Interactive Traffic Analysis and Visualization with Wisconsin Netpy

Cristian Estan Garret Magin
Computer Sciences Department
University of Wisconsin-Madison
{estan,garret}@cs.wisc.edu

December 6, 2005

Abstract

Monitoring traffic on important links allows network administrators to get insights into how their networks are used or misused. Traffic analysis based on NetFlow records or packet header traces can reveal floods, aggressive worms, large (unauthorized) servers, spam relays, and many other phenomena of interest. Existing tools can plot time series of pre-defined traffic aggregates, or perform (hierarchical) "heavy hitter" analysis of the traffic. Wisconsin Netpy is a software package that goes beyond the capabilities of other existing tools through its support for interactive analysis and novel powerful visualization of the traffic data. Adaptive sampling of flow records ensures that the performance is good enough for interactive use, while the results of the analyses stay close to the results based on exact data. Among the salient features of the package are: hierarchical analyses of source addresses, destination addresses, or applications within aggregates identified by user-defined filters; time series plots that separate the traffic into categories specified with ACL-like syntax at run time; interactive drill-down into analyses of components of the traffic mix; "heatmap" visualization of traffic that describes how two "dimensions" of the traffic relate to each other (e.g. which sources send to which destinations, or which sources use which service, etc.).

1 Introduction

The unrestricted packet communication supported by the Internet offers immense flexibility to the endhosts in how they use the network. This flexibility has enabled the deployment of new applications such as the web long after the IP protocol has been standardized and has contributed significantly to the success of the Internet. On the other hand network operators want to monitor and to some extent control how their networks are used. Firewalls, network address translation, and traffic shaping boxes offer a degree of control that helps keep networks manageable. But even within the constraints of the policies implemented through these devices, the network traffic is very variable and traffic monitoring is necessary. An analysis of network traffic can reveal important usage trends such as the application mix and the identity of the heaviest traffic sources or destinations. Sometimes these analyses can reveal misuses of the network: compromised desktop computers turned into spam relays, remote computers scanning the network for vulnerabilities, network floods directed against a single victim, or caused by a worm trying to spread aggressively. It is often the case that the analysis is urgent because it is carried out to explain a degradation in network service. It is also often the case that the network administrator does not know in advance which ports or IP addresses to focus on and he goes through an iterative process before being able to find convincing evidence for the cause of the problem. Fortunately there are many traffic analysis and visualization tools to assist the network administrator in the task of exploring and understanding the traffic carried by their network. Wisconsin Netpy is a new and powerful addition to this large family.

1.1 Related work

Tobias Oetiker's MRTG [12] is an early traffic visualization tool widely used by network administrators to track the volume of IP traffic based on SNMP counters provided by routers and switches. His RRDtool [12] provides support for visualizing time series plots of arbitrary data and it is actually used by all traffic visualization applications discussed in this section. Jeff Allen's Cricket [1] takes MRTG's idea one step further by allowing the user to track a large number of variables (say the traffic of various links in the network) using a scalable config tree. These tools offer visual information only about the volume of the traffic, but nothing on the composition of the traffic mix.

The NetFlow flow records generated by routers are an information source much richer than the SNMP counters. Toolkits such as OSU flow-tools [7] and SiLK [8] have tools for manipulating and analyzing NetFlow data. Typical analyses allow finding the top sources and destinations of traffic. cflowd [2] is a related package that also supports various types of traffic matrices. Dave Plonka's FlowScan [13] and Cristian Estan's AutoFocus [5] also provide time series of various predefined categories of traffic represented within the traffic mix. The supercomputing community is also interested in IP traffic visualization, and its members have written tools such as"the spinning cube of potential doom" [11] and NVisionIP [10].

AutoFocus and Ryo Kaizaki's aguri [9] employ a novel type of analysis related to top k reports called hierarchical heavy hitters or traffic cluster analysis [6, 3]. This type of analysis is used extensively by Netpy and we describe it in Section 2.3.

Network administrators are often interested in the largest sources or destinations as measured in bytes, packets or flows. One important observation is that while existing tools give the exact traffic for these heavy hitters, the user can often accept small errors. In fact such small errors are already present if one uses sampled NetFlow. Netpy exploits this observation by sampling flow records to speed up traffic analysis and to reduce disk usage. For measuring the traffic in bytes or packets we use the "smart sampling" of flow records introduced by Duffield et al.[4], described in Section 4.

2 Traffic analysis with Netpy

The user can direct Netpy to perform traffic analyses through a graphical user interface or through a console that supports interactive queries as well as scripts. All analyses use an intermediary database of flow records (see Section 3 for more details about Netpy's structure). But what kind of traffic analyses can one perform with Netpy? That's the question we answer through the rest of this section.

2.1 Time series plots with user defined categories

Time series plots are an easy to read visual representation of the traffic. Existing tools such as FlowScan and AutoFocus allow the network administrator to define various traffic categories based on port numbers or network prefixes and have them plotted with separate colors. This way the plots reveal information about the cause of various spikes. Furthermore with separate plots measuring the traffic in bytes, packets, (and flows in a future version) the user will be able to detect not only large floods, but also scans that generate many flows, but not many bytes.

