LOBSTER: A European Platform for Passive Network Traffic Monitoring

Demetris Antoniades
FORTH-ICS
Heraklion, Greece
danton@ics.forth.gr

Michalis Polychronakis
FORTH-ICS
Heraklion, Greece
mikepo@ics.forth.gr

Antonis Papadogiannakis
FORTH-ICS
Heraklion, Greece
papadog@ics.forth.gr

Panagiotis Trimintzios
ENISA
Heraklion, Greece
panagiotis.trimintzios@enisa.eu

Sven Ubik
CESNET
Prague, Czech Republic
ubik@cesnet.cz

Vladimir Smotlacha
CESNET
Prague, Czech Republic
vs@cesnet.cz

ABSTRACT

Over the past few years we have been witnessing a large number of new programs and applications which generate prolific amounts of questionable, if not illegal, traffic that dominates our networks. Hoping from one port to another and using sophisticated encoding mechanisms, such applications have managed to evade traditional monitoring tools and confuse system administrators.

In this paper we present a concerted European effort to improve our understanding of the Internet through the LOBSTER passive network traffic monitoring infrastructure. By capitalizing on a novel *Distributed Monitoring Application Programming Interface* which enables the creation of sophisticated applications on top of commodity hardware, LOBSTER empowers a large number of researchers and system administrators into reaching a better understanding of the kind of traffic that flows through their networks.

We have been running LOBSTER for more than a year now and we have deployed close to forty sensors in twelve countries in three continents. Using LOBSTER sensors

- we have captured more than 600,000 sophisticated cyberattacks which attempted to masquerade themselves using advanced polymorphic approaches
- we have monitored the traffic of entire NRENs making it possible to identify the magnitude (as well as the sources) of file-sharing (peer to peer) traffic.

Categories and Subject Descriptors

C.2.3 [**Computer Communication Networks**]: Network Operations,Network Monitoring; C.2.0 [**Computer Communication Networks**]: General,Security and protection

General Terms

Netork monitoring, Security, Traffic classification, Distributed monitoring

TRIDENTCOM 2008, 17th – 20th Mar 2008, Innsbruck, Austria.
 ISBN 978-963-9799-24-0
DOI 10.4108/icst.tridentcom.2008.3153

1. INTRODUCTION

Network traffic monitoring is getting increasingly important for a large set of Internet users and service providers, such as ISPs, NRENs, computer and telecommunication scientists, security and network administrators, and managers of high-performance computing infrastructures. As networks get faster and as network-centric applications get more complex, our understanding of the Internet continues to diminish. The world is often surprised with security breaches, such as Internet worms and Denial of Service (DoS) attacks, and the extent of unclassified traffic due to applications that use dynamic ports.

Current monitoring standards often force administrators to tradeoff functionality for interoperability [16]. Passive traffic monitoring and capture has been regarded as the main solution for advanced network monitoring and security systems that require fine-grained performance measurement, such as "deep" packet inspection [8]. Current passive network monitoring applications are commonly based on data gathered at a single observation point. For instance, Network Intrusion Detection Systems (NIDS) usually run on a single monitoring host that captures the network packets and processes them by performing tasks such as filtering, TCP stream reassembly and pattern matching [21].

Several emerging applications would benefit from monitoring data gathered at multiple observation points across a network. For instance, Quality of Service (QoS) applications could be based on traffic characteristics that can be computed only by combining monitoring data from both the ingress and egress nodes of a network. However, a distributed monitoring infrastructure can be extended outside the border of a single organization and span multiple administrative domains across the Internet. The installation of several geographically distributed network monitoring sensors provides a broader view of the network in which large-scale events could become apparent.

Recent research efforts [24–26] have demonstrated that a large-scale monitoring infrastructure of distributed cooperative monitors can be used for building Internet worm detection systems. Distributed DoS attack detection applications would also benefit from multiple vantage points across the Internet. Finally, user mobility necessitates distributed monitoring due to nomadic users who change locations frequently across different networks.

It is clear from the above that distributed network monitoring is becoming necessary for understanding the performance of modern networks and for protecting them against security breaches.

This paper presents the design, deployment and achievements of LOBSTER, a large-scale infrastructure for distributed passive

network monitoring. LOBSTER provides an advanced pilot European passive Internet traffic monitoring infrastructure that improves our understanding of the Internet and contributes towards solving difficult performance and security problems. Based on appropriate abstractions and willing cooperation among points of presence, LOBSTER contributes towards effectively monitoring the underlying network, providing early warning for security incidents, and providing accurate and meaningful measurements of performance.

The main goal of LOBSTER is to deploy an advanced pilot European Internet Traffic Monitoring Infrastructure based on passive monitoring sensors at speeds starting from 2.5 Gbps and possibly up to 10 Gbps. It also aims to develop appropriate data anonymising tools to prohibit unauthorized tampering with the original traffic data, and to develop novel applications to improve network monitoring, such as traffic characterization and zero-day worm spread detection. Another objective is to provide properly anonymised traffic data to network researchers and security analysts.

Till now the LOBSTER monitoring infrastructure consists of 36 sensors worldwide, most of them located in Europe, operating up to 35 Gbit/sec totally and monitoring about 2.5 million IP addresses. These LOBSTER monitoring sensors run 8 different applications for performance and security monitoring. To the best of our knowledge, LOBSTER constitutes the largest platform for Internet traffic monitoring in Europe.

The paper continues with a description of the LOBSTER platform high level architecture in Section 2. Section 3 introduces the Distributed Monitoring Application Programming Interface which facilitates the development of new LOBSTER applications, while Section 4 presents the architecture of a distinct monitoring sensor. The mechanisms for ensuring only authorized use of the monitoring sensors and for preserving the privacy of the monitored organizations are described in Section 5. Section 6 outlines several useful applications developed by LOBSTER for performance and security monitoring, while Section 7 presents the current deployment of the distributed infrastructure around the world. Section 8 gives details about the availability of the software and finally Section 10 concludes the paper.

2. THE LOBSTER PLATFORM ARCHITECTURE

The need for elaborate monitoring of large-scale network events and characteristics requires the cooperation of many, possibly heterogeneous, monitoring sensors, distributed over a wide-area network, or several collaborating Autonomous Systems (AS). In such an environment, the processing and correlation of the data gathered at each sensor gives a broader perspective of thestate of the monitored network, in which related events become easier to identify.

