# When AppMon met Stager

Nikos Nikiforakis, Demetres Antoniades, Evangelos P. Markatos, Sotiris Ioannidis
Institute of Computer Science
Foundation for Research & Technology – Hellas
{nikifor,danton,markatos,sotiris}@ics.forth.gr
Arne Oslebo
UNINETT, Trondheim, Norway
arne.oslebo@uninett.no

***Abstract*—Monitoring applications provide an important service in network related activities, such as network monitoring, network management and network software engineering. They facilitate the need of understanding exactly what occurs inside our networks and how each network interacts with the rest of the Internet. From private and local networks, to large-scale corporate networks and intranets, there is an ever-growing need to characterize and analyze network traffic. Unfortunately, network monitoring applications have the side effect of generating huge amounts of real-time data, that need to be processed, stored and presented, in an effective fashion. If this is done correctly and efficiently, network administrators, researchers, as well as users, can extract useful information from them, such as, traffic patterns, newly deployed network protocols, *etc.***

**In this paper we present our experiences on the combination of two tools, *AppMon* and *Stager*, and the study of the resulting system. *AppMon* is a network monitoring toolkit which performs *per-application traffic classification*. *Stager* is a tool which stores, aggregates and presents long-term network statistics, coming from multiple monitoring sides. We modified, combined, and extended these tools so that real-time data produced by *AppMon* are transferred, converted and stored through *Stager*. The resulting system gives access to *valuable aggregated long-term* network data which were not available through existing tools and methods.**

## I. Introduction

Computer networks have gotten so fast and so complex over the years, that we now face a daunting task whenever network administrators and researchers try to analyze and understand network traffic.

In the past it was easy to categorize network traffic, since all network-centric applications used static, well-known ports, such as port 80 for HTTP and port 21 for FTP (port 20 for FTP-data transfers). These static ports are still used, but the new generation of network applications such as peer-to-peer (P2P) programs and voice-over-IP (VOIP), no longer use static and predefined network ports [1], [2]. All the file sharing and Internet telephony programs (as well as worms and trojans) dynamically allocate ports every time they are installed and/or executed. This dynamic allocation of ports has stalled nearly all network monitoring applications, since the majority of them, use a list of ports and protocols to associate traffic with a specific application. To address this limitation, new monitoring applications use more sophisticated methods of traffic categorization, such as deep packet inspection and packet flow states. These tools have proven themselves very useful and have greatly assisted both researchers and network administrators in better understanding the traffic *"produced"* and *"consumed"* by their networks.

The usefulness of these kind of network monitoring applications is twofold. Their ability to present the real-time traffic distribution of a network, assists in identification and response on network problems and unusual events. Also, the data produced by these applications can be stored and used in presenting long-term statistics of the network usage. This paper's focus is on this second aspect. Our work aims at the collection, storage and presentation of the results produced by a traffic classification application, where unfortunately a new problem emerged. Due to the tremendous speed increase in computer networks, as well as increase in network users, we are faced today with huge amounts of network monitoring data to process. Capturing, storing, and analyzing, such large amounts of data in order to extract useful information is proving to be quite a challenging task. This paper presents our approach of capturing, storing, handling and presenting network data. Capturing is performed by *AppMon*, a novel network monitoring toolkit we have been developing at FORTH over the last couple of years. A more detailed description will be presented in Section III-A. Storing and handling, are performed by a tool developed at Uninett, called *Stager* (see Section III-B for more details). We developed a new system that combines and extends *AppMon* and *Stager*, and allows for a plethora of information extraction from both real-time and post mortem network data traffic.

Network administrators and researchers now have the unique ability of *"traveling back in time"* and examining exact screenshots of their network traffic at any date and time they may choose.

## II. Related Work

On the front of network traffic classification, several methods have been proposed over the years. The oldest and probably most outdated method is the classification by port number, which *AppMon* only uses as a backup identification method – when everything else fails – for standard web services and for well-known services were the payload is encrypted. Karagiannis *et al.*, presented a new way of characterizing traffic using statistics and graphlets called Blinc [3]. They stipulate that each class of network traffic (web, P2P, *etc.*) has different identifiable characteristics which can be used to

classify it. *AppMon* is different from Blinc, since it uses deep packet inspection (in a manner similar to [2], [4]) instead, and flow-state analysis, in order to classify the captured network traffic.

