CS 640 project for Fall 2005

Cristian Estan

September 13, 2005

1 Introduction

As project, you will extend the application developed last Fall by CS 640 students: Wisconsin Netpy. It is an Internet traffic analysis and visualization application using NetFlow data collected by routers. This description of the project assumes that there will be four teams working on different parts of the project. Currently Netpy is divided into three main components: the database, the analysis engine, and the user interface (includes the GUI and the console). Three of the teams will take ownership of these components and the fourth one will be working on extensions to the traffic analysis functionality that will be implemented as plugins to the main application. Table 1 summarizes the features to implement, sorted by priority. How many of these features will get implemented depends on how many of you will work on the project. The lower priority features are more likely to be cut. I hope that you will come up with some ideas of features you would like to add.

The bulk of the work on this project will be to add new features to the application. These features can be grouped into two broad categories: features that allow the user to perform more powerful analyses of the traffic (presented in Section 2), and features that improve the usability of the software (presented in Section 3).

1.1 Correctness of results

One "feature" not explicitly discussed in the following sections is the maturity of your code. You are expected to make a reasonable effort to ensure that there are no serious bugs in your code. The most important thing is to ensure that the results of the analyses are correct. If faced with a choice between implementing one more feature and checking the correctness of existing code, go for the correctness.

2 More powerful analysis

Roughly half of your work will be on features that allow the user to perform analyses that give him more useful details on the composition of the traffic mix. These should facilitate a better understanding of the traffic mix and lead to better decisions on the network administrator's part.

2.1 New hierarchies for analysis

Netpy's analysis is built around the concept of "dimensions of analysis" such as source address, destination address, or application. For each dimension, the simplest analysis just maps the traffic into disjoint groups (e.g. grouping the traffic by the source address of the packets) and reports those above a threshold (by default 5% of

Feature, number of section describing it	Importance	Amount of work for each team			
		Database	Analysis	Interface	Plugin
Faster database 3.1.2	High	Large	None	None	None
Timestamps in database 2.5	High	Medium	None	None	None
Sampled NetFlow 3.5.1	High	Small	None	None	None
Faster analysis engine 3.1.1	High	None	Large	None	None
Plug-in hierarchy support 2.1.2	High	Small	Large	Large	None
Comparison reports 2.3	High	None	Large	Medium	None
Address and port helper 3.3.1	High	None	None	Large	None
Strengthening filters 2.2.1	High	Small	None*	Medium	None
User defined categories 2.1.3	High	None	None	None	Large
DNS IP hierarchy 2.1.4	High	None	None	None	Large
Packaging 3.4	High	Medium			
Flow counts 2.4.1	Medium	Medium	None*	Small	None
SYN counts 2.4.2	Medium	Medium	None*	Small	None
Accuracy feedback 3.3.3	Medium	Small	None*	Small	None
Accuracy selection 3.3.4	Medium	Small	None*	Small	None
Automatic time selection 3.2.1	Medium	Small	None*	Small	None
Dest. port hierarchy 2.1.1	Medium	None	Small	None*	None
Navigation shortcuts 3.2.2	Medium	None	None	Medium	None
Consistent metaphors 3.3.2	Medium	None	None	Medium	None
Filter drill-down work 3.2.3	Medium	None	None	Medium	None
Precomputation 3.1.3	Medium	Small	None	Medium	None
WHOIS IP hierarchy 2.1.5	Medium	None	None	None	Large
Undo for additions to db. 3.6.2	Low	Large	None	None	None
Packet header traces 3.5.2	Low	Small	None	None	None
Database size management 3.6.1	Low	Small	None	None	Small
Multiview navigation 3.2.4	Low	None	None	Large	None
Better plots 3.3.5	Low	None	None	Medium	None
Filter extensions 2.2.2	Low	Small	None	Medium	Small
BGP IP hierarchy 2.1.6	Low	None	None	None	Large

Table 1: Features are sorted by importance and within importance by which group would need to do most work on a given feature. The last 4 columns represent an estimate of the amount of work each group would have to do (None* means less than 5 lines of code). All the planning of interfaces has to accommodate all the features, but implementation should be incremental: we will cut the features there is no time for. Any team can do the packaging, we will decide later.

the traffic). Netpy associates with each dimension hierarchies that map the traffic into smaller and smaller groups. At the root of each hierarchy is a group that contains all traffic.