LOBSTER aims to deploy a large scale infrastructure for distributed Internet traffic monitoring, based on passive monitoring sensors that are located across Europe. Figure 1 illustrates a high-level view of the LOBSTER distributed passive network monitoring infrastructure. Monitoring sensors are distributed across several autonomous systems, with each AS having one or more monitoring sensors. Each LOBSTER sensor may monitor the link between the AS and the Internet (as in AS 1 and 3), or an internal link of a local sub-network (as in AS 2). An authorized user, who may not be located in one of the participating ASes, can run monitoring applications that require the involvement of an arbitrary number of the available monitoring sensors.

To support collaborative passive network monitoring across a large number of geographically distributed—and possibly heterogeneous—sensors, LOBSTER needs a uniform access platform that will provide to the monitoring applications a common interface for concurrent access to several distributed sensors. This platform is realized by introducing an Application Programming Interface for Distributed Monitoring, called DiMAPI. Based on the network flow *scope* abstraction, DiMAPIenables the manipulation of compound network flows that may consist of traffic captured at several geographically distributed monitoring sensors. DiMAPIallows monitoring applications to interact concurrently with multiple remote sensors across the Internet. It provides a flexible and expressive programming interface that fulfills the needs of emerging application and it is able to exploit specialized monitoring hardware, when it is available, transparently to the applications. The basic functionality of DiMAPIis described in Section 3.

Each LOBSTER sensor runs a monitoring daemon which is responsible for packet capturing and processing. A communication agent accepts requests from monitoring applications, based on DiMAPI, and forwards them to the monitoring daemon. The monitoring daemon performs the packet processing imposed by the application and produces the corresponding results, that are returned to the applications through the communication agent. An authentication daemon is responsible to authenticate the authorized users before starting to run their monitoring applications in each sensor, while the monitoring daemon is responsible to enforce discrete data anonymization policies for applications of different group of users, as explained in Section 5. The components of a LOBSTER monitoring sensor are further discussed in Section 4.

The LOBSTER monitoring applications are developed based on DiMAPI. Each application first creates a network flow scope with a set of monitoring sensors and then configures the flow, by restricting the packets that consist this flow and defining the processing that will be performed by each monitoring daemon. Finally, the monitoring sensors return the results to the application, periodically or upon requests, which are responsible to correlate, analyze and present them to the end users.

A large-scale network monitoring infrastructure, like LOBSTER, consisting of many sensors across the Internet is exposed to several threats that may disrupt its operation, so privacy and security issues must be concerned. The authentication daemon provides *access control* by authenticating each user before starting an application, so non-authorized users cannot misuse the LOBSTER infrastructure neither obtain access to sensitive data. Moreover, since all communication between user applications and the remote sensors will be made through public networks across the Internet, encryption using the Secure Sockets Layer protocol (SSL) ensures the *confidentiality* of the transferred data. Finally, LOBSTER is a distributed monitoring infrastructure that promotes sharing of network packets and statistics between different parties, so the exchanged data should be *anonymized* before given to monitoring applications for security, privacy, and business competition concerns that may arise due to the lack of trust between the collaborating parties. Since different users and applications may require different levels of anonymization, in LOBSTER anonymization mechanism each user belongs to a group, called *virtual organizations*, in which different anonymizations policies are applied, configured by the sensors administrators.

3. THE DISTRIBUTED NETWORK MONITORING API

In order to take advantage information from multiple vantage points in LOBSTER, monitoring applications need a uniform access platform for interaction with several remote monitoring sensors. In this extent, we have developed DiMAPI [22], an API for

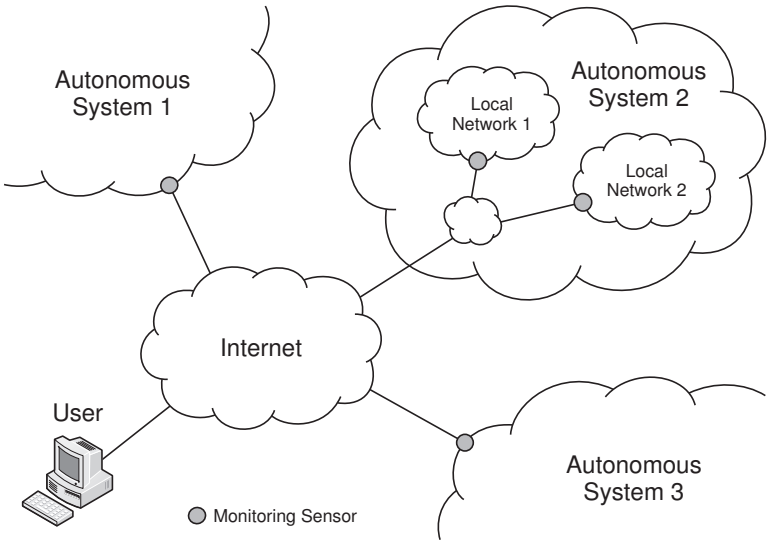

Figure 1: A high-level view of a distributed passive network monitoring infrastructure.

distributed passive network monitoring, that fulfills this requirement by facilitating the concurrent programming and coordination of a set of remote sensors within a single monitoring application. DiMAPI enables users to efficiently configure and manage any set of remote passive monitoring sensors, acting as a middleware to homogeneously use a large distributed monitoring infrastructure. Furthermore, DiMAPI exploits any specialized hardware available at the monitoring sensors, and efficiently shares the monitoring infrastructure among many users.

3.1 Network Flow Scope

One of the main novelties of DiMAPI is the introduction of the network flow *scope*, a new attribute of network flows. In DiMAPI, each flow is associated with a scope that defines a set of monitoring interfaces which are collectively used for network traffic monitoring.

Generally, given an input packet stream, a *network flow* is defined as a sequence of packets, subset of the original packet stream, that satisfy a given set of conditions [19]. These conditions can be arbitrary, ranging from simple header-based filters to sophisticated protocol analysis and content inspection functions.

The notion of scope in DiMAPI enables a network flow to have as input packets from several monitoring interfaces. With this definition, the abstraction of the network flow remains intact: a network flow with scope is still a subset of the packets of an input packet stream. However, the input packet stream over which the network flow is defined may come from more than one monitoring points. In this way, when a monitoring application performs operations to manipulate or extract information from a network flow with a scope of multiple sensors, it effectively manipulates and extracts information concurrently from all these monitoring points.

In the example of Figure 2, a monitoring application creates a network flow associated with two remote sensors located in two different organizations, FORTH and UNINETT. The user's monitoring application restricts the packets of the flow to only those that are destined to some web server, i.e., the packets with destination port `80`. As a result, the network flow consists of packets with destination port 80 that are captured from both UNINETT's and FORTH's sensors.