Netpy also supports these types of time series plots. The user can specify the categories using an ACL-like syntax: each rule specifies a source and destination prefix, protocol number and source and destination port range; flows are mapped to the category associated with the first rule they match. For FlowScan and AutoFocus the user needs to specify the categories of interest before the NetFlow data is "imported", whereas with Netpy the user specifies the categories at run time and it is quicker to recompute the plot after the user changes the ACL rules defining the categories because the analysis relies on the database, not on the large NetFlow files.

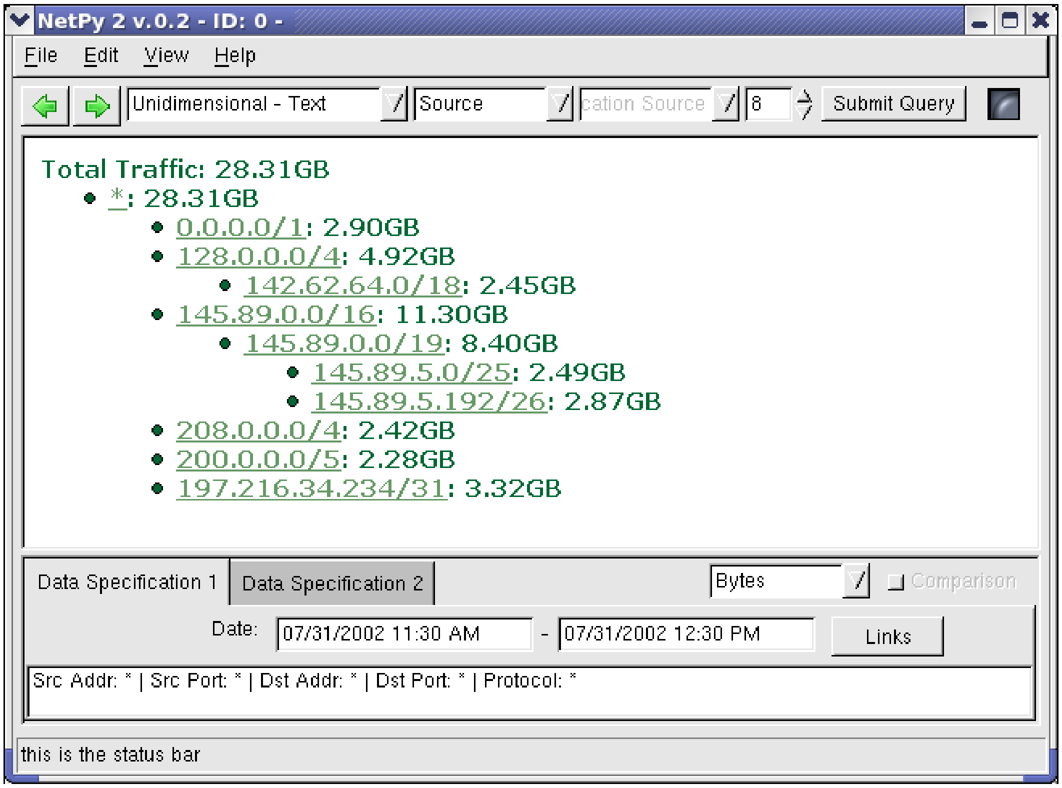

Figure 1: Hierarchical heavy hitter analysis on the sources of the traffic. Indentation is used to highlight prefixes including each other. The analysis finds the appropriate granularity based on the current traffic: 0.0.0.0/1 is half the address space and 197.216.34.234/31 contains just two IP addresses (all addresses are anonymized), but they are both reported because their traffic is above the threshold.

2.2 The scope of traffic analysis

For time series plots, and all the other analyses, one needs to define which NetFlow records constitute the input to the analysis. Existing toolkits often allow the network administrator to configure separate traffic reports for separate links. Netpy separates NetFlow data into different links as data is imported into the database (see Section 3.1 for more details). When running an analysis, the user specifies which links' traffic to work with. The user also specifies the time interval of interest to the analysis.

The user can also specify a filter to apply to the data matching the previous two criteria. The filter consists of one or more rules similar to router ACLs (each rule specifies a source address prefix, destination address prefix, source port range, destination port range, and protocol number) and flow records that don't match any of the rules in the filter are not considered in the analysis. The GUI's interactive drill-down feature works by setting the filter to select only the traffic of interest.

