YAF: Yet Another Flowmeter

Christopher M. Inacio
CERT
Software Engineering Institute
Carnegie Mellon University
inacio@cert.org

Brian Trammell
Communication Systems Group
ETH Zurich
trammell@tik.ee.ethz.ch

Abstract

A flow meter generates flow data - which contains information about each connection observed on a network - from a stream of observed packets. Flow meters can be implemented in standalone measurement devices or inline on packet forwarding devices, such as routers. YAF (Yet Another Flowmeter) was created as a reference implementation of an IPFIX Metering and Exporting Process, and to provide a platform for experimentation and rapid deployment of new flow meter capabilities. Significant engineering effort has also gone into ensuring that YAF is a high performance, flexible, stable, and capable flow collector. This paper describes the some of the issues we encountered in designing and implementing YAF, along with some background on some of the technologies that we chose for implementation. In addition we will describe some of our experiences in deploying and operating YAF in large-scale networks.

1 Introduction

Network traffic continues to grow at an exponential rate, with global internet traffic forecast to increase 34% year-on-year though the first half of this decade [5]. Understanding the uses of the network and the needs of its users is necessary for both operations and planning, for both business and technical reasons. The need for network monitoring has therefore never been greater in today's large-scale networks. While various tools exist to aid in this problem, network flow data represents the most comprehensive way to get an in-depth understanding of network activity while still leveraging a huge amount of data reduction necessary in order to practically analyze large-scale network traffic.

The CERT Network Situational Awareness (NetSA) Group had previously developed the System for Internet Level Knowledge (SiLK) [12] in order to address the analysis issues in this area. The SiLK tools are designed to support the understanding of network flow information for both network traffic and engineering, as well as security. SiLK provides a set of command-line tools modeled after the standard UNIX command-line tools to analyze the collected data. A typical SiLK workflow consists of a query to retrieve information from a SiLK data repository, which is then piped into a set of SiLK tools to further process the results. The data record format for SiLK is proprietary format, but the data fields are fundamentally similar to the NetFlow v5 record, as SiLK was originally designed to process NetFlow v5 data.

However, this approach left us at the mercy of existing flow meters, such as those deployed on forwarding devices, to generate the flow data on which SiLK operates. Existing solutions had various issues. Flow meters on forwarding devices often lose flows, because high-fidelity flow generation is rightly a lower priority for these devices than forwarding packets. Flow meters using unreliable transport for export also suffer from flow loss, especially during times of high traffic load. In addition, at the time no openly available flow meter had support for the then-emerging IPFIX [6] standard.

YAF (Yet Another Flowmeter) was designed to address this situation. We set out to build a standards-conformant, high-performance, bidirectional network flow meter. Standards-conformance was important to ensure a long operational lifecycle and wide interoperability. We selected the IPFIX standard, based on Cisco NetFlow V9, the successor to the successful de facto standard Cisco NetFlow V5 export protocol. The authors actively participated in the standards process within the IETF to feed our experiences in building and deploying YAF into improving the standard itself, and continue to do so.

Performance was of utmost concern given the scale of the networks we needed to monitor, and the ever-increasing link speeds of the Internet backbone and large enterprise borders. Bidirectionality was important to enable analysis on both sides of a communication, as well

as to slightly increase export efficiency by eliminating redundant information.

The result of this effort is a software tool, `yaf`, which captures live packets or reads packet trace files, and exports IPFIX flows to a collector or to an IPFIX file [25]. It exports IPFIX bidirectional flows [24], and optionally supports a set of additional Information Elements for additional information derived from packet-level or packet payload information, such as TCP initial sequence numbers or payload Shannon entropy.

YAF, in itself, is not a network analysis application or an intrusion detection system. Instead, it is intended as a stage in a comprehensive flow-based measurement infrastructure, with a focus on security-relevant applications.

The rest of this paper is organized as follows. Section 2 describes network flow data, and the various protocols in use for exporting flows, especially IPFIX, and especially as used by YAF. From there we explore the details of the design of YAF in detail in section 3, focusing on those choices which make YAF unique. Related work is described in section 4. We then describe a few existing applications of YAF in section 5, including its application with SiLK [12] within the NetSA Security Suite and its use in the middle tier of PRISM [11], a multi-stage privacy-preserving network monitoring architecture.