3.2 Basic Operations of DiMAPI

DiMAPI builds on the generalized network flow scope abstraction and offers a standardized API, flexible and expressive enough to capture emerging application needs.

Central to the operation of DiMAPI is the action of creating a new network flow scope:

```
int mapi_create_flow(char *scope)
```

This call creates a network flow and returns a flow descriptor `fd` that refers to it. By default, a newly created flow consists of all network packets that go through the monitoring interfaces that are included in `scope`. When a network flow is not needed any more, it can be closed using `mapi_close_flow()`.

The abstraction of the network flow allows users to treat packets belonging to different flows in different ways. For example, after specifying which packets will constitute the flow, a user may be interested in *capturing* the packets (e.g., to record an intrusion attempt), or in just *counting* the number of packets and their lengths (e.g., to measure the bandwidth usage of an application), or in *sampling* the packets (e.g., to find the IP addresses that generate most of the traffic). DiMAPI allows users to clearly communicate to the underlying monitoring systems these different monitoring needs by applying functions to network flows:

```
int mapi_apply_function(int fd, char * funct, ...)
```

The above call applies the function `funct` to every packet of the network flow `fd` in each of the monitoring sensors in the scope, and returns a relevant function descriptor `fid`. Depending on the applied function, additional arguments may be passed. Based on the header and payload of the packet, the function will perform some computation, or may optionally discard the packet.

DiMAPI provides several *predefined* functions that cover a broad range of standard monitoring needs. Several functions are provided for restricting the packets that will constitute a network flow to those that satisfy an appropriate filter or other condition, e.g., packets that are destined to a specific port number or packets that contain a specified byte sequence. Many other functions are provided for processing the traffic of a flow, like counting the number of packets and bytes of a flow, sampling packets, computing a digest of

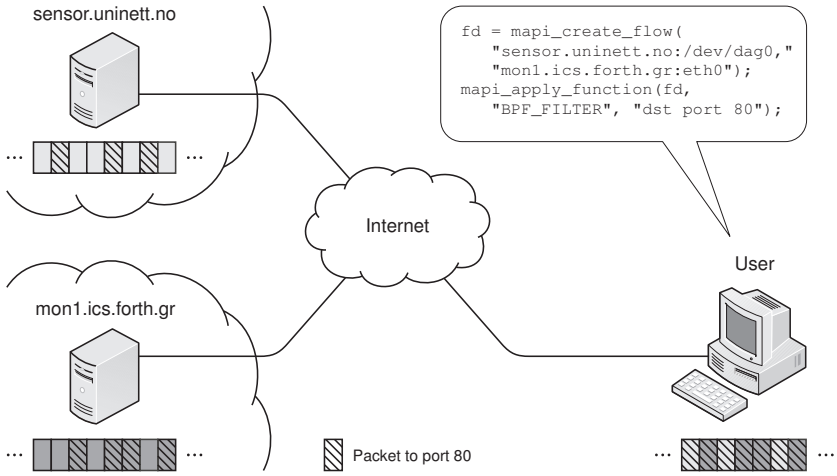

Figure 2: An example of a network flow scope with multiple sensors.

each packet, stream reassembly, NetFlow-like data generation and other. Moreover, DiMAPI users are able to add their own custom functions for operating on packets.

After applying the desirable list of functions to a network flow, the user calls the function

```
int mapi_connect(int fd)
```

in order to connect to the flow with flow descriptor `fd` to start receiving results from each sensor.

These functions give the opportunity for processing packets and computing network traffic metrics locally at the monitoring sensors, without receiving all the actual packets in the remote applications. The monitoring sensors send to the remote applications only the results they need, according to the applied functions. This approach is much more effective, since the network overhead is minimized and the latency is significantly reduced when the sensors sends only useful result summaries instead of all the captured packets to the applications. Results retrieval is achieved using the following call:

```
mapi_results_t* mapi_read_results(int fd, int fid)
```

The above call receives the results computed by the function denoted by the function descriptor `fid`, which has been applied to the network flow `fd` and returns a data structure with the results. When receiving results from multiple remote sensors, `mapi_read_results()` returns a vector of results, one per each remote monitoring sensor.

Once a flow is established, packets belonging to that flow can be sent one-at-a-time using the following blocking call:

```
struct mapipkt * mapi_get_next_pkt(int fd, int fid)
```

The above function reads the next packet that belongs to flow `fd`.

3.3 Example of Monitoring Application Using DiMAPI

In this section we describe a simple monitoring application built on top of DiMAPI. This example application identifies covert traffic from Gnutella file sharing clients. Several Gnutella clients offer the capability to operate using HTTP traffic through port 80, masquerading as normal web traffic in order to bypass strict firewall configurations aiming to block P2P traffic. The following pseudocode illustrates how DiMAPI can be used for writing a simple application that identifies file sharing clients joining the Gnutella network using covert web traffic.

```
/* create a scope of three monitoring sensors */
fd = mapi_create_flow(
    "host1:eth2, host2:/dev/dag0, host3:eth1");

/* keep only web packets */
mapi_apply_function(fd, "BPF_FILTER",
    "tcp and port 80");

/* indicating Gnutella traffic */
mapi_apply_function(fd, "TRACK_GNUTELLA");

/* and just count them */
fid = mapi_apply_function(fd, "PKT_COUNTER");

/* activate the flow */
mapi_connect(fd);

/* get the number of monitoring sensors */
scope_size = mapi_get_scope_size(fd);

/* forever, report the number of packets */
while(1) {
  sleep(60);
  dres = mapi_read_results(fd, fid);
  total_pkts = 0;
  for (i=0; i<scope_size; i++) {
    printf("Gnutella pkts in sensor %d: %llu\n",
         i, *(unsigned long long*) dres[i].res);
    total_pkts += *(unsigned long long*) dres[i].res;
   }
  printf("Total Gnutella pkts: %llu\n", total_pkts);
}
```

We initially create a network flow with a scope of three remote monitoring sensors and apply the function `BPF_FILTER` to keep only the packets seemingly destined to, or coming from, a web server. Next, we apply the built-in DiMAPI function `TRACK_GNUTELLA`, which searches for specific protocol patterns to identify the Gnutella packets. After specifying the characteristics of the network flow, we instruct the monitoring system that we are interested in just counting the number of packets, by applying the `PKT_COUNTER` function. Finally, we activate the flow. At this point, each monitoring sensor has started inspecting the monitored traffic for covert Gnutella traffic and keeps a count of the matching packets. Then, the application periodically receives the result of the `PKT_COUNTER` function from each monitoring sensor by call-

ing `mapi_read_results()` in a infinite loop. The results from each monitoring sensor are stored in a vector of `mapi_results_t` structs. Thus, we can either analyse and report results from each monitoring sensor separately, or compute overall statistics.