On the part of monitoring data representation, we found out that most tools for presenting network data were application-specific. That is, they were created with a very specific data-set in mind, *e.g.* NetFlow [5]. *Stager* is different due to its *"modular"* nature. *Stager* consists of a core that handles all the computations, web-frontend and database communication. The application-specific work is done through a set of modules. By using different modules, data visualization of different traffic-capturing tools, can be done through the same application. The code bridge that we constructed for our system, is in essence the development of the *AppMon*-specific module for *Stager* along with the architecture and methods of connecting multiple-remote *AppMon* network sensors to a central *Stager* server.

## III. *AppMon* AND *Stager* OVERVIEW

Before we go into the details of our new integrated system, we will give a brief overview of the *AppMon* toolkit and the *Stager* tool, and how they operate.

### A. AppMon Description

The *AppMon* toolkit [6] passively monitors traffic passing through a monitored link and analyzes active network flows. Flows are identified by a 5-tuple: source and destination IP addresses, source and destination port numbers, and transport layer protocol. Once identified they are categorized according to the application that generated them.

Traffic categorization is performed using information from both the packet header and the payload. *AppMon* uses different classification methods for different application protocols such as deep packet inspection for P2P networks and control-port monitoring for well known protocols.

*AppMon* stores the data mined from the previous steps in a round-robin database which is updated every 10 seconds (configurable parameter) with new data. To present the traffic classification *AppMon* , uses a web interface (see Figure 1).

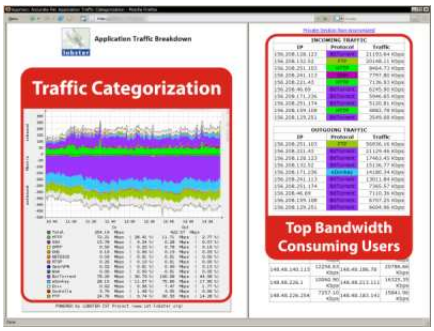


Fig. 1. *AppMon* Web Interface. Traffic classification of the entire network is visualized in the left pane of the interface, while the right pane displays the "heaviest," in terms of network use, IP-application pairs.

On the left pane of the Web interface, a graph visualizes both incoming and outgoing traffic distribution of the monitored network. The time period presented on the graph is user selectable. The right pane of the Web interface *AppMon* displays information about the hosts that produce and consume the traffic during the selected time period. Specifically, the top 10 IP addresses and protocol combinations that are responsible for the incoming and outgoing traffic.

### B. Stager Description

*Stager* [7] is a system for aggregating and presenting network statistics. It is a generic tool that can be customized to present and process any kind of network statistics. It consists of two basic parts: $(i)$ The back-end that collects data from different monitoring applications, formats them and stores them as reports in a database. It automatically handles the aggregation of hourly statistics into days, weeks, and months. $(ii)$ The front-end that connects to the database and handles the presentation of the stored data. It is responsible for creating the appropriate tables, matrices and on-the-fly plots, which it then embeds in an efficient and simple point and click interface. The reports are fully customizable and their definitions are stored in the database. PostgreSQL is used as the underlying DMBS.

*Stager* names each distant monitoring application in a monitoring location as an "*observation point*." For example, a traffic classification application and an RRT measurement application are considered two different observation points.

Users of *Stager* have access to the data of different observation points through a Web Interface. Through the main interface, shown at Figure 2, the user can select both the observation point and time period she wishes. The data created by the monitoring applications and stored by *Stager* can be presented either using a table format, or using a graph interface. *Stager*'s interface also gives the user the ability to see long-term network data aggregated by the back-end. By clicking on the appropriate buttons of the user interface, the user can instantly change the granularity of data shown from minutes to hours, days, months and even years.

Since *Stager* is a generic tool, it consists of different modules, each one written to suit the needs of a particular application. Until now *Stager* included back-end modules to collect and aggregate data for NetFlow, MPing, and SNMP. Our work extends *Stager* to accept data collected by *AppMon*. To accomplish this, we built a new module responsible for handling the specific data format, and create the ability of observing multiple monitoring points with just a glimpse into an easy to use and understand Web Interface.

## IV. IMPLEMENTATION

In this section we present our design choices and describe how we build our new system by integrating and extending *AppMon* and *Stager*.

### A. Problem Description

*AppMon* is a simple, yet powerful tool, capable of presenting a very precise view of the traffic on a specific network.