2 Network Flow Data: Properties and Protocols

YAF exports flow data. A flow, simply stated, represents a connection between two sockets. More generally and formally, a flow is "a *set of packets* passing an observation point in the network *during a certain time interval* sharing *a set of common properties*, each of which is the result of applying a function to packet, transport, or application header fields; characteristics of the packet itself; or information about the packets treatment." [6]. Specific flow export methods and protocols may use more restrictive definitions than this, for example, by constraining the set of common properties (the flow key) or the method for selecting time intervals. Flows may be unidirectional, in which case they represent one direction of a socket connection, or bidirectional, in which case they represent both directions, or the entire interaction.

The time interval defining a flow generally spans from the first observed packet of the flow to one of three events: either the natural end of the flow, the idle timeout of the flow, or the active timeout of the flow. The natural end of the flow is determined by observing and maintaining the state of the flow for connection-oriented protocols such as TCP or SCTP. The natural end can be determined exhaustively, completely modeling the state machine for the transport layer protocol, or approximately, e.g. by counting a flow as every packet between the first SYN and the first FIN or RST observed for TCP.

The idle timeout of the flow is the longest period of time between packets after which the flow will be considered idle; this is the natural way to expire flows in non-connection-oriented protocols such as UDP. Idle timeouts are generally configurable, and lead to a measurement tradeoff: a short idle timeout leads to faster reaction and lower state utilization during flow metering at the expense of risking expiring flows prematurely.

The active timeout of the flow is the longest lifetime a flow is allowed to have; any flow longer after the idle timeout will be exported, and subsequent packets accounted to a new flow. This is a final backstop against growth of the flow table.

The exact relationship between idle and active timeout and export time is implementation-specific. For example, active timeout can be implemented as a continuous or periodic process; the latter approach leads to some variation in the actual active timeout in the exported data.

The following few sections describes the origin of the IPFIX flow protocol. The discussion is organized from a historical perspective in chronological order.

2.1 Cisco NetFlow v5

Defined by Cisco, NetFlow v5 [4] is a widely deployed de facto standard protocol and raw storage representation for network flow data. It is based on a fixed-length binary record format, with a fixed set of fields. This implies support only for export of IPv4 flows and 16-bit autonomous system numbers, which has led to its being superceded in recent years by NetFlow v9 (see section 2.2), but existing repositories of flow data as well as long replacement cycles of routers which support NetFlow v5 ensure this protocol and representation will be around for some time.

NetFlow v5 is a unidirectional protocol, with the flow meter sending packets via UDP to the collector. It is a "fire-and-forget" protocol; there is no provision for upstream control messages or error reporting, other than that provided by UDP itself via ICMP. This design choice was made to minimize resource usage and state requirements on the flow meter, which in NetFlow v5 is assumed to be a router.

A NetFlow v5 data stream is made up of packets, each of which has a header followed by a number of records. NetFlow v5 records contain start and end timestamps in terms of the reporting line card's uptime in milliseconds, source and destination IPv4 address, source and destination port, protocol, type-of-service, union of all TCP flags in the flow, input and output interface, source and

destination autonomous system number, and source and destination prefix mask length.

The packet header contains the system uptime in milliseconds at export, as well as the system realtime clock at export with nanosecond resolution, which allows flow timestamps to be expressed in millisecond resolution. It also contains a sequence number, which is used to detect dropped NetFlow v5 records.

2.2 Cisco NetFlow v9

Cisco NetFlow v9 [7] is the successor to NetFlow v5, deployed to support IPv6 as well as flexible definition of new record types. It abandons the fixed record format for a template-based system wherein the record format is defined inline. As NetFlow v9 was the base protocol from which IPFIX was developed, the mechanisms it uses are essentially the same as those in IPFIX, though some terminology may be different; therefore, the details of this approach will be elaborated in the following section.

While its flexible data definition makes it nonsensical to speak of a NetFlow v9 record format, and the data exported by Cisco's implementation of NetFlow v9 is administrator-configurable, the information commonly provided in a NetFlow v9 record is more or less equivalent to that available in NetFlow v5.