2.3 Hierarchical heavy hitters

The network administrator cannot always know in advance what port numbers or IP prefixes will dominate the traffic, so forcing her to specify in advance

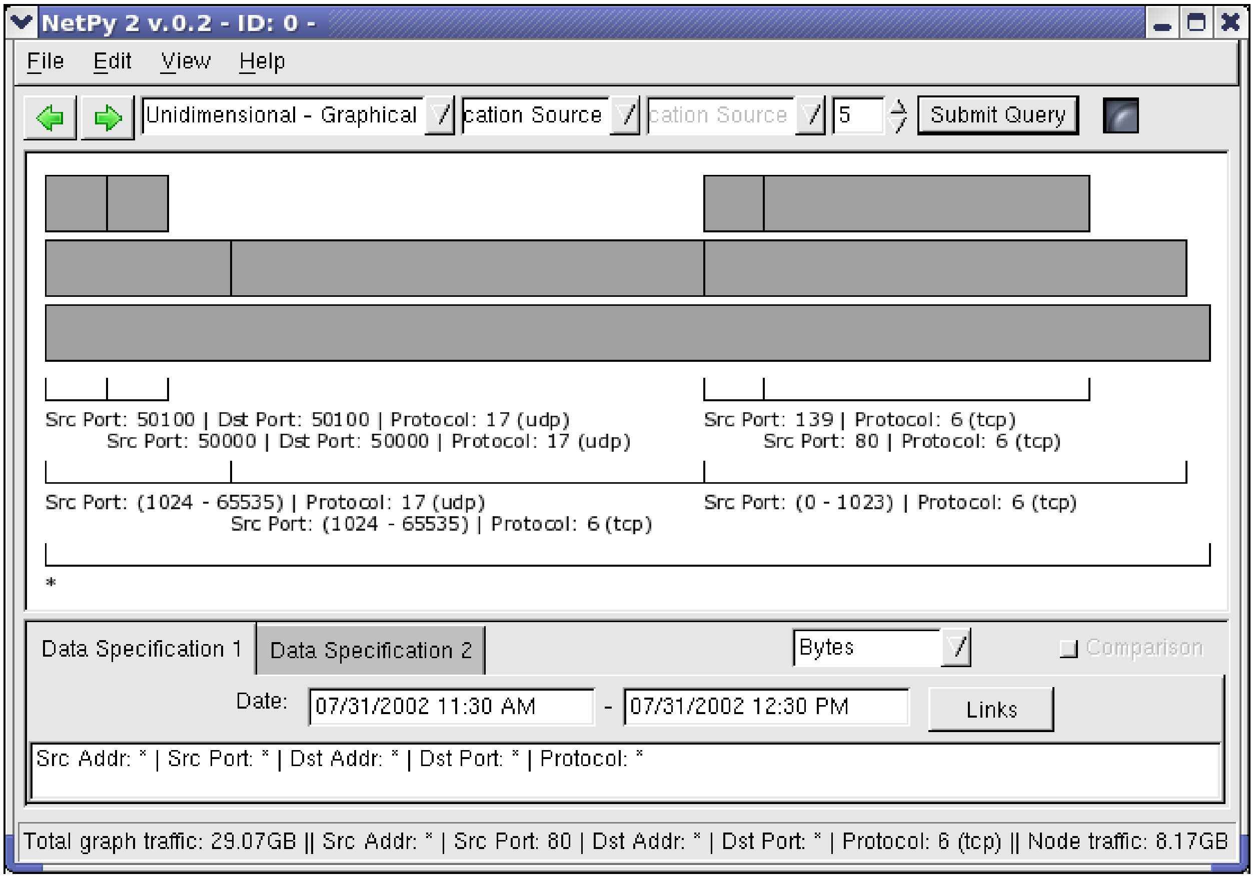

Figure 2: Hierarchical heavy hitter analysis on the application hierarchy. Based on the traffic the analysis picked to report the traffic for high UDP source ports, for high and low TCP source ports, for individual TCP source ports 80 (web) and 139 (Netbios) and two pairs of UDP ports used by a database application.

the ACL rules defining the categories doesn't always work. This is especially true after one drills down into a small, unfamiliar portion of the traffic mix. A traditional solution to this problem is to use "top K reports": one computes the traffic of each source address and reports the top K (say top 20). A related solution is the "heavy hitter report" which reports all sources whose traffic is above a given threshold in the data analyzed (say more than 1% of the total traffic). A problem with both these solutions is that they tell us nothing about sources that send little traffic: if for example we have a prefix with many small sources that nevertheless add up to a large portion of the traffic (a large modem pool), we would want to find out about their behavior. Netpy relies on the "hierarchical heavy hitter" algorithm that finds not just individual addresses, but also prefixes whose traffic is above a certain threshold specified as a percentage of the traffic being analyzed. This algorithm has the property that it not only identifies the prefixes generating significant traffic, but it also automatically finds the right prefix lengths to use when describing various portions of the traffic.

The hierarchical heavy hitter algorithm works as follows: first it reports all individual IP addresses whose traffic is above the threshold, next it aggregates the *remaining* traffic at the /31 level and reports any prefixes that are above the threshold, next it aggregates the remaining traffic at the /30 level and so on until it reaches the root of the IP address hierarchy. The criterion for reporting more general prefixes is that the difference between their traffic and the traffic of more specific prefixes already reported is more than the threshold. However, when Netpy reports such a prefix, it reports its total traffic not the difference between its traffic and that of more specific prefixes. Figure 1 shows the result of a hierarchical heavy hitter analysis on the source addresses in the traffic mix using a threshold of 5%. Through the threshold the user can control the level of detail: with a lower threshold, more prefixes are reported, with a higher threshold the user gets a coarser view.

