(e)

Figure 6. Event diagrams showing flows during residual analysis. (a) Original unidentified traffic (b) Flows with "mail" label (c) The residual after filtering out "mail" from Figure 6a. (d) Flows with the "scan" label (e) The residual after filtering out the "scan" label from Figure 6(c).

As an example of how higher-level events reveal larger patterns in the data, consider the structure of scan traffic. Figure 4 shows the bytes transferred for individual flows, while Figure 5 shows the aggregated traffic for each individual fast or slow scan. There are many fewer scan events than flow events, reducing the complexity of the display, and making the characteristics of each scan type more evident: slow scans tend to have more extreme ratios of source to destination bytes than fast scans; and fast scans tend to send and receive less data than slow scans.

To quantify the reduction in complexity, we calculated how many flow events are collapsed into each high level event by traffic type, as shown in Table 5.

### 6.3.2 Semantic Filtering

We were able to form semantically meaningful filters using the model labels. To illustrate, consider these sample iterations of the analysis cycle. After each iteration, we removed newly-classified traffic to yield residuals for further analysis.

Let us assume that we are currently examining the event diagram of Figure 6(a). Using the model creation process, we produce models of the patterns shown in Figure 6(b), which represent mail traffic. We then compute the residual by filtering out traffic with label “mail” from Figure 6(a) to yield Figure 6(c), in which we can observe and model the patterns shown in Figure 6(d) – which are scans. Then in turn, we compute the residual by filtering out traffic with label “scan” from Figure 6(c), to produce Figure 6(d). This process can continue until we have labeled all the salient features in the traffic.

### 6.3.3 Mapping Visual Attributes

As detail is abstracted away, clutter on the display is reduced. With fewer classes of data, visual attributes such as color become more effective. For example, we can map color to the class labels to produce semantically colored visualizations. Consider Figure 7, an event diagram with source ports arranged vertically and where the flow events are colored by traffic label. Network analysts often ignore source ports to concentrate on the destination ports because the server port is typically indicative of the service being provided. However, once the traffic has been modeled and the labels associated with colors, the source port plot can reveal interesting aspects of program behavior. Note the orange bands of SSH attacks are so intense that they repeatedly cycle over the upper half port space, indicating an attack from a Linux machine. The blue DNS traffic shows web servers that resolve host names for their logs creating long, gently-sloping tails. Scans exhibit a variety of characteristics: when the aggressive programs create a blizzard of connections in a narrow time sequence, scans show up as vertical lines; sequential scans create long slanted lines; and slow scans appear as sparse horizontal lines.

## 7 DISCUSSION

In this section we will focus on three points of discussion, first, performance; second, the usability of the system, and finally the expressiveness of logical models.

The performance of the system is dependent on the amount of data that is being examined. In the dataset of one million events used for evaluation, the system performance was fast enough to be used interactively. Producing and updating visualizations typically take less than 30 seconds, but updating the knowledge base takes approximately 10 minutes.

Visual analysis is effective when patterns are clear. We chose Gantt charts, event diagrams, and scatter plots because they are good at displaying contiguous groupings, which are much easier to select and evaluate than those that overlap or are spread out. We want to incorporate other visualizations and selection techniques to expand the range of patterns the system can model. The effectiveness of this approach will depend on the relationship between the types of patterns that can be shown and the types of patterns that can be modeled.

The interactive model creation process described in this paper appears to work well in practice. However, we have not yet performed controlled user studies. One reason for this is that there exists no obvious comparable way to create logical clauses other than typing in a logical formula. The dynamic selection interface is much faster than using a text box. Nevertheless, it would be useful to precisely measure speed of construction, flexibility,

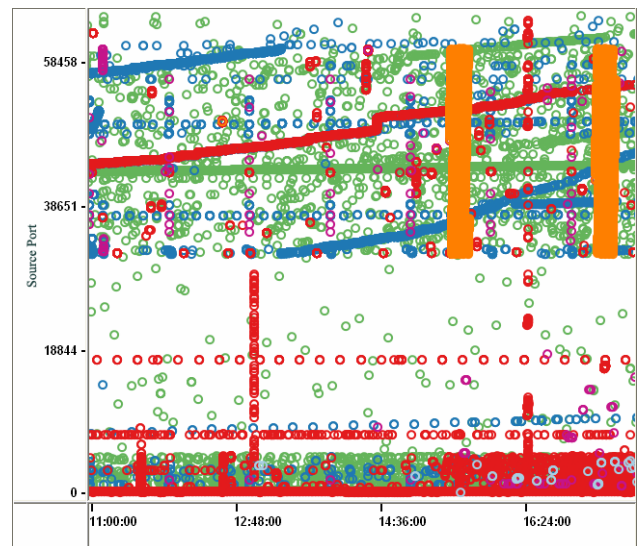


Figure 7. Event diagram showing mail (green), DNS (blue), scan (red), SSH logins (purple), SSH attacks (orange), and chat (indigo)

accuracy and convenience. For these evaluations we need to create standard datasets and perhaps use datasets from more domains than just network traffic analysis.

Behavioral modeling is uncertain; especially since errors are unavoidable in network data collection, but the binary nature of logic does not capture uncertainty. It would be desirable to tag events, patterns and clauses with a confidence value. To do this, we are investigating methods of using probabilistic reasoning. One possibility is Markov Logic Networks, which can be used to perform inference from a first-order logic representation [19].