Implementing the above simple distributed monitoring application using existing tools and libraries would have been a tedious process, resulting in longer code and higher overheads. For example, we could use tools like `snort` [21] or `ngrep` [20], which allow for pattern matching in the packet payload, for capturing the Gnutella packets at each remote sensor. At the end-host, we should have to use some scripts for starting and stopping the remote monitoring applications and for retrieving and collectively reporting the results, through some remote shell such as `ssh`. However, such custom schemes do not scale well and cannot offer the ease of use and flexibility of DiMAPI for building distributed monitoring applications.

Alternatively, one could build the application using solely `WinPcap` [1] or `rpcap` [11]. Both libraries extend `libpcap` [13] with remote packet capture capabilities, allowing captured packets at a remote host to be transferred to a local host for further processing. In order to count the covert Gnutella packets using one of these libraries, the application has to first transfer locally *all* the captured web packets, separately from each remote sensor, and then identify locally the Gnutella packets, count them, and finally drop them. The pattern matching operation has to be performed locally since `libpcap` does not offer any pattern matching operation. However, transferring all the web packets from each remote sensor to the local application incurs a significant network overhead. In contrast, DiMAPI enables traffic processing at each remote sensor, which allows for sending back only the computed results. In this case, only the *count* of Gnutella packets is transferred through the network, which incurs substantially less network overhead.

Furthermore, DiMAPI exploits any specialized hardware available at the monitoring sensors, and efficiently shares the monitoring infrastructure among many users. The monitoring daemon on each sensor groups and optimizes the monitoring operations requested by the users of the system, providing the same or even better performance compared to `libpcap` [19].

4. LOBSTER SENSOR ARCHITECTURE

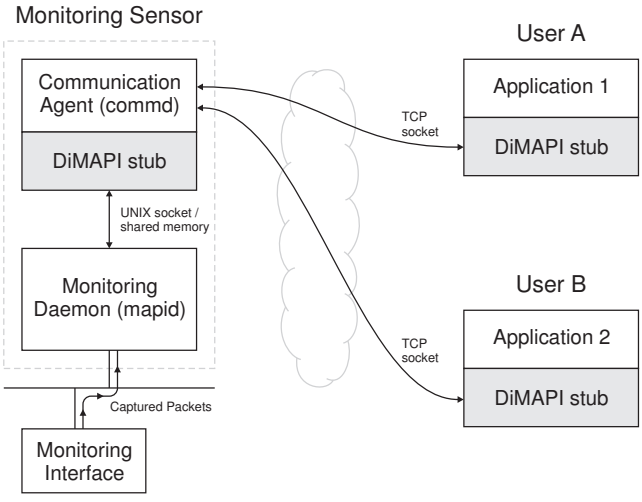

Figure 3: Architecture of a LOBSTER monitoring sensor.

Figure 3 illustrates the architecture of a LOBSTER monitoring sensor. The overall architecture includes one or more monitoring interfaces for capturing traffic, a monitoring daemon, which provides optimized passive monitoring services, a DiMAPI stub, for writing monitoring applications, a communication agent, which facilitates communication with multiple remote monitoring applications, an authentication daemon for access control and finally, the actual monitoring applications.

The monitoring sensor machine is equipped with one or more monitoring interfaces for packet capture, and optionally an additional network interface for remote access. The latter is the sensor's "control" interface, and ideally it should be separate from the packet capturing interfaces.

Packets are captured and processed by a monitoring daemon, called `mapid`. `Mapid` is a single process that serves multiple monitoring applications in parallel, so makes it possible to perform several performance optimizations and lead to better performance if compared with stand-alone monitoring applications that are not based in DiMAPI. Also, `mapid` is optimized to perform intensive monitoring tasks at high speeds, exploiting any features of the underlying hardware. Local monitoring applications communicate directly with `mapid` via a subset of the DiMAPI stub that is optimized for fast and efficient local access. This is achieved by performing all communication between local applications and `mapid` via shared memory and UNIX sockets [19].

Remote applications must be able to communicate their monitoring requirements to each sensor through the Internet. In order to avoid any modifications to `mapid`, an intermediate *communication* agent, called `commd`, is responsible for handling all the communication between `mapid` and the remote applications. `Commd`, which runs on the same host as `mapid`, acts as a proxy for the remote applications, forwarding their requests to `mapid`, and sending back to them the computed results. The DiMAPI stub is responsible to support the DiMAPI functionality in a monitoring application, running completely transparently for the user.

Finally, an authentication daemon is responsible to provide *access control*, so that only authorized users will be able to use the monitoring sensor through an application. Whenever a user's monitoring application connects to a LOBSTER monitoring sensor and requests the creation of a network flow, it passes the user's certificate. The authentication daemon performs access control based on the user's request and certificate. This certificate also specify the usage policy applicable to that user.

5. AUTHENTICATION AND ANONYMIZATION MECHANISM

Since LOBSTER's goal is to deploy a large infrastructure of network monitoring sensors, some critical concerns are the authentication and authorization of the platform's users and the confidentiality of the shared data. LOBSTER sensors are to be deployed among different organizations, so privacy concerns of each organization should be taken in mind. On the other hand, monitoring applications should have sufficient information in order to perform the indented analysis of the data.

Having all these in mind, we created an authentication, authorization and data anonymization mechanism to preserve both the privacy of the users in the monitored networks and also the usefulness of the monitoring data.

We used the abstraction of *Virtual Organizations* (VOs), that has been successfully used in the area of GRID Computing. A VO represents a group of users that have the same needs and the same privileges. A similar approach is used in the UNIX operating system user groups. Each user has a unique identifier, but also belongs

to a group. Access is defined on user level, but also in group level. For example, a user can set the permissions of a file to be readable only by a specific group of users. The advantage of this approach is the simplification of access controls, since there is no need to make decisions per user basis, but per groups instead.

LOBSTER uses a similar approach. A virtual organization, in LOBSTER, represents a group of people with common research interests and needs, which will have common access privileges. For example, assume researchers in the field of Internet security that run a project to identify worm outbreaks. In order to accomplish this, they need access to network traffic and more specifically to packet payloads. To enable these researchers use the LOBSTER infrastructure, we create a new VO, named wormTeam. Every sensor that supports this VO should define a policy which will be applied to all the members of the wormTeam.