2.3 IPFIX

IPFIX is a template-based, record-oriented, binary export format. The basic unit of data transfer in IPFIX is the *message*. A message contains a header and one or more *sets*, which contain *records*. A set may be either a template set, containing templates; or a data set, containing data records. A data set references the template describing the data records within that set. This is the mechanism which lends IPFIX its flexibility.

Within the message, each set has a 16-bit ID in its set header. This identifies whether the set contains templates, or data records. In the latter case, the data set ID matches the template ID of the template which describes the records in that data set. A template is then essentially an ordered list of information elements identified by a template ID. An information element (often abbreviated IE) represents a named data field of a specific data type. The data types supported by IPFIX cover the standard primitive types (e.g. unsigned32, boolean) plus additional types for addresses and timestamps; each data type defines an encoding. IEs are then instances of these types, each with its own specific meaning.

IPFIX provides a registry of information elements, administered by IANA [14], that cover most common network measurement applications. This was initially defined in RFC 5102 [18], and is extended both by subsequent IPFIX RFCs as well as by a community process with expert review. Information elements may also be scoped to SMI Private Enterprise Numbers; these can be used to export information (as by YAF) not suitable for standardization through the IANA process.

Because Templates are generally exported once per session, the cost of self-representation is amortized over many records. In this way, IPFIX can support a wide variety of record formats, avoiding tying the implementation of a flow meter to a specific export data structure, without the overhead of other representations with semantic flexibility per record, e.g. XML. This extensibility allows innovation in flow metering and export, and as such was the natural choice for YAF.

2.3.1 As exported by YAF

As shown in 2, YAF can export an extensive set of fields, a superset of those available in earlier NetFlow versions, omitting those specific to packet-forwarding devices. Many of these are IPFIX-standard fields defined in the IANA registry, while others (those with an annotation in the "YAF-specific" column) are enterprise-specific Information Elements defined specifically for YAF.

YAF also takes extensive advantage of IPFIX's template mechanism to enable efficient export, as detailed in section 3.4. As shown in the "Present when" column in table 2, YAF exports IPv4 addresses only when the flow is an IPv4 flow, and IPv6 addresses only when the flow is an IPv6 flow. Reverse information elements are only exported for flows which actually have packets in the reverse direction. In addition, command-line arguments enabling various additional features of YAF at runtime (e.g. DPI, entropy calculation, and others to be described later in this work) cause YAF to capture that data and add information elements to its export templates to represent them. Each exported record contains only the information elements it needs, with YAF selecting the appropriate template at runtime, exporting it if it has not yet been exported, and starting the export of a new Data Set if necessary.

3 Detailed Design of YAF

YAF is designed as a bidirectional network flow meter. At its core, it takes packet data from some source, decodes it, and associates the packet data with a flow. When flows are determined to be complete, it exports them. This is a rather simplified view, to which we will add some more detail in the following subsections.

First we follow a packet through the various stages of the basic YAF workflow shown in 1, from capture through to export. Then we examine other interesting

Name	Present when	YAF-specific
flowStartMilliseconds	always	
flowEndMilliseconds	always	
octetTotalCount	always	*may use reduced-length encoding*
reverseOctetTotalCount	biflow	*may use reduced-length encoding*
packetTotalCount	always	*may use reduced-length encoding*
reversePacketTotalCount	biflow	*may use reduced-length encoding*
sourceIPv6Address	IPv6	
destinationIPv6Address	IPv6	
sourceIPv4Address	IPv4	
destinationIPv4Address	IPv4	
sourceTransportPort	always	*may contain ICMP type/code*
destinationTransportPort	always	
protocolIdentifier	always	
flowEndReason	always	*may contain SiLK-specific flags*
silkAppLabel	–applabel	DPI application label
payloadEntropy	–entropy	Shannon payload entropy
reversePayloadEntropy	biflow –entropy	Shannon reverse payload entropy
mlAppLabel	–mlapplabel	Machine-learning app label
reverseFlowDeltaMilliseconds	biflow	RTT of initial handshake
tcpSequenceNumber	TCP	
reverseTcpSequenceNumber	TCP biflow	
initialTCPFlags	TCP	TCP flags of first packet
unionTCPFlags	TCP	TCP flags of $2..n^{th}$ packet
reverseInitialTCPFlags	TCP biflow	TCP flags of first reverse packet
reverseUnionTCPFlags	TCP biflow	TCP flags of $2..n^{th}$ reverse packet
vlanId	–mac	
reverseVlanId	–mac	
ingressInterface	–live dag multi-IF	
egressInterface	–live dag multi-IF	
osName	–p0fprint	p0f Operating System name
osVersion	–p0fprint	p0f Operating System version
reverseOsName	biflow –p0fprint	p0f reverse Operating System name
reverseOsVersion	biflow –p0fprint	p0f reverse Operating System version
firstPacketBanner	–fpexport	First forward packet IP payload
reverseFirstPacketBanner	biflow –fpexport	First reverse packet IP payload
secondPacketBanner	–fpexport	Second forward packet IP payload
payload	–export-payload	First n bytes of application payload
reversePayload	biflow –export-payload	First n bytes of reverse application payload