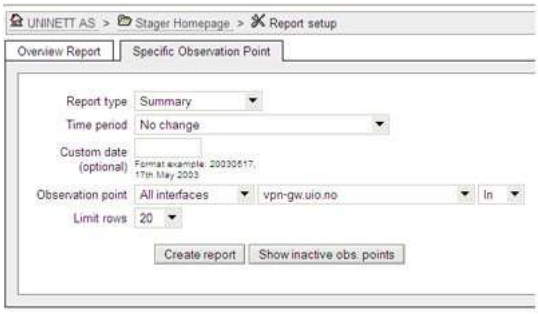


Fig. 2. *Stager* Web Interface. The interface gives the ability to the user to see the data from one (or multiple) observation point for a selected period of time.


Fig. 3. The *AppMon*-over-*Stager* layered Architecture

It has the ability to provide instant information to network administrators, *e.g.* that, currently, HTTP traffic accumulates at 3Mbps and is responsible for 23% of the overall network traffic.

The problem is that while RRD keeps full information about the current and previous day, it starts auto-aggregating data that were older than two days in an attempt to save disk space and make data processing more manageable. As time progresses, we lose the hour/minute/second precision that is generated by *AppMon*. To provide a good overview of a network's traffic over time, we *must* preserve information at a very small granularity. With the existing system it was impossible to extract any valuable information from past months since RRD aggregated the past data and then deleted them.

To overcome the above problems and difficulties we decided to integrate *AppMon*-reported results into *Stager*, and build an *AppMon*-specific back-end to handle these results.

### B. Approach

Both *AppMon* and *Stager* are made available as standalone open-source applications. But even though we could easily alter their code-base during the integration, one of our main considerations was to integrate the two application in a way that they could still be used as standalone and work independently of each other.

Since both tools store and retrieve their necessary data from files, extraction of data from these files and the creation of new ones would produce the wanted results (sticking to the Unix philosophy), without the need of modifying the programs' source code.

### C. Layered Architecture

Figure 3 presents the architecture resulting from the integration of the two tools. It is based on a *pseudo-layered* concept. As with all layered concepts and architectures, each layer has no knowledge of what goes on, on the layers below or above it. With this approach, no unnecessary modifications are made to *AppMon* and we also get an additional advantage: at any point in time, *Stager* can be un-installed and reinstalled without *AppMon* ever knowing the difference. There is no need to back up files, recompile *AppMon*, save settings, *etc.*

We logically divided the needed tasks in three distinct operations:

1) Extract and Format.
2) Transfer and Store.
3) Input into *Stager*.

Based on the above list, we created three different sub-modules each one performing one of the listed tasks.

### D. Sub-module Implementation

We will not present the implementation details involving the construction of each sub-module.

*1) Extract and Format:* The first module is responsible for the extraction and formatting of *AppMon* data. As described before, *AppMon* uses an RRD Database to store all of its gathered network traffic; so this sub-module has to extract the appropriate data from there and format them in a more "Stager-friendly" format. The actual implementation of the aformentioned extraction and formatting is done by a shell script using the RRD tool and standard Unix utilities.

*2) Transfer and Store:* The second sub-module is responsible for the transfer of the data extracted from *AppMon*. There are two possible scenarios on how this occurs. In the first one, the *AppMon*-Sensor and the *Stager*-Server are installed on the same host. If this is the case, then the transferring of the file is just a local copy of the temporary file (containing appmon data) to the *Stager* directory. In the second scenario (Fig. 4) the *AppMon*-Sensor and the *Stager*-Server are installed on different hosts, maybe even located in different networks. This scenario is more likely since most network administrators and researchers monitor more than one subnets. Each network has an *AppMon*-Sensor installed that reports back it's results. Since it is not wise to transfer sensitive network-related information over the Internet, we chose to encrypt the results file before sending it over the wire.

*3) Input into Stager:* The last module has to supply the network-records directly into the *Stager* tool, which will in turn insert them in the PostgreSQL database. We have implemented this module using PHP, in order to be compatible with the rest of *Stager*'s scripts. Our *AppMon*-specific module is created in a way that *Stager* can automatically use it without any user intervention. For each registered observation point, *Stager* calls our *AppMon* module providing the name of the current observation point. Our module uses this information

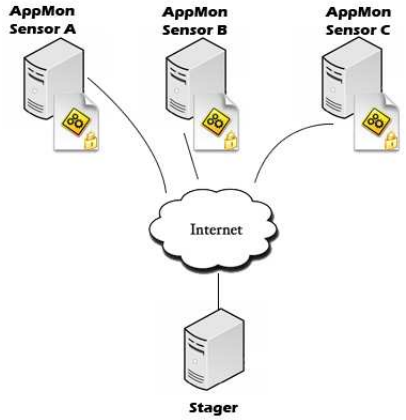


Fig. 4. Step 2: A number of remote *AppMon*-Sensors sending network-monitoring data securely (encrypted files) to a central *Stager*-Server where our last sub-module will insert them into *Stager*'s database.