Each LOBSTER user can be member of several VOs. When, however, LOBSTER users try to access packets from a particular monitoring sensor, they must specify one of the VO's that they belong to. Thus, we need the appropriate mechanisms for registering LOBSTER users with a VO and then mapping VOs with specific anonymization policies.

When deploying a sensor, the local administrator setups the configuration file by adding the virtual organizations that wants to support. For each VO, the monitoring daemon applies a different anonymization policy to the packets, as defined in the configuration file, and then passes them to the network flows created from users that belong to this VO. Figure 4 shows how different virtual organizations are handled inside `mapid`. Each captured packet is applied to different anonymization operations for every VO's policy. `Mapid` stores the specifications of the network flows created by the users of each VO and dispatches the anonymized packets to them for further processing. Thus, a single packet is anonymized once for all the active flows of a specific virtual organization.

In the example of Figure 4, the sensor supports three VOs, a security VO, a network measurement VO and a traffic classification VO. For the security VO access to the whole packet payload is needed, so the chosen anonymization policy is to map IP addresses and keep the payload untouched. On the other hand, the measurement VO does not need any access to the packet payload but needs information from the header of both IP and transport layer. Finally, the traffic classification VO is given access to the packet payload and has prefix-preserving anonymization on the IP addresses.

A new user of LOBSTER, that wants to become member of an existing virtual organization, must contact an external authority and get a certificate (login name and password) which can be used for running applications using a set of monitoring sensors. As soon as the user gets the certificate, (s)he can run monitoring applications by calling a specific DiMAPI function for authentication before creating a network flow. Each monitoring sensor runs an authentication daemon which receives this authentication requests and verifies the user's certificate by contacting the external authority. Also, the external authority verifies the virtual organization that the user belongs to.

6. APPLICATIONS

This section describes the various applications that are currently operating on the LOBSTER infrastructure. Though a user can easily deploy its own applications, LOBSTER offers a variety of applications ready to use. We classify the applications by the field of interest. We first describe applications used for network performance monitoring. These applications include per-application traffic categorization, available bandwidth monitoring, packet loss measurements and generic network statistics. As a second application category, we present a group of tools for flexible traffic anonymization. Finally, we present a class of security applications, that mainly facilitate for attack detection.

6.1 Performance Monitoring

6.1.1 appmon

Appmon [4] is an application for network traffic characterization. It splits the consumed bandwidth to the applications that generated it and identifies the top hosts with the highest bandwidth consumption. In order to achieve identification of network traffic we use three different approaches for different kind of applications. For complex applications, which use dynamically generated port numbers, we use sophisticated trackers that deploy deep packet inspection it two forms. The on form is to look inside the application message for applications-specific protocol patterns. The other form is to fully decode the applications protocol to identify the new, dynamically generated port number which is going to be used and categorize the corresponding flows to belong to this application. For applications that are based on well known static port numbers for data transfers, we use BPF filters in order to categorize them. We plan to implement application decoders even for more simple applications in order to avoid misclassification of other traffic or duplicate classifications of a certain flow.

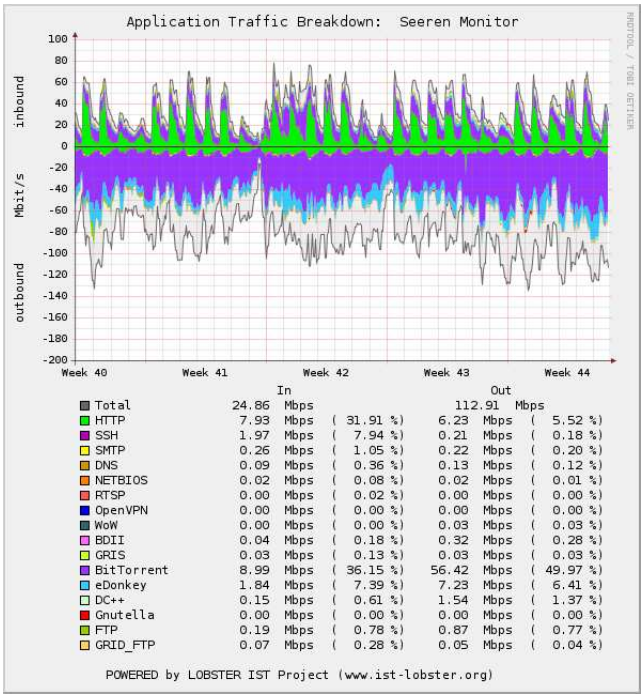

Figure 5: Per-application traffic distribution as presented by appmon.

Figure 5 shows the per-application traffic distribution as presented by appmon in one of the currently deployed LOBSTER sensors. The figure shows both inbound and outbound traffic for a period of one month, using two hour averages.

6.1.2 abw

ABW [23] application monitors how bandwidth usage on a network link is divided into various protocols in different layers corresponding to OSI model (e.g., transport layer or application layer,

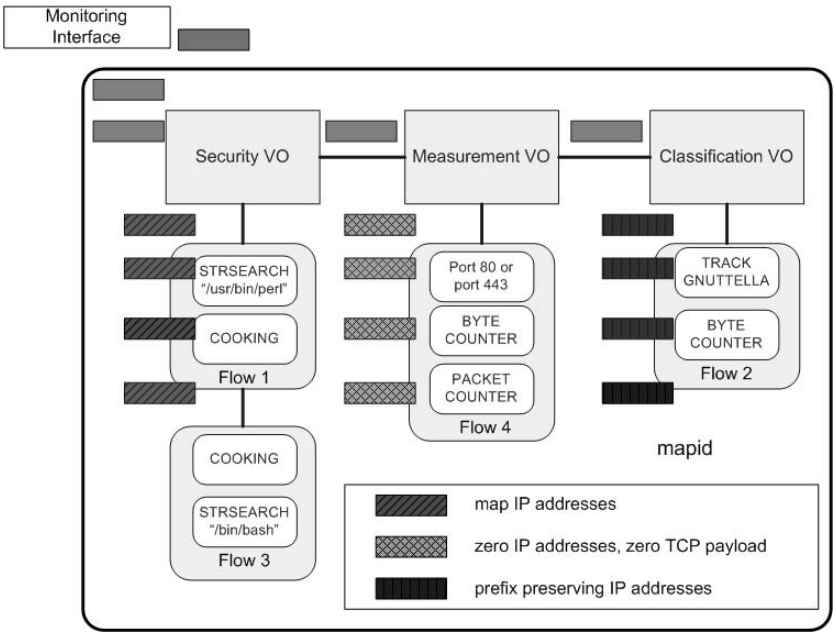

Figure 4: The LOBSTER anonymization mechanism: different anonymization policies are performed for each virtual organization.

and also IPv6 and multicast traffic). It indicates bandwidth usage in different timescales including short-term samples (1 second averages) in order to detect short peaks. It shows real bandwidth dynamics on a monitored line, without stressing user traffic with performance test, that is an inherent property of passive network monitoring.