## 8 CONCLUSIONS AND FUTURE WORK

We have presented a network traffic analysis system that supports the use of previous visual discoveries to enhance visual analysis. In particular, we have shown how analysts using our system can build upon previous visual analysis discoveries to visually explore, and analyze more complex and subtle patterns.

Our plans for future work are fourfold. First, we will perform a study of the accuracy of our classifications. Second, our current implementation supports only a small fraction of visualization techniques, and we would like to extend its visualization capabilities to make patterns more salient. Third, we would like to increase the expressiveness of the EPL, so that the analyst can describe more complex patterns. Finally, we would like to describe uncertainty by extending the knowledge representation to allow for probabilistic reasoning.

### ACKNOWLEDGEMENTS

This work was supported by the Stanford Regional Visualization and Analytics Center (RVAC). This Center is supported by the National Visualization and Analytics Center (NVAC™), a U.S. Department of Homeland Security program operated by the Pacific Northwest National Laboratory (PNNL), a U.S. Department of Energy Office of Science laboratory.

Thanks also to the members of the Stanford Computer Graphics Laboratory for allowing us to collect and analyze their network flow data.

## REFERENCES

- [1] M. Ankerst, M. Ester, H.-P. Kriegel, "Toward an Effective Cooperation of the User and the Computer for Classification" in *Proc. 6th Int'l Conf. Knowledge Discovery and Data Mining (KDD 00)*, ACM Press, 2000, pp. 179-188.
- [2] Arcsight. <http://www.arcsight.com/whitepapers.htm>, 2005
- [3] Argus. <http://www.qosient.com/argus/packets.htm>, 2001
- [4] R. Ball, G. Fink, and C. North. "Home-centric visualization of network traffic for security administration," In *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, 2004
- [5] R.A. Becker, S.G. Eick, and A.R. Wilks. "Visualizing Network Data" In *IEEE Transactions on Visualization and Computer Graphics*, 1995
- [6] S.K. Card, J.D. Mackinlay, and B. Shneiderman. *1999 Readings in Information Visualization: Using Vision to Think*, San Francisco : Morgan Kaufmann Publishers, 1999
- [7] H.B. Enderton. *A Mathematical Introduction to Logic*, New York: Academic Press, 2001
- [8] R. Erbacher. "Visual traffic monitoring and evaluation," In *Proceedings of the Conference on Internet Performance and Control of Network Systems II*, 2001
- [9] M.R. Genesereth, N.J. Nilsson. *Logic foundations of artificial intelligence*. San Francisco: Morgan Kaufmann Publishers, 1987
- [10] J.R. Goodall, W. Lutters, P. Rheingans, and A. Komlodi. "Preserving the Big Picture: Visual Network Traffic Analysis with TNV", in *IEEE Workshop on Visualization for Computer Security*, 2005
- [11] T. Karagiannis, K. Papagiannaki, M.Faloutsos, "BLINC: Multilevel Traffic Classification in the Dark", In *ACM SIGCOMM 2005*, 2005.
- [12] M. Lad, D. Massey, and L. Zhang. "Visualizing Internet Routing Dynamics using Link-Rank", *UCLA Technical Report TR050010*, March 2005
- [13] K. Lakkaraju, R. Bearavolu, A. Slagell, W. Yurcik, and S. North. "Closing-the-Loop in NvisionIP: Integrating Discovery and Search in Security Visualizations," In *IEEE Workshop on Visualization for Computer Security*, 2005
- [14] D. Luckham. *The Power of Events*, San Francisco, Addison-Wesley, 2002
- [15] M. Mansouri-Samani, M. Sloman. "GEM: A Generalized Event Monitoring Language for Distributed Systems". *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4(2):96-108, 1997
- [16] S. McCanne, V. Jacobson "The BSD Packet Filter: A New Architecture for User-level Packet Capture," In *Winter USENIX conference*, 1993.
- [17] T. Munzner, E. Hoffman, K. Claffy, and B. Fenner. "Visualizing the global topology of the Mbone," In *IEEE Symposium on Information Visualization*, 1996
- [18] QILabs. [http://www.q1labs.com/resources/white\\_papers.html](http://www.q1labs.com/resources/white_papers.html), 2005
- [19] M. Richardson, and P. Domingos. "Markov Logic Networks," *Technical Report*, Department of Computer Science and Engineering, University of Washington, Seattle, WA. <http://www.cs.washington.edu/homes/pedrod/mln.pdf>, 2004
- [20] B. Shneiderman. "Dynamic queries for visual information seeking", *IEEE Software* 11, 6 (1994), 70-77
- [21] S.T. Teoh, K.L. Ma, S.F. Wu, D.T. Jankun-Kelly. "Detecting Flaws and Intruders with Visual Data Analysis". In *IEEE Computer Graphics and Applications*, Volume 24, Issue 5, 2004.
- [22] W. Yurcik, K. Lakkaraju, James Barlow, and Jeff Rosendale. "A prototype tool for visual data mining of network traffic for intrusion detection". In *3rd IEEE International Conference on Data Mining (ICDM) Workshop on Data Mining for Computer Security (DMSEC)*, 2003
- [23] D. Zhu, A.S. Sethi. "SEL, A New Event Pattern Specification Language for Event Correlation". In *10th International Conference on Computer Communications and Networks*, 2001