The same type of hierarchical heavy hitter approach applies to destination IP addresses too. The approach actually generalizes to any hierarchy we can define on one or more of the packet header fields present in the flow record. To capture information about the applications in use, Netpy defines the following hierarchy: the first level below the root divides the flows by protocol, the second level divides the flows by source port into flows originating from low ports (0 to 1023) usually used by servers and high ports (1024 to 65535) usually used by clients, the third level divides the flows by actual source port value and the fourth level divides them by source and destination port value. The analysis will pick the granularity of the results based on the actual traffic. For example if there is a large TCP connection (e.g. a huge backup), the amount of traffic between its source port and destination port will be reported. If there is a source port used by many small connections (e.g. web traffic on port 80), the total traffic coming from port 80 will be reported. If there is no dominant source port, but the source ports used are in the high port range (e.g. traffic coming from a network of typical desktop computers). An example of this type of analysis is shown in Figure 2.

2.4 Bidimensional analysis

The analyses looking at simple hierarchies such as the ones above can tell you that TCP port 80 and UDP port 53 generate a lot of traffic, and they can also tell you that servers A and B generate a lot of traffic, but you won't be able to tell which one is a web server and which one is a DNS server. With Netpy's "unidimensional" reports the user can look at these hierarchies in isolation. Netpy also has "bidimensional" reports that look at two hierarchies at once: Netpy computes the relevant categories for both dimension and reports a crossproduct of the results – for every pair of categories from the opposite hierarchies, the traffic matching both categories is reported. For the example above, if we run a bidimensional analysis on the application and source address dimensions, the application dimension will pick (protocol=TCP,source port=80) and (protocol=UDP,source port=53) as relevant categories and the source address will pick (source address=A) and (source address=B). The bidimensional report will have the traffic of the following four combined categories of traffic: (protocol=TCP,source port=80,source address=A), (protocol=TCP,source port=80,source address=B), (protocol=UDP,source port=53,source address=A), and (protocol=UDP,source port=53,source address=B).

In the GUI, the two dimensions of a bidimensional report are the two sides of a square and the categories defined within individual dimensions are represented as small segments on the sides of the square. The rectangles within the square represent the combined categories. The darkness of the rectangles indicates the amount of traffic of the combined category with darker shades indicating more traffic. The GUI displays the actual amount of traffic in any combined category when the user moves the mouse over the corresponding rectangle. Using darkness to convey the intensity of traffic (or other data) is known as the "heatmap" representation. Figure 3 shows a bidimensional analysis with the application hierarchy as the horizontal dimension and the source IP address hierarchy as the vertical dimension.

The bidimensional reports do not capture all the information present in the (textual) multidimensional reports used by AutoFocus that consider all five