Adding more dimensions of analysis (and thus more hierarchies) to Netpy gives the user the power to see different types of details about the traffic mix. Netpy currently uses three hierarchies. You will add up to five new hierarchies to Netpy, one directly, and the rest as "plugins" the user can decide to use or not. All the types of analyses supported by Netpy (time series, unidimensional, bidimensional, and the comparison reports described in Section 2.3) should work with all of these hierarchies (or arbitrary pairs of hierarchies for bidimensional reports).

2.1.1 Destination port based hierarchy

The existing application hierarchy can detect individual source ports sending much traffic, but not destination ports receiving a lot of traffic. The reason for this is that traffic is grouped by source port at higher levels of the hierarchy. To correct this shortcoming, you should implement a second application hierarchy in which you reverse the role of source and destination ports: destination ports will influence the higher levels of the hierarchy and source ports only the last.

2.1.2 Support for plug-in hierarchies

One of the main additions to Netpy will be the support for extensions via plugin traffic hierarchies. You will have to design and implement an interface Netpy will expose to these plugins. The fourth team (a.k.a. the plugin team) will use this interface extensively. The other teams will have to implement the functions exposed by this interface.

The addition of plugins (and their options) will be controlled by a configuration file on the Netpy server. The details of this file should be visible from the user interface, but the user interface should not support changes to this file. Each plugin will have multiple instances (e.g. we can group both the source and destination addresses based on DNS names). Each plugin instance can have up to one configuration file (e.g. a list of user defined traffic categories) and local data (e.g. a local database of IP address to DNS name mappings). Each plugin will interface with all three components of Netpy.

Plugins should be able to register some type of callback function with the database so that they can update their local data when new flow records are added to the database. For example the DNS plugin could perform reverse DNS lookups to find the names of new IP addresses that are added to the database. Also plugins should provide a function that allows the database management code to ask them to remove old local data. For example mappings for IP addresses that have been removed from the database (or mappings that expired) should be removed when the database management component is activated.

The analysis engine will still perform the hierarchical heavy hitter analyses on the plugin hierarchies. The plugins need to implement two functions to assist the analysis engine: one mapping flows to groups in the hierarchy, and another one for sorting groups. The mapping function will take as an input a flow record identifier (source and destination IP address and port, plus protocol number) and it will map it to a list of groups the flow belongs to from most general (all traffic) to most specific. These lists will be the internal encoding of where the flow is within the hierarchy. The analysis en-

gine will use this internal encoding to perform the analysis. The number of groups flows map to does not have to be the same for all flows. For example www.google.com would map to the following list of 4 groups: all traffic, traffic from .com addresses, traffic from google.com addresses and traffic from www.google.com. But www.cs.wisc.edu would map to a list of five. The second function the plugin has to provide would receive a list of such encodings present in the result of a query and sort them according to whatever order makes sense for the semantics of the hierarchy. This order will be used when displaying the results.

The plugin will have to provide two functions to the user interface. The first function would map the internal encoding to a human readable string. The second function would map the internal encoding to a human readable string of limited length (say 15 characters).

2.1.3 User defined categories

A first plugin would allow the users to explicitly define the categories of traffic that define the hierarchy used for analysis. The configuration file for this type of plugin will contain a list of ACL-like rules (same syntax as the filter rules of Netpy with the extensions from section 2.2.1), each mapping the matching traffic to a user defined category. Each flow belongs to the category that goes with the first rule it matches in the categories file. These categories can be hierarchical: a rule can map the traffic to the "email" category, other rules can map traffic to the "mail/SMTP" or "mail/IMAP" subcategories. This plugin will need no local data to perform its function.