6.1.3 Stager

Stager is an application for aggregating and presenting network statistics. Stager is generic and can be customized to present and process any kind of network statistics. The backend collects data and stores reports in a database, automatically handling the aggregation of hourly statistics into days, weeks, and months. The Web frontend presents data in tables, matrices, or plots and the displayed reports are fully customizable.

6.1.4 Packet Loss

The Packet Loss application [15] uses distributed passive network monitoring for real-time estimation of the packet loss ratio between different domains. It is based on tracking the *expired flows* at each monitoring sensor. Each monitoring sensor counts the packets per each flow (packets with the same protocol, source and destination IP address and source and destination port) and consider a flow as expired after being inactive for a specific time interval (e.g. 30 seconds). Then, by comparing statistics for the same expired flow from different sensors, we can accurately compute the packet loss for this flow, avoiding to consider as lost any packets that are still in transfer. Using DiMAPI, a central monitoring application correlates the results from different pairs of LOBSTER monitoring sensors and computes the actual packet loss ratio between each pair of sensors. This passive monitoring approach for packet loss estimation is accurate and reliable, while at the same time exhibits inherent advantages such as scalability and a non-intrusive nature.

6.1.5 LONT

The LOBSTER Network Telescope is an application that performs distributed measurements with LOBSTER and afterward post processes and visualizes the results. The set-up of the application is generic and current post processing supports the visualization of source or filtered packets e.g. from SPAM or worms. The application tracks a specific service for suspicious Internet traffic. The user specifies a description of suspected traffic and the desired form of post processing and results. The LONT service translates this description into MAPI calls and instructs LOBSTER nodes to examine the network for the specified traffic.

6.2 Traffic Anonymization

Since LOBSTER promotes the sharing of network packets and statistics between different parties, exchanged data should be *anonymized* before made publicly available for security, privacy, and business competition concerns that may arise due to the lack of trust between the collaborating parties. In these extent LOBSTER provides an extensive network anonymization tool [10], that implements several predefined policies and also supports the easy development of any possible anonymization policy.

6.3 Cyberattack Detection

6.3.1 dIDS

dIDS is a distributed intrusion detection tool. It creates network flows for intrusion detection rules, and applies them in several monitoring sensors. In this way it manages to provide the defense capabilities needed to respond to large-scale attacks.

6.3.2 EAR

EAR [2] is an application for zero-day Internet worm detection based on the identification of packets with similar contents directed to multiple destination hosts. EAR monitors the initial part of every TCP connection and processes the data stream sent from the client (connection initializer) to the server. It creates fingerprints on substrings of constant length from the data and counts the occurrence of these fingerprints among different connections. When

a certain threshold is reached, EAR issues an alert which denotes that a worm is trying to spread itself over the Internet.

6.3.3 nemu

nemu stands for Network-level Emulation. The application implements a code emulation technique that tries to detect polymorphic attacks [17, 18]. These attacks are hard to detect using traditional signature based methods, since the attack payload changes from host to host.

Nemu, based on passive network monitoring, loads and executes, in a x86 emulator, the payload of the packets designated to an internal service. If the packet contains an attack then the emulator will detect executable code into the payload.

A major advantage of this technique is that it is able to detect zero day attacks without the need of payload-specific signatures.

Figure 6 presents an overall view of the attack activity in one of the LOBSTER sensors during the last year, based on the attacks detected by nemu. The upper part of the figure shows the attack activity according to the targeted port. Red dots show attacks launched from external IP addresses, while the gray dots denote attacks originated from infected hosts within the monitored network.

7. DEPLOYMENT

During the last three years, LOBSTER deployed 36 operational passive monitoring sensors worldwide, in 12 different countries and 16 different organizations, that are mainly ISPs and NRENs networks. Figure 7 shows the deployment of these sensors around the world. Most sensors are located in Europe, while sensors also exist at Asia (Singapore) and at the United States of America (Columbia). These sensors, operating up to gigabit speeds, are together able to monitor more than 2.5 million IP addresses at any time, using the applications developed in LOBSTER.

Furthermore, as a result from the security applications deployed at these sensors, more than 600,000 sophisticated cyberattacks have been detected and captured by LOBSTER.

8. AVAILABILITY

All tools developed by the LOBSTER project as well as the distributed monitoring application programming interface (DiMAPI) are available through the project's website: `http://www.ist-lobster.org`. To facilitate the easy deployment of new LOBSTER sensors we also offer a precompiled linux distribution that includes all the necessary LOBSTER software. This distribution is available on a live cd and can easily run in any computer without any effort for installation. LOBSTER also encourages the sharing of network data among researchers. Thus, we make publicly available anonymized network traces with the attacks that we have detected so far. These traces can be found at `http://lobster.ics.forth.gr/traces`.

9. RELATED WORK

As network traffic monitoring is becoming increasingly important for the operation of modern networks, several passive monitoring infrastructures have been proposed.

CoMo [9] is a passive monitoring infrastructure which allows users to query network data gathered from multiple administrative domains, by providing a number of generic query mechanisms. It is based on a number of distributed monitoring nodes, consisting of the CoMo core processes and a number of user defined plug-in modules. Each one of these nodes is able to answer queries based on the modules that are plugged-in.

A similar approach is followed by Gigascope [6] that is a stream database for storing captured network data in a central repository for further analysis using the GSQL query language. Users are able to specify special properties to query operators using a data definition language. Gigascope is able to satisfy fast simple network monitoring needs by serving user's SQL-like queries from a central database.

Sprint's passive monitoring system [7] was installed within the Sprint IP backbone network and it was collecting data from different monitoring points into a central repository for further analysis.

All the above infrastructures are mainly based on databases with predefined custom schemes which collect data from distributed sensors and accept SQL-like queries from monitoring applications. In order to implement new functionality, new plugins must be written and embedded within the monitoring sensors. Compared to LOBSTER, none of these systems provides any API which will aid the developer to create novel distributed monitoring applications. LOBSTER adopts a different approach, by providing DiMAPI for distributed passive monitoring application development instead of supplying with a database for data queries. Moreover, LOBSTER implements a data anonymization mechanism to cope with any possible privacy issues.