Table 2: Information elements in a YAF record

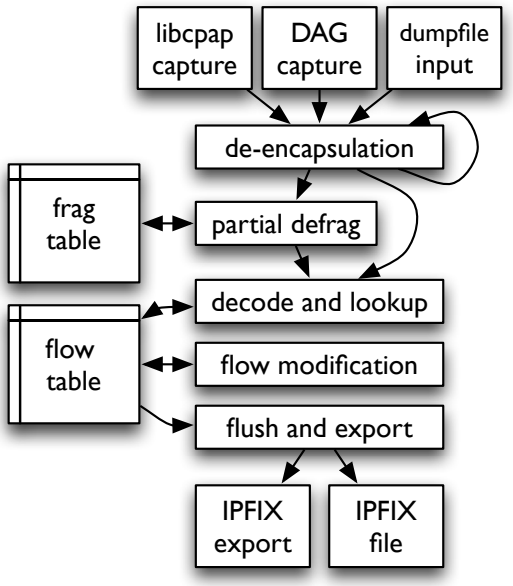

Figure 1: Basic Data Flow in YAF

aspects of YAF's design, and additional optional features it supports compared to other flow meters.

3.1 Recursive De-encapsulation

Packet data input can come from a variety of sources, including `libpcap` dumpfiles, live capture on commodity interfaces via `libpcap` as well as specialized devices with `libpcap`-compatible interfaces such as Bivio and Napatech devices, and Endace DAG cards. Each of these sources generally yields Layer 2 and above information; YAF then recursively unwraps encapsulations to arrive at an IP header, possibly storing certain information (e.g., VLAN tags or MAC addresses) for later export with the flow. In addition to the ubiquitous Ethernet encapsulation, YAF also supports a variety of less common, carrier-use encapsulations. YAF can decode GRE, MPLS, MPLE, PPPoE, cHDLC, Linux SLL, PPP, and PCAP raw. Running on appropriate hardware, this allows YAF to decode network information from Ethernet to DS3 links, to OC-192 connections. Additionally, as depicted in diagram 1, YAF can also decode odd combinations of encapsulation by running through the encapsulation phase multiple times. For example, one site running YAF encapsulates Ethernet over MPLS. Additionally, YAF is constructed to allow new encapsulations to be cleanly added. The decoding system requires only minimal modification to support a new encoding. YAF relies on the capture system to be able to identify the base encapsulation.

3.2 Decoding

De-encapsulated packets are passed to the Layer 3 and 4 decoding layer, which extracts flow keys and counters from the packet data. The flow key determines which flow the packet belongs to, and in YAF consists of the traditional "5-tuple" (source and destination address, source and destination port, protocol) as well as the IP version number (4 or 6) if YAF is compiled for dual-stack support. The flow key may also optionally include the VLAN tag and, in the case of a DAG card as source, the DAG interface number on which the packet was captured. This flow key is used for lookup in the flow table.

3.3 The Flow Table

The YAF flow table is implemented as a hashtable-indexed pickable queue. This data structure is essentially a queue paired with a hashtable. It allows random access to any entry in the flow table via the hashtable, but also constant access to the least-recently-seen entries, which allows efficient timeout of idle flows. This design evolved in part from the bin queue used in NAF [26].

The flow key calculated from the decoding stage is looked up in the flow table's hashtable. If no active flow record corresponding to the flow key is found, a new record is created. Regardless, the flow record is modified with information from the packet (e.g., counters, payload and payload-derived information), and moved to the head of the flow table's pickable queue to implement idle timeout. Active timeout is evaluated when each flow is selected: if a packet belongs to a flow that is older than the active timeout interval, that flow is removed from the flow table and exported, and a new flow record is created for the incoming packet.

From this point on, the YAF data flow operates on flows only.

Since YAF flow records in memory are all equal size, and they have variable lifetimes, they are allocated using a slab allocator [3], which allows fast reuse of expired flow records. This gives YAF additional performance over true dynamic allocation, but still allows the flow table to grow and shrink with variable traffic load unlike with a statically-allocated table. However, since the slab allocator never returns memory to the operating system, its memory footprint will generally not be reduced during low-traffic periods. Growth of the flow table can be controlled by command-line options setting the idle and active timeouts as well as the target maximum table size, which dynamically reduces the timeouts in order to prevent resource exhaustion during traffic bursts or intentional denial-of-service attacks against the flow meter.

During the long transition from IPv4 to IPv6, a sufficiently large and complex organization may use both

protocols for some time; therefore, a design goal of YAF is to be able to support measurement of both IPv4 and IPv6 traffic efficiently, with efficient runtime storage and export of both IPv4 and IPv6 flows from the same interface.

If YAF is compiled with IPv6 support, it will dynamically create either an IPv4 or IPv6 flow table entry based upon the protocol of the flow. IPv4 and IPv6 flows are defined as overlaid C structures, so most of the YAF code for handling them that does not handle flow table entry allocation or endpoint addresses treats the two flow types equally. From the slab allocator's point of view, this is like having two separate flow tables, but both IPv4 and IPv6 flows are unified in the same pickable queue. This feature comes at the cost of some additional memory to store the overlaid structure and some delay in selecting flow type at flow creation compared to an IPv4-only YAF, but much less memory than would be required if IPv4 and IPv6 flows were stored in a single `union` data type with enough space for the larger addresses. Efficient template selection as in section 3.4 below minimizes export bandwidth penalty for dual-stack support.

3.4 Efficient Export and Template Selection

When a flow ends, whether through natural completion (presently supported only through the TCP FIN handshake) or idle or active timeout, it is ready for export.

Though YAF exports what is semantically one type of data, it uses multiple IPFIX templates to maximize export efficiency. As mentioned in section 2.3, IPFIX Templates are identified by a 16-bit number. YAF essentially uses some of these bits as flags in order to enable or disable fields in the Template used to export each flow, based on the flow's characteristics. The characteristics used for template selection are:

- whether the flow requires full-length (64-bit) counter export, or can be represented with 32-bit packet and octet counters (reduced-length encoding)
- whether the flow is an IPv4 or IPv6 flow
- whether the flow is a biflow (it has at least one packet in the reverse direction)
- whether the flow is a TCP flow
- whether layer 2 (MAC and VLAN) export is enabled
- whether the flow was captured on a DAG card, and has DAG interface information
- whether the flow has an application label
- whether the flow has entropy information
- whether the flow has a p0f fingerprint
- whether the flow has payload, and payload export is enabled

Compare these characteristics with the record structure in table 2.

When a record is ready to be exported, YAF selects a template by deriving a template ID from the properties of the flow table entry and the configuration of the YAF instance. If this template ID corresponds to a template that has not yet been exported, it exports the template; if it doesn't match the that of the last exported record, it starts exporting a new IPFIX set.

For example, a short IPv4 UDP flow with no reverse direction will be exported using a template containing IPv4 address elements, no reverse elements, no TCP elements, and reduced length counters.

When a flow is exported, YAF forgets about it, and its entry is recycled by the slab allocator.

This concludes our trip through the "normal" YAF workflow; subsequent subsections handle YAF's design approach to particular caveats of flow metering, or optional features supported by YAF.

3.5 Just Enough Defragmentation

IP packet fragmentation causes a problem for flow metering. Some implementations of flow assembly, especially those on resource-constrained devices or on devices where flow metering is a secondary, lower-