2.1.4 DNS based IP address hierarchy

The DNS based IP address hierarchy will use reverse DNS lookup to map IP addresses in the database to DNS names. Individual lines in the analyses will report traffic sent to (or received from) IP addresses mapping to a DNS name, or many DNS names within the same domain. To differentiate between a single name and domains containing multiple names, domains representing multiple DNS names should be represented with a leading "*". For example "*.cs.wisc.edu" would represent various computers in the cs.wisc.edu domain (e.g. www.cs.wisc.edu, mail.cs.wisc.edu, ogre.cs.wisc.edu, etc.) as opposed to "cs.wisc.edu" which represents a computer called cs.wisc.edu. You should build a local cache of IP address to DNS name mappings. The cache should also store negative results (lookup unsuccessful). DNS mappings of IP addresses can change with time. Therefore, if an IP address has an old entry in the cache and it appears in the traffic mix again, it should be looked up again. The time after which an address is looked up again should be user configurable and it should default to one week. The user should be able to instruct the plugin to perform DNS lookups only when data is added to the database, only when the analysis engine queries the database, or never (in this case only cached mappings are used). Also the user should be able to rate limit the lookups to avoid generating large spikes in DNS traffic.

2.1.5 WHOIS based address hierarchy

Some prefixes can be meaningfully mapped to various organizations in charge of them. Often an organization has multiple non-contiguous prefixes. The relationship between organizations

is often hierarchical: the department received its address ranges from the campus, the campus from the ISP, the ISP from a registrar (say ARIN). A simplified version of this plugin can allow the user to enter mappings from prefixes to organizations in a text file local to the plugin. A full version should use the whois service to query the registrars' databases. These plugins should offer the same type of control as the DNS plugin over when the lookups are performed and what the maximum rate for lookups is. Note that when the whois database (or the configuration file) contains multiple prefixes matching an IP address, the most specific (longest) matching prefix should be used and the others ignored.

2.1.6 BGP based IP address hierarchy

BGP routing tables can be used to map prefixes to autonomous system numbers. This plugin would use BGP routing tables as input instead of WHOIS data. In many other respects it would be similar.

2.2 Stronger filters

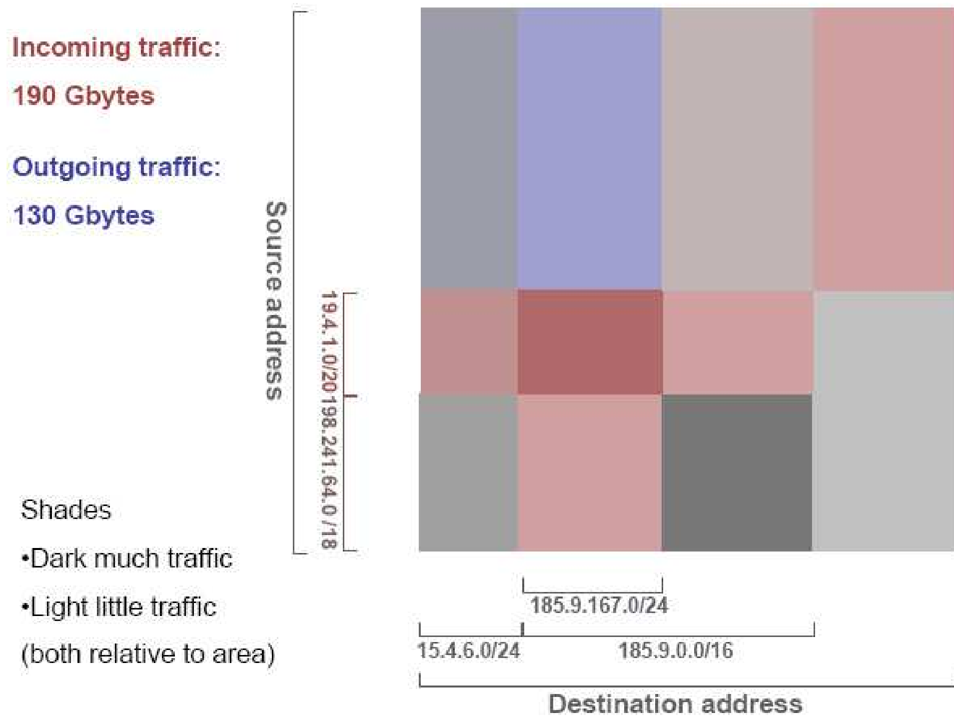
It is essential for incident response that the network administrators be able to "drill-down" into a part of the traffic mix of interest to them. Netpy's filters are the tool that allows them to do that. Extending the semantics of these filters increases their expressive power.

2.2.1 Using existing filter fields

Currently filter rules look at 5 fields source address, source port, destination address, destination port and protocol to decide whether the rule matches a given flow record or not. For the two IP addresses, the rule can specify prefixes they have to be within, for the port numbers ranges, and for the protocol an individual value it has to match. The rule can also ignore any of the fields. By extending what rules can do with the individual fields, we can enhance their expressiveness. Adding the "not" operator to address and port fields will allow the user to express rules that match addresses not in the given prefix and ports not in the given range. Adding a global negation operator to the rule will "reverse" its semantics: the rule will match the flows it would have ignored and ignore the ones it would have matched without the global negation. Also for IP addresses the user should be able to specify IP address ranges in addition to using prefixes.

2.2.2 Extensions to new hierarchies

Filters can be further strengthened by adding additional methods of filtering to the rules. One possibility is to allow the use of the hierarchies implemented by plugins for filtering (this would require extending the API for plugins). This would allow the user to select with a single rule the flows mapping to one of the user defined categories, those whose source addresses are in a given DNS domain, or whose destination addresses map to a given ISP (based on whois data). There is another useful extension that doesn't directly correspond to any of the plugins discussed here: selecting flows whose source or destination IP address is in an explicitly enumerated set. An "IP set" would consist of a file with individual IP addresses, prefixes, or address ranges generated by a third party application and used by Netpy for filtering. Using such an IP set would allow us to focus the analysis for example on the computers infected with a certain worm if a third party application could provide us with a list of the IP addresses of computers (suspected to be) infected.



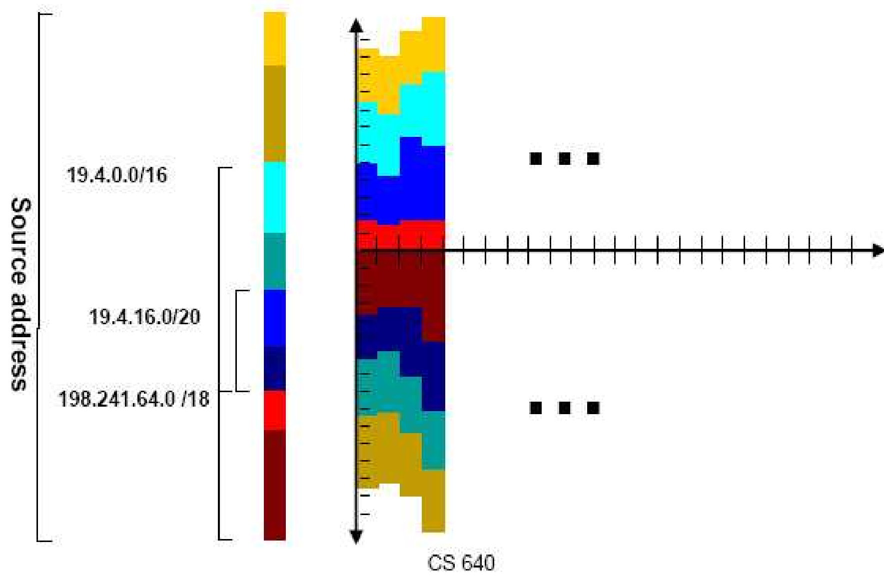


Figure 1: Mock-ups of comparison reports for the bidimensional and time series case.

2.3 Comparison reports

An important feature only partially implemented by Netpy is comparison reports. These allow the user to compare any two portions of the traffic mix: one link's traffic against the other's, today's traffic against yesterday's (or last week's), the traffic of one prefix against the traffic of the other, the distribution of source addresses when we measure the traffic in bytes versus when we measure it in packets, etc. Comparison reports apply to all three types of reports supported by Netpy: unidimensional reports, bidimensional reports and time series plots. The basic idea is that we take two "data sets" and we analyze them together with respect to the same dimension. It is easiest to do this for time series plots: we first perform the hierarchical heavy hitter analysis on the sum of the two datasets to obtain the groups along the analysis dimension to divide the traffic into, next we just plot the amount of traffic in each group for the first data set above the x axis and for the second data set below the x axis. For unidimensional and bidimensional reports we generate a report for the sum of the data sets and use color to distinguish between the contributions of the two sets. If we use red for the first set and blue for the second, groups where both sets contribute equally would be represented with a neutral gray (the darkness of this gray would indicate traffic volume for bidimensional reports), the groups where the first data set dominates would be reddish (the stronger the dominance the cleaner the red), and groups where the second data set dominates would be blueish.

There are two important "modifiers" to how data sets are used that are important for comparison reports. By default comparison reports should normalize the data: the traffic of the two data sets must be scaled so that the visual representation of the total traffic of the two data sets occupies the same length on the screen. This is compulsory when one compares different units: traffic measured in bytes versus traffic measured in packets. When the traffic is measured in the same unit for both sets, the user should be able to turn normalization off. The user should also be able to "toggle" the direction of the traffic in the second set: source addresses become destination addresses and source ports become destination ports (and vice versa). This is useful when comparing the two directions of the same link. For example the incoming direction of the link to the Internet will have external addresses as source addresses and internal addresses as destination addresses, whereas the opposite direction of traffic will have source addresses from the internal network and destination addresses from the rest of the world. By toggling the sources and destinatins for outgoing direction, a comparison report on the destination address would show the distribution of the traffic among internal networks for the two directions of the link.

2.4 Different "units" of measurement

Currently Netpy can measure the traffic in either bytes or packets. Measuring the number of connections is a better way to reveal network scans and some types of denial of service attacks. There are two ways to count the number of connections: by counting the number of SYN packets we can get an accurate count of the number of TCP connections started, by counting distinct flow identifiers in the traffic (counting active flows) we can capture UDP flows and ICMP scans as well.

2.4.1 Active flow counts

Counting active flows has the advantage of also capturing UDP and ICMP scans. It has the disadvantage that active flow counts cannot be recovered accurately if we use as input sampled data such as sampled NetFlow. The active flow count for a traffic group during a time interval is defined as the number of distinct flow identifiers in the traffic from that time bin.

To preserve active flow counts the database must use a different sampling method than the one used for packets and bytes. This is already partially implemented in Netpy.

2.4.2 SYN based flow counts

SYN based flow counts rely on the fact that the first packet of each TCP connection has the SYN flag set. This is true for both directions (from client to server and from server to client) as long as there are no losses and retransmissions (in that case there can be more packets with the SYN flag set). The flow records collected by routers keep a flag that is set if at least one of the packets counted against the record had the SYN flag set. It is easy to undo the effects of sampling on SYN packet estimates: one just needs to multiply the number of records with the SYN flag set with the inverse of the sampling rate. To recover SYN based flow counts, you need to extend the database to also store the SYN flag information.

2.5 Adding database timestamps

Currently the database does not keep explicit timestamps. It stores traffic information for separate 5 minute time bins in separate files, it is able to provide reports at granularity of 5 minutes or more. By also storing some crude timestamps (say around one byte in size) for when a given flow started and ended within the 5 minute time bin, we will be able to perform analyses that look at the traffic at granularity lower than the 5 minute bin size. This would come at the cost of increased flow record sizes (and extra code to implement it).

3 Better usability

Stronger analysis will increase the power of Netpy, but judging from my past experience, its success with network administrators depends more on improving its usability. This section presents the ways in which new features can improve the usability of the application.

3.1 Speed improvements

The performance of the system is important. Quick results can be the key to a positive user experience (assuming they are correct). The way to find performance problems is through profiling. You should insert basic profiling hooks into your code so that you can measure how long a certain request took, and within that, how long it took to read data out of the database and how long it took to run the analysis algorithms, etc.

A relatively easy way to improve performance is through caching. Netpy already uses caching of analysis results extensively. You can extend the existing caching mechanisms as you see fit. Netpy also uses sampling to reduce the number of flow records the analysis engine works with when the database offers too many of them.

3.1.1 Faster analysis engine

The current performance bottleneck for queries over short timescales is the analysis engine. The

reason are the inefficient python algorithms implementing the hierarchical heavy hitter analysis. You should reimplement the analysis in C and use more efficient algorithms wherever possible. You should be able to obtain at least a factor of 10 speedup.

3.1.2 Faster database

For queries that involve reading much data from the database, disk reads can become a performance bottleneck. There are some genuinely hard queries that we cannot do much to improve: those that filter out a very small subset of a very large traffic volume. For these we need to store a large database with little or no sampling and read it all in. But for queries that work with large portions of the traffic we shouldn't read in a large database and later apply sampling to reduce the number of flow records we hand to the analysis engine. A more aggressively sampled database would be better.

To accommodate both types of queries we can use a database structure with multiple levels of sampling: for each 5 minute time bin keep an aggressively sampled version of the database and progressively larger versions until we reach the disk usage limit specified by the user. When the user specifies a query, the database module would read the smallest version of the database and if it provides accurate enough results[1], the resulting flow records are delivered to the analysis engine. If the smallest version of the database is too inaccurate, the database module will read larger and larger versions that use less aggressive sampling. For example the smallest version could use a sampling rate so aggressive that its size would be around 10,000 records, the next larger version could contain the next 30,000 records, the next larger the next 120,000 records, the next one 480,000 records, etc.

For queries that cover a large time period, say a week, the database module would have to read many files (2,016 for a week) corresponding to the 5 minute bins within the time period of interest. This might provide too much detail and involve significant costs in terms of disk transfers. To avoid this problem the database file hierarchy should store summaries of the traffic using coarser bin sizes too: say one hour bins, four hour bins, one day bins, and one week bins, all keeping summaries of the traffic they cover using around 10,000 records. You need not store progressively larger versions of the data for these larger time bins because to get better accuracy the database module can just read the next smaller bin size which provides more detail.

3.1.3 Precomputation

Netpy has various caches. One can speedup analyses by computing the results of anticipated queries in advance when the data is added to the database. This way the results will be cached and when the user asks for the query she will get a quick (and correct) reply. You should find a convenient way for the user to specify what analyses to precompute.

3.2 Easier traffic mix exploration

One of the important features of Netpy is that it allows the user to explore the traffic mix asking repeated queries until he finds the right combination of filters and other parameters that result in conclusive answers to the problem he is interested in.

[1] We will discuss at office hours how you determine whether a certain sampled summary of the data is accurate enough or not.

3.2.1 Automatic selection of most recent time window

For the time interval the analysis is performed on, the user should be able to select "the last hour", "the last day", and so on without having to explicitly enter the times. This requires interaction with the database module which knows what the most recent data is. The user interface should still convey what the exact times are for the report, but the user should not have to enter them explicitly. One of these settings should be the default when the user interface starts up.

3.2.2 Shortcuts in the GUI

You could add to the user interface buttons for common actions: moving to the next/previous hour/day/week, zooming in to a specific hour of the day the analysis is on, or zooming out to the surrounding week. These would add to the already popular "back" and "forward" buttons. Feel free to add other navigation buttons you find useful, but don't clutter the screen.

3.2.3 Better integration of filters and GUI drill-down

Currently clicks on various elements in the reports change the first rule in the filter. If there is a single rule, this is the correct semantics, but if there are multiple rules, it is not. It should be relatively easy to fix with the current types of rules. You should think through the issues that arise once we extend the filters as proposed in sections 2.2.1 and 2.2.2.

3.2.4 Multiview navigation

Currently Netpy allows separate windows to show different views of the data and perform different analyses. What each window shows is selected separately. The new multiview windows would have multiple views, but a common interface for selecting what gets displayed (the time interval, the filters and the links). The threshold should also be a parameter that can be set for each window. The user should be able to select separately what type of report each view within the window shows and whether it counts the result in bytes, packets or flows. The second dataset for comparison reports should also be selectable separately for each view. You should implement two multiview layouts: one view that allows the user to select any three unidimensional graphical reports (displayed in a row), and one that allows the user to select any six reports (two rows of three reports each). If you implement other multiview layouts that make sense those will count as extra credit.

3.3 Conveying information better

Even without changing the analyses Netpy is capable of performing, the user interface can be modified to convey more information, or to make it easier to understand.

3.3.1 Address, prefix and port helper

On mouse-over, display the DNS name of the IP address (reverse lookup), or the prefix it belongs to based on a list of prefix names provided by the user. Prefixes in the GUI can also be mapped to these user defined prefixes. The user should also be able to associate application names with port numbers the same way. The user should have the option of disabling DNS lookups. These features are related to the plugins from sections 2.1.4 and 2.1.5, but they are not the same.

3.3.2 Consistent user metaphors

There are many metaphors you can use to convey information to the user. For example whenever you measure the traffic in bytes you display one icon, when you measure it in packets another one, for flows another one, etc. Or you could use different colors for these cases. Another idea is related to the information that is displayed at the bottom of the window whenever you move the mouse over a "pane" representing a part of the traffic. You could also highlight the pane by drawing a contour around it. This is not as gratuitous as it sounds because sometimes the information refers to say a prefix encompassing multiple panes. This is a very open-ended thing. Let you imagination run wild and come up with the coolest metaphors you can think of. The important thing is that it should be intuitively clear to most users what you want to convey (and it shouldn't be annoying for the few who don't get it).

3.3.3 Conveying accuracy of results

The database module can estimate the size of the sampling error affecting the output of a report. This information should be conveyed (possibly in a simplified fashion) to the user. For example you could use the ratio between the standard deviation of the estimate of the total selected traffic and the actual value.

3.3.4 Selecting target accuracy

The user should be able to specify a desired level of accuracy for the results. This option should be available through the graphical user interface and the console. If the user-specified limits for the database size force you to do some sampling, it might be impossible to provide results as accurate as the user wants. But when you have a choice of digging deeper into the database or not, the user's selection can guide the choice the database module makes.

3.3.5 Better plots

The time series plots could be made easier to read. Take a look at Tobias Oetiker's RRDtool for inspiration.

3.4 Packaging

Packaging the software is incredibly important. If prospective users have to spend hours to download and compile the various graphical libraries required, they will just give up irrespective of how cool an application you put together. It is important package the application as "RPM"s or "debian packages" for Linux, and maybe with the appropriate tools for Windows (or even Mac if you want to). There should be two separate packages: one containing the database module, the analysis engine and the console, and another one with the graphical user interface. Users might choose to run the first package on a Linux server and the GUI on a Windows laptop.

3.5 Various types of input

Currently Netpy accepts NetFlow data as input.

3.5.1 Handling sampled NetFlow input

Netpy also accepts sampled NetFlow because it has the same format. But you should make the database aware that it is consuming sampled NetFlow so that in the results it can compensate for the sampling that has already happened by the time the flow records reach Netpy.

3.5.2 Reading packet header traces

Some people collect packet header traces with general purpose computers. There are third party programs that turn these into flow records of the type routers generate. Integrate them with Netpy so that it can consume packet header traces.

3.6 Better database management

3.6.1 Database size management

As part of a real traffic monitoring infrastructure, your application will likely have to process and store gigabytes or terabytes of data. Controlling the size of the database becomes very important. Currently the user can specify in a configuration file how much data per hour of traffic the database should hold for each link. Old data can be deleted from the command line. You should implement an additional command that would allow the user to shrink old data (by decreasing the sampling rates) instead of deleting it. For example decreasing old data's disk usage from say 20 megabytes per hour to say 1 megabyte per hour, would still allow the user to run analyses on old data, but the results would be less accurate due to the more aggressive sampling.

3.6.2 Undoing additions to the database

What happens if the user accidentally adds a batch of flow records twice to the database? Because of the sampling that the preprocessing phase does, it is impossible to cleanly remove data from the database once it has been added. Therefore it is especially important to help the user avoid the unpleasant situation when the same NetFlow file is added twice to the database due to some bug in their script. For extra credit you can implement two features providing assistance with this situation. Your application can keep a log of the names of the files that have been added to the database and exit with an error message if the same file is added twice. An even stronger feature is to also log the first say 100 records from each file and check when new data arrives. This would allow you to refuse adding the same data twice even if the file has been renamed in the meantime. You should provide a command line option that would allow the user to override both of these features.

With extra tagging you could undo additions, but it is impossible to undo the effects of the extra sampling they cause. Explore this direction.