Arlos et al. [5] propose DPMI, a distributed passive measurement infrastructure that supports various monitoring equipment within the same administrative domain. DPMI defines the means of creating a testbed that will provide passive monitoring capabilities to data consumers based on a number of predefined measurement points

ETOMIC [14] is a European Union sponsored effort, that aims at providing a Paneuropean traffic measurement infrastructure that facilitates for any kind of active probing techniques using highprecision, GPS-synchronized monitoring nodes.

Finally, a lot of work is being done in the area of monitoring of high performance computing systems, such as clusters and Grids. Ganglia [12] is a distributed monitoring system based on a hierarchical design targeted at federations of clusters. GridICE [3] is a distributed monitoring tool integrated with local monitoring systems with a standard interface for publishing monitoring data. These systems could utilize at lower levels the functionality offered by DiMAPI.

10. CONCLUSION

In this paper we presented LOBSTER, a large scale infrastructure for Internet traffic monitoring. LOBSTER sensors are able to perform sophisticated operations on top of commodity hardware, benefiting from a novel Distributed Monitoring Application Programming Interface.

By providing user authentication and data anonymization, LOBSTER facilitates the cooperation among different organizations and different network research needs. Distinct organizations can protect their privacy by enforcing policies regarding the network data exposed to the users of other organizations. By providing a flexible anonymization framework, different policies can be applied for different groups of users that will not decrease their understanding of the network traffic.

LOBSTER software includes several ready to use applications for effective network monitoring and security. Though, any user is able to write new applications that fulfill his needs utilizing the extended functionality offered by DiMAPI.

LOBSTER is fully operating for the last years, counting 36 monitoring sensors so far. These sensors are located mostly in Europe, but collaboration also exists with organizations from all around the world. Monitoring about 2.5 million IP addresses, LOBSTER is

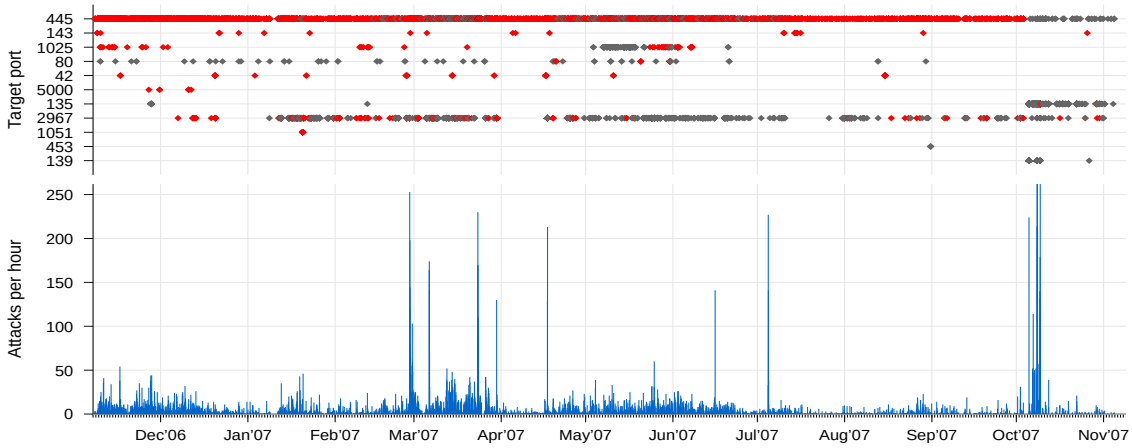

Figure 6: Attacks captured by nemu during the last year. Upper part shows attack activity according to the targeted port.

able to identify the magnitude of file sharing traffic and capture more than 600,000 sophisticated cyberattacks. To the best of our knowledge, LOBSTER constitutes the larger platform for Internet monitoring in Europe and is among the largest platforms worldwide.

The future of the LOSBTER infrastructure includes the development and deployment of new potential applications for present and future network monitoring needs and the expansion of the infrastructure by deploying new monitoring sensors and involve more organizations. The ease of deployment of a new monitoring sensor and the mechanisms provided by LOBSTER for ensuring privacy give opportunities for significant extension of the current infrastructure.

11. ACKNOWLEDGMENTS

This work was supported by the IST project LOBSTER funded by the Europen Union under Contract No. 004336. The work of Demetris Antoniades, Michalis Polychronakis, Antonis Papadogiannakis and Evangelos Markatos was also supported by the GSRT project Cyberscope funded by the Greek Secretariat for Research and Technology under the Contract No. PENED 03ED440. This work was done while P. Trimintzios was with FORTH-ICS.

12. ADDITIONAL AUTHORS

Additional authors: Arne Øslebø (UNINETT, Trondheim, Norway, email: `Arne.Oslebo@uninett.no` and Evangelos P. Markatos (FORTH-ICS, Heraklion, Greece, email: `markatos@ics.forth.gr`).

13. REFERENCES

[1] WinPcap Remote Capture. `http://www.winpcap.org/docs/docs31beta4/html/group__remote.html`.

[2] P. Akritidis, K. Anagnostakis, and E. Markatos. Efficient content-based detection of zero-day worms. *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, 2, 2005.

[3] S. Andreozzi, N. D. Bortoli, S. Fantinel, A. Ghiselli, G. Rubini, G. Tortone, and M. Vistoli. GridICE: a Monitoring Service for Grid Systems. *Future Generation Computer Systems Journal*, 21(4):559–571, Apr. 2005.

[4] 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.

[5] P. Arlos, M. Fiedler, and A. A. Nilsson. A distributed passive measurement infrastructure. In *Proceedings of the 6th International Passive and Active Network Measurement Workshop (PAM'05)*, pages 215–227, 2005.

[6] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *Proceedings of the ACM SIGMOD international conference on Management of data*, 2003.

[7] C. Fraleigh, C. Diot, B. Lyles, S. Moon, P. Owezarski, D. Papagiannaki, and F. Tobagi. Design and Deployment of a Passive Monitoring Infrastructure. In *Proceedings of the Passive and Active Measurement Workshop*, Apr. 2001.

[8] M. Grossglauser and J. Rexford. Passive traffic measurement for IP operations. In *The Internet as a Large-Scale Complex System*, pages 91–120. 2005.

[9] G. Iannaccone, C. Diot, D. McAuley, A. Moore, I. Pratt, and L. Rizzo. The CoMo White Paper, 2004. `http://como.intel-research.net/pubs/como.whitepaper.pdf`.