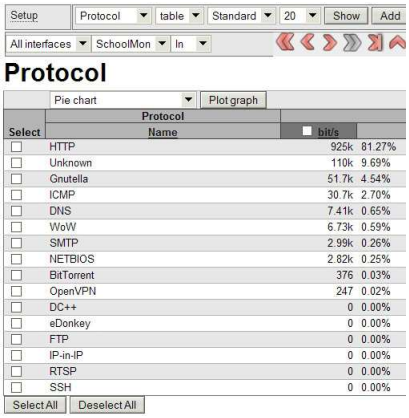


Fig. 5. Actual report for a remote *AppMon* sensor, showing the traffic distribution for various network protocols.

to locate the file containing the data for the specific observation point amongst the ones containing data for different observation points. Finally *Stager* takes control of our data and automatically aggregates and presents them as necessary.

## V. Experimental Evaluation

### A. Deployment

In our experiments we included 4 different sensors in 4 different networks. In all 4 networks, it is now possible to examine spikes in traffic, through *Stager*, that we were not able to do previously. Also through *Stager*'s aggregation, we can now see the trends in network applications (for example all networks were running Bittorrent clients) and how these trends change over time (the percentage of bandwidth allocated to Bittorrent grows steadily over time while other protocols such as FTP or Gnutella either remained the same, or declined).

### B. Resulting Data from Web Interface

The user is now capable of using *Stager*'s interface to examine data created by *AppMon* which were not available in the old *AppMon* Web interface. In the main interface, the user can now choose the specific *AppMon*-Sensor he wishes to observe, and select the exact time-period.

The report created by the previous step, is immediately available to the user for examination. Using the time-period arrows of the UI, the user can literally follow the trail of bandwidth consumption in time.

## VI. Conclusion

In this paper we argued for the need for long-term storage of the data created by network monitoring applications. These applications while very useful to network administrators and researchers worldwide, produce a great amount of data (proportional to the traffic of the monitored network), which, if not handled properly, is either lost (deleted) or useless (not presentable in an efficient way).

In our study, we chose *AppMon* as our network monitoring toolkit and *Stager* as our data storage and aggregation tool. Our aim was to combine the two tools, thus creating a system which would efficiently and securely monitor and transfer data, from remote *AppMon* sensors, to a central *Stager* server. This combination gave us access to data that were not available to us, through the old *AppMon* interface. *AppMon*'s small granularity was not restricted any more by RRDs aggressive aggregation and we had the ability to request even 4-month old, 5-minute screenshots of our monitored networks. *Stager*'s automatic aggregation gave us a new, more detailed view of our monitored networks. We can now request a network traffic screenshot of our networks at *any* point in time (*e.g.*, 13:15 on the 3rd of the month, 5 months ago), aggregated long period views, and everything in between.

## Acknowledgments

This work was supported by the IST project LOBSTER funded by the European Union under contract number 004336 and by the Marie Curie Actions - Reintegration Grants project PASS.

## References

[1] A. Moore and K. Papagiannaki, "Toward the Accurate Identification of Network Applications," *Passive And Active Network Measurement: 6th International Workshop, PAM 2005, Boston, MA, USA, March 31-April 1, 2005: Proceedings*, 2005.

[2] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "Is P2P dying or just hiding," *IEEE Globecom*, 2004.

[3] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: Multilevel traffic classification in the dark," in *Proceedings of ACM SIGCOMM 2005 Conference*, 2005.

[4] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," *Proceedings of the 13th conference on World Wide Web*, pp. 512–521, 2004.

[5] K. Lakkaraju, W. Yurcik, and A. J. Lee, "Nvisionip: Netflow visualizations of system state for security situational awareness," in *Proceedings of VizSEC/DMSEC 2004 Conference*, October 2004.

[6] D. Antoniades, M. Polychronakis, S. Antonatos, E. P. Markatos, S. Ubik, and A. Oslebo, "Appmon: An application for accurate per application traffic characterization," in *Proceedings of IST Broadband Europe 2006 Conference*, December 2006.

[7] A. Oslebo, "Stager: A web based application for presenting network statistics," in *Proceedings of NOMS 2006 Conference*, 2006.