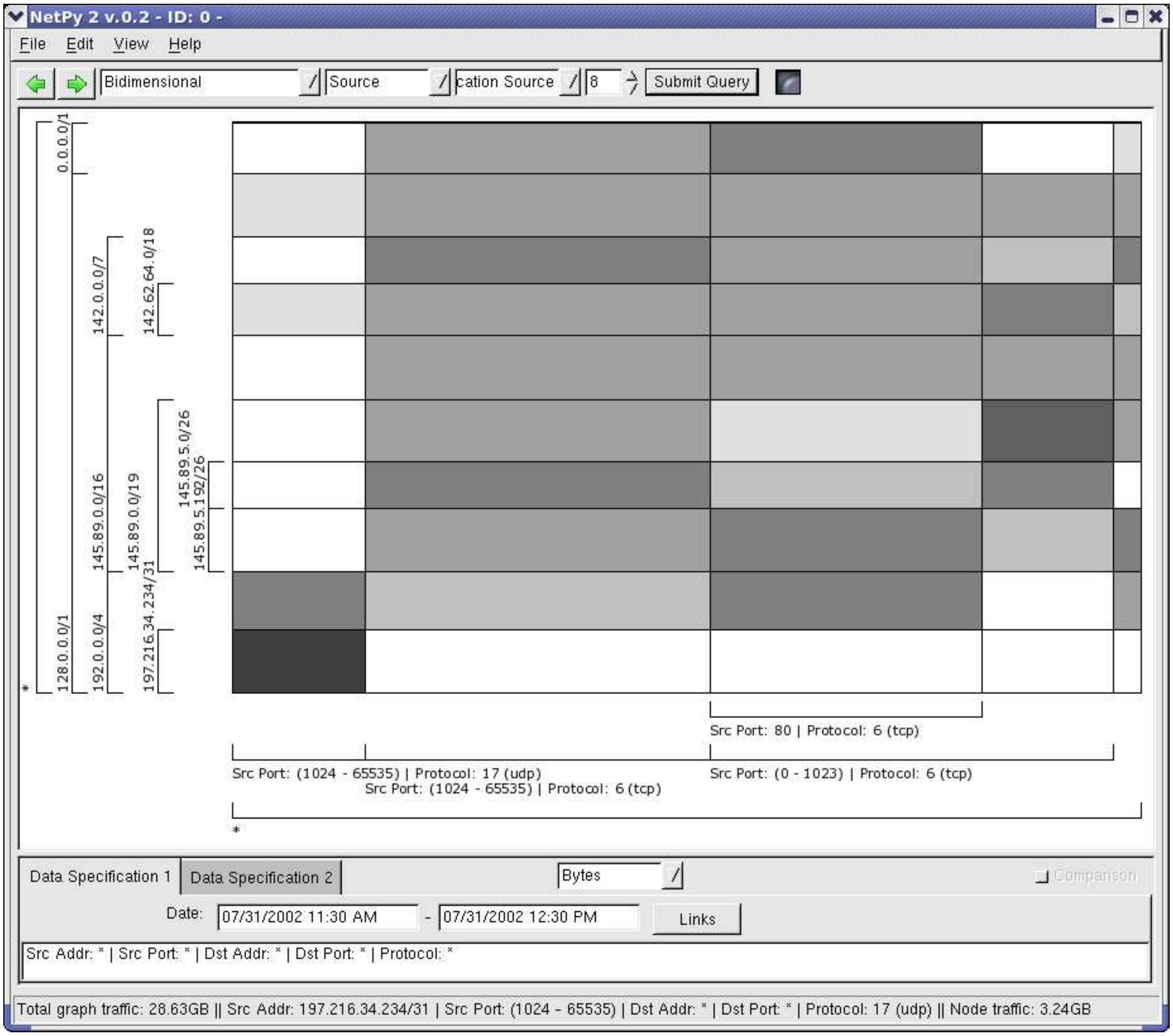

Figure 3: The bidimensional Hierarchical heavy hitter analyses on the application hierarchy and source IP address hierarchy shows more details than the two unidimensional analyses. Note how this shows that 197.216.34.234/31 sends mostly UDP traffic from high ports and that 192.0.0.0/4 sends no TCP traffic from low ports other than port 80 (all IP addresses are anonymized).

packet header fields at the same time. The advantage of these bidimensional reports is that they can be computed much faster than the multidimensional reports and yet they can convey much of the information present in a multidimensional report.

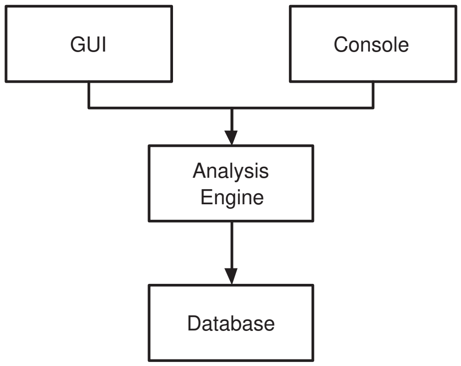

Figure 4: Netpy program structure.

3 Structure of Netpy

Netpy has four main parts: the database, the analysis engine, the console and the GUI. The role of the database is to store preprocessed NetFlow records and deliver the records selected for the current analysis to the engine. The analysis engine runs the hierarchical heavy hitter algorithm, and all other analysis algorithms supported by Netpy. The console is a text based interface to the analysis engine and the only interface that allows the network administrator to update the database. The GUI is an interface that visualizes the traffic analysis results and helps the user navigate the traffic data.

3.1 The database

The Netpy database is an intermediary representation of the NetFlow flow records. The aim of the database is to preprocess the flow records in a way that ensures that when the user asks for an analysis, one has to read from disk the minimum amount of data needed to compute the result. Quick analyses are important for interactive exploration of the traffic mix. The entire database manipulation code is written in C and it links against the flow-tools library.

Field name	Match
Exporter address	prefix match
Engine type	exact match
Engine ID	exact match
Source address	prefix match
Destination address	prefix match
Next hop addr.	prefix match
Input interface	exact match
Output interface	exact match

Table 1: In the links.conf configuration file the user can define any number of links. The file has a list of rules with the NetFlow fields in this table. Each flow record is mapped to the link associated with the first rule it matches. For all fields, a '*' in the rule matches all possible values.

3.1.1 Database structure

The Netpy database is actually a hierarchy of files with simplified flow records. This format has the following four main advantages over just storing NetFlow records directly: data reduction and better control over disk usage through adaptive sampling when there are too many flow records; storing flow records for different links in different files; using separate files for time bins that make it easier to only read in the records of flows active during the selected time interval; more compact flow records with fewer fields.

Netpy groups flow records into “links” based on the links.conf configuration file that uses the fields from Table 1 to select the link a flow record belongs to. This allows the network administrator to separate traffic carried on various links of a router that is nevertheless reported together. The flow records corresponding to different links are then stored in separate directories. Thus when the user runs an analysis on only one of the links, we don’t have to go through all flow records, but only read those mapping to that link.

The links.conf file also specifies a cap for each link on how much hard disk space an hour’s worth of traffic can take. When the number of flow records exceeds the allotted space, Netpy applies sampling using a rate that ensures that the disk usage stays within budget. The smaller the disk usage allowed, the more aggressive the sampling has to be and the less accurate the results of the analyses will be. See Section 4 for results on the amount of error introduced by sampling.

Each directory corresponding to a link contains individual files with flow records, each representing a 5 minute time bin. For the current version of Netpy, this does limit the time intervals. The user can only request analyses on multiples of 5 minutes, but once the user specifies the interval, we can read in the right flow records by just reading the files representing the 5 minute bins included in the interval. Some of the original NetFlow flow records can span two or more bins. We handle these by splitting them and storing the resulting records in their respective bins. This splitting of records results in an increase of only 2% in the number of records stored in the database which is a price worth paying.

The flow records in the database contain only the fields used in the analyses: source and destination IP address, protocol, source and destination port and byte and packet count. We need not store timestamps because the file a flow record is in identifies which 5 minute time bin it belongs to. We also discard most of he fields from Table 1 because the useful information they hold has already been incorporated in the choice of the link the flow record is mapped to. This way the size of a flow record is reduced from 60 bytes to 21.

3.1.2 Reading from the database

The analysis engine specifies what data to select for the analysis. This specification has four parts: the list of links to include, the time interval, a filter, and whether the analysis counts bytes, or packets. The first two parts of the specification determine which database files are read. As the files are read in, all records are compared against the filter and the ones not matching any rule are ignored. The records at this point represent the traffic the analysis will be run on, but the database performs two more operations to help the analysis engine: it samples and sorts the data.

Analyses that cover a large time interval and don't specify very selective filters can read millions of flow records from the database. Given that the analysis engine implements complex algorithms in python, it runs slow on this many records. Before passing the results to the analysis engine, we apply the same adaptive sampling algorithms used when writing the database to ensure that the number of flow records passed to the analysis engine is not very large (no more than 100,000 in the current version). The database also sorts the records by the field used in the analysis.

3.2 The analysis engine

The current version of Netpy has an analysis engine implemented entirely in python[1]. It runs the hierarchical heavy hitter algorithms and all other algorithms doing the analysis of the traffic. Analyses can complete in under five seconds or take as long as a minute. Running a destination address hierarchy analysis for an hour's worth of traffic takes $5.3s$ to complete. Running a application hierarchy analysis on the same time interval takes $2.7s$. On a 24 hour time period the analysis takes $34.7s$ and $27.0s$, for address and application analysis, respectively. We plan to reimplement most analysis algorithms in C and we expect a speedup by at least a factor of 100. It happens quite often that the user asks for the same analysis again, for example by pushing the "back" button in the interface. To avoid accessing the database and doing the computations again, the analysis engine keeps a cache of analysis results and it reads the results of old analyses out of this cache. The size of the cache is modest because the results of the analyses are typically quite small. The entries in the cache are gzipped binary dumps of the analysis data structures, the average files size is $1KB$.

The database and the analysis engine are normally part of the same process as the GUI or the console. It is also possible to run the GUI remotely and in this case the analysis engine runs as a daemon on the machine with the database.

[1] Netpy's name comes from the fact that it does network traffic analysis in python.

3.3 The console and the graphical user interface

The console and the graphical user interface, both implemented in python, are the two interfaces to Netpy. The console supports a simple language of commands for updating the database and performing analyses. It is the only interface that allows the user to add new NetFlow data to the database and delete old flow records. The console can accept commands interactively or as a script. The GUI, built using the wxPython user interface toolkit, visualizes the traffic analysis results and helps the user navigate the traffic data. Drill-down through clicks on graphical elements representing IP address prefixes or port ranges is integrated with filters. The "back" and "forward" buttons further facilitate the exploration of the traffic mix.

4 Sampling algorithms used by Netpy

There are two measures of traffic that Netpy analyses can choose to compute: the number of bytes, and the number of packets within various categories of traffic that make up the traffic mix. The aim of the sampling algorithms is to take a large number of flow records and reduce it to a smaller sample that is an unbiased, low error representation of the original traffic. By sampling we fundamentally loose information, so it is unavoidable that there will be errors when we estimate the traffic of some categories of traffic, and categories with little traffic are especially vulnerable.

Our aim is to pick a sampling function that ensures that the sampling error is small for the large categories of traffic. Let's focus on byte counts first. A simple solution is to sample each flow record with probability p, and for all sampled flow records to multiply their byte counts by $1/p$ to compensate for the flow records that were not selected in the sample. For example if $p = 1/5$ we would multiply by 5 the byte counts of all the sampled records. This method ensures that the number of flow records is reduced by approximately a factor of p. The errors introduced by this method are not very high if the sizes of the flows are close, but if there are a few very large flows the errors can be significant. Say the traffic mix consists of $1,000$ flows of $10KB$ each and one flow of $10MB$, and thus the actual total traffic is $20MB$. The sample will contain around 200 of the small flows, each counted with $50KB$ of traffic so their contribution will be estimated correctly at around $10MB$, but the situation is different with the large flow: if it doesn't get sampled (and this has a probability of 80%), we don't count it at all, if it gets sampled, we count it as $50MB$. Thus we either underestimate the total traffic by a factor of $(10+0)/20 = 0.5$, or we overestimate it by a factor of $(10 + 50)/20 = 3$.

The solution to this problem is to use size dependent sampling, also known as smart sampling which was proposed by Duffield et al.[4]. This methods picks the sampling probability in a way that favors the large flows. More exactly the algorithm picks a threshold z, and the flows with size $s \geq z$ are kept in the sample while the ones with $s < z$ are kept with probability $p_s = s/z$. If one of these small flows is sampled, its byte count is multiplied by $1/p_s = z/s$ which gives us a byte count of $s \cdot z/s = z$.

Smart sampling has the property that if the original set of flow records has a total traffic of T, the expected number of flow records after sampling is at most T/z, irrespective of how many flow records the original set has, and how their sizes are distributed. It also has the property that if the total traffic of a category is C the standard deviation (average error) of the estimate of the traffic of the category after sampling is at most $\sqrt{Cz}$, but it can be smaller depending on the distribution of the sizes of the flows that are part of the category. For example if the total traffic is $T = 100,000MB$, and we use a threshold of $z = 1MB$, the number of flow records in the sample is expected to be at most $100,000$ (if the original traffic mix has many flows significantly larger than $1MB$, the number of flow records in the sample will be significantly below $100,000$). If we want to estimate the total traffic T based on the sample, the standard deviation of the result will be at most $\sqrt{Tz} = 316MB$ and the the probability that we overestimate or underestimate the total traffic of $100,000MB$ by more than $3\sqrt{Tz} = 948MB$ (an error of less than 1%) is below 0.3%. If we look at a smaller category with a traffic of $C = 10,000MB$, the probability that we underestimate or overestimate its traffic by more than $3\sqrt{Cz} = 300MB$ (an error of 3%) is below 0.3%. The larger z, the smaller the sample size, the larger the errors. If we increase z by a factor of 100, we reduce the sample size by a factor of 100, but we increase the errors by a factor of 10.

When adding data to the database Netpy uses the limit imposed on the disk usage of the database to indirectly determine the threshold z. For example if the limit for one hour's data is set to $10MB$ in links.conf, this translates to $853KB$ for each of the 12 files representing a 5 minute bin, and since the size of a flow record is 21 bytes this translates to $41,600$ records. Thus the value of z will be at most one 41,600th of the total traffic for the 5 minutes. This translates to a standard error for the estimate of the total traffic during those five minutes of at most $\sqrt{1/41,600} = 0.49\%$. If we look at a smaller category of say 1% of the traffic during those five minutes the standard error of the estimate of the components traffic is at most 4.9%. Of course, if one looks at longer time periods (an hour, or a day) since there will be more flow records, the relative errors in the estimates will go down.

The sampled flow records used for estimating byte counts can also be used for estimating packet counts. Since the sampling is biased towards flows with many bytes it will also catch flows with many packets. Packet sizes vary between 40 bytes and 1500 bytes so it will happen that a flow with fewer larger packets will be preferred over a flow with more but smaller packets, but since the ratio between the largest and the smallest packet is only 37.5 the types of pathological errors that are possible with uniform record sampling are not possible. Let s' be the number of packets in a flow record. If $s \geq z$, the flow record will be sampled and s and s' will remain unchanged. If $s < z$ and the flow record is sampled we multiply the packet count by $1/p_s$ and thus have a packet count of $s' \cdot z/s$ in the flow record we keep in the sample. The errors in the estimates for the number of packets in various categories of traffic are similar to the errors in the byte counts.

5 Future work

While Netpy is the result of a lengthy design and development process, we plan to improve it further. A first thing to do is to add flow counting functionality (partially implemented) because flow counts are better at revealing many types of traffic a security-conscious network administrator might want to know about such as scans and floods with source addresses spoofed at radom. We can group the improvements we plan into improvements that will increase the speed of the analyses, and improvements that will increase their power.

The current performance bottleneck is the hierarchical heavy hitter algorithms in the analysis engine. We plan to reimplement all hierarchical heavy hitter algorithms in C and based on measurements of AutoFocus' C backend that runs similar algorithms we expect a speedup by at least a factor of 100. The database read can also become a bottleneck when reading large amounts of data (e.g. running an analysis on an entire month). We plan to address this performance bottleneck by conceptually keeping multiple versions of the database a very small one with very aggressive sampling, a medium one and a large one with mild sampling. An analysis on long time intervals would use the coarsest database to reduce the amount of disk reads, while one on short time scales would use the most detailed one to get accurate results. In practice we can integrate these multiple conceptual databases into a single file hierarchy. More compact encodings of flow records or the use of gzip to compress the flow record files will reduce disk usage and increase performance since less data will have to be read.

There are a few directions in which we plan to increase the power of Netpy's analyses. Due to the 5 minutes bins used currently the analysis cannot look at a granularity finer than 5 minutes, even though the original NetFlow data would have allowed it. By adding small timestamps to the flow records we hope to be able to support analysis at the granularity of seconds. Another direction of improvement relies on the observation that while currently all analyses work on a portion of the traffic mix, it often makes sense to compare the current traffic against historical traffic to find the things that have changed. We plan to extend Netpy with "comparison reports" working on two data sets at a time. A third direction plans to address limitations due to the fact that the analysis of IP addresses relies on the implicit hierarchy in the IP address space. The problem with this approach is that because of how the IP address space is allocated, prefixes are not always meaningful, they can include portions of unrelated organizations. We plan to extend Netpy with three more hierarchies for IP addresses: one based on DNS reverse mappings of IP addresses, one based on whois data, and one based on BGP routing table information. Hierarchical heavy hitter algorithms can be easily adapted to all of these hierarchies and they can provide valuable new insights into the traffic mixes on our networks.

Our hope is that many of these improvements will be implemented and mature enough for widespread use by the end of the 2005.

6 Conclusions

IP traffic can be unpredictable and traffic analysis can help with incident response as well as with long term planning of network growth. This paper presents Wisconsin Netpy, an application for analyzing and visualizing NetFlow traffic data. While Netpy is certainly not the first application in this space, we believe that it incorporates important new ideas that enable powerful exploratory analyses of the traffic mix not supported by other tools currently used by network administrators. The use of a small database of sampled traffic enables prompt analyses that allow the network administrators to refine their queries iteratively. The unidimensional and the novel bidimensional hierarchical heavy hitter analyses can provide a detailed view of the traffic. The visualization of analysis results in the form of time series plots and heatmaps makes it easier for a human observer to absorb information about the composition of the traffic mix. We hope that Netpy will contribute to a better understanding of how networks are used and through this understanding to better managed networks that offer more reliable service to millions of Internet users worldwide.

7 Acknowledgments

We thank John Henry, Fred Moore, Jaeyoung Yoon, Brian Hackbarth, Ryan Horrisberger, Pratap Ramamurthy, Dan Wendorf, Steve Myers, and Dhruv Bhoot who were part of the teams that worked on Netpy as a class project in Fall 2004 and 2005. Early conversations with Glenn Fink and Chris North at Virginia Tech lead to the use of heatmaps as visualization metaphor. We thank Mike Hunter for suggestions for features in Netpy and Dave Plonka for providing us with generous amounts of NetFlow data to test on and valuable feedback.

Author Information

Cristian Estan graduated from the Technical University of Cluj-Napoca, Romania in 1995 with a degree in Computer Science. His first real job, was as network/system administrator, and it taught him that configuring software or networking gear always takes longer than expected. After moving to the U.S. in 1998 he worked at two startups and eventually managed to get a Ph.D. at U.C. San Diego. Currently he is an assistant professor at U. W. Madison and can be reached at estan@cs.wisc.edu.

Garret Magin graduated from the University of Wisconsin Madison in May of 2005 with degrees in computer science, computer engineering, and math. He recently started working in the embedded networking space on the Windows CE core networking team. When he is not at work and its not raining he is off riding his Honda Superhawk. He can be reached at garret.magin@microsoft.com.

References

[1] Jeff R. Allen. Driving by the rear-view mirror: Managing a network with cricket. In *USENIX 1st Conference on Network Administration*, April 1999.

[2] cflowd: Traffic flow analysis tool. `http://www.caida.org/ tools/measurement/ cflowd/`.

[3] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in data streams. In *VLDB*, December 2003.

[4] Nick Duffield, Carsten Lund, and Mikkel Thorup. Charging from sampled network usage. In *SIGCOMM Internet Measurement Workshop*, November 2001.

[5] Cristian Estan. Autofocus: A tool for automatic traffic analysis. In *29th meeting of NANOG*, October 2003.

[6] Cristian Estan, Stefan Savage, and George Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proceedings of the ACM SIGCOMM*, August 2003.

[7] Mark Fullmer and Steve Roming. The osu flow-tools package and cisco netflow logs. In *USENIX LISA*, December 2000.

[8] Carrie Gates, Michael Collins, Michael Duggan, Andrew Kompanek, and Mark Thomas. More netflow tools for performance and security. In *USENIX LISA*, November 2004.

[9] Ryo Kaizaki. Aguri: An aggregation-based traffic profiler. `http://www.csl.sony.co.jp/person/kjc/kjc/ software.html#aguri`.

[10] Kiran Lakkaraju, William Yurcik, and Adam J. Lee. Nvisionip: Netflow visualizations of system state for security situational awareness. In *ACM VizSEC/DMSEC04*, October 2004.

[11] Stephen Lau. The spinning cube of potential doom. In *Communications of the ACM*, volume 47, June 2004.

[12] Tobias Oetiker. Mrtg - the multi router traffic grapher. In *USENIX LISA*, December 1998.

[13] David Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *USENIX LISA*, pages 305–317, December 2000.