[10] D. Koukis, S. Antonatos, D. Antoniades, E. Markatos, and P. Trimintzios. A Generic Anonymization Framework for Network Traffic. *Communications, 2006 IEEE International Conference on*, 5, 2006.

[11] S. Krishnan. rpcap. `http://rpcap.sourceforge.net/`.

[12] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7), July 2004.

[13] S. McCanne, C. Leres, and V. Jacobson. libpcap. Lawrence Berkeley Laboratory, Berkeley, CA. (software available from http://www.tcpdump.org/).

[14] D. Morato, E. Magana, M. Izal, J. Aracil, F. Naranjo, F. Astiz, U. Alonso, I. Csabai, P. Haga, G. Simon, et al. The European Traffic Observatory Measurement Infrastructure (ETOMIC): a testbed for universal active and passive measurements. *Testbeds and Research Infrastructures for the*

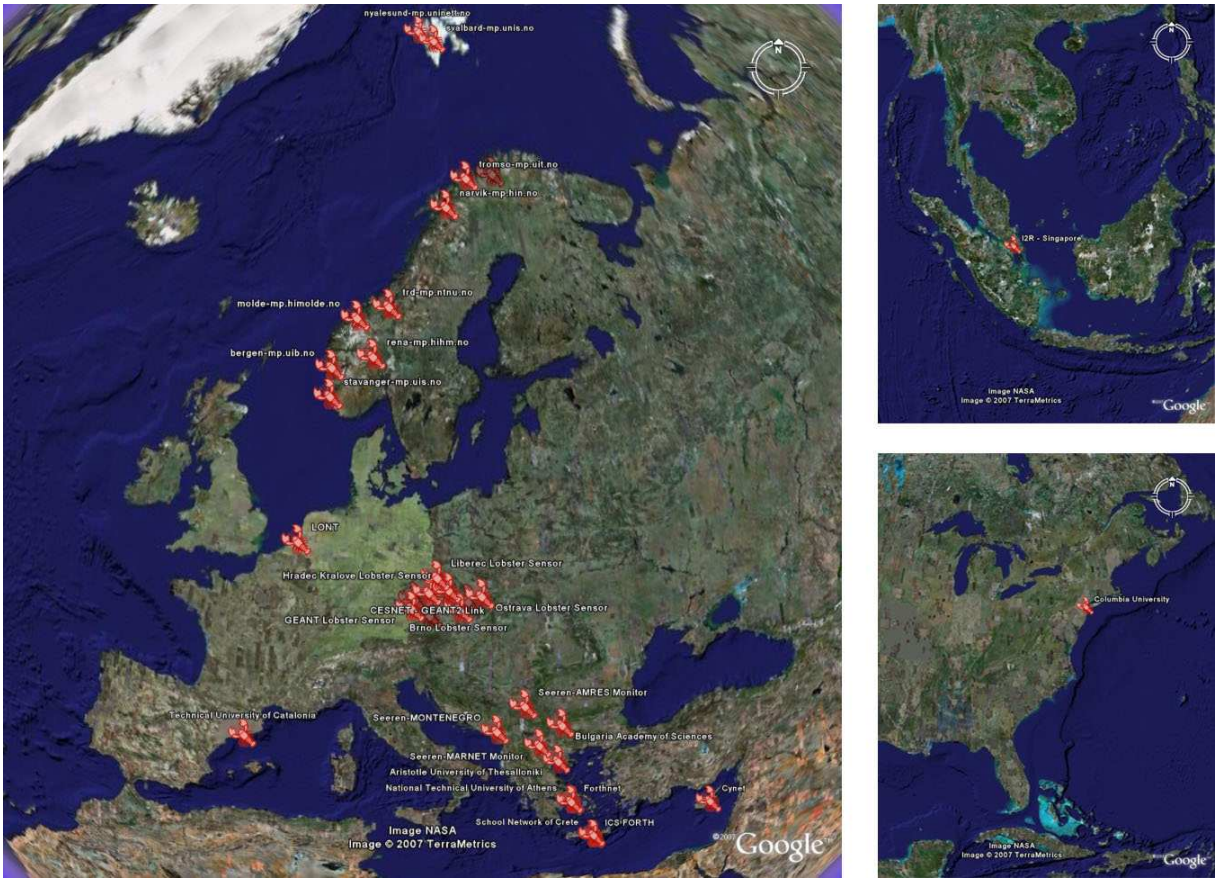

Figure 7: Large-scale LOBSTER deployment with 36 sensors around the world.

Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on, pages 283–289, 2005.

[15] A. Papadogiannakis, A. Kapravelos, M. Polychronakis, E. P. Markatos, and A. Ciuffoletti. Passive end-to-end packet loss estimation for grid traffic monitoring. In *Proceedings of the CoreGRID Integration Workshop*, 2006.

[16] Peter Morriessy. RMON2: To the Network Layer and Beyond! *Network Computing*, Feb. 1998. `http://www.nwc.com/903/903f1.html`.

[17] M. Polychronakis, K. Anagnostakis, and E. Markatos. Network-level Polymorphic Shellcode Detection using Emulation. In *Proceedings of the Third Conference on Detection of Intrusions and Malware and Vulnerability Assesment (DIMVA)*, 2006.

[18] M. Polychronakis, K. Anagnostakis, and E. Markatos. Emulation-Based Detection of Non-self-contained Polymorphic Shellcode. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2007.

[19] M. Polychronakis, K. G. Anagnostakis, E. P. Markatos, and A. Øslebø. Design of an Application Programming Interface for IP Network Monitoring. In *Proceedings of the 9th IFIP/IEEE Network Operations and Management Symposium (NOMS'04)*, pages 483–496, Apr. 2004.

[20] J. Ritter. ngrep – Network grep. `http://ngrep.sourceforge.net/`.

[21] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 1999 USENIX LISA Systems Administration Conference*, November 1999.

[22] P. Trimintzios, M. Polychronakis, A. Papadogiannakis, M. Foukarakis, E. P. Markatos, and A. Øslebø. DiMAPI: An application programming interface for distributed network monitoring. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2006.

[23] S. Ubik, D. Antoniades, and A. Oslebo. Abw–short-timescale passive bandwidth monitoring. In *Sixth International Conference on Networking*, 2007.

[24] K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2005.

[25] J. Wu, S. Vangala, L. Gao, and K. Kwiat. An effective architecture and algorithm for detecting worms with various scan techniques. In *Proceedings of the 11th Network and Distributed System Security Symposium (NDSS)*, 2004.

[26] C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. In *Proceedings of the 10th ACM conference on Computer and communications security (CCS)*, pages 190–199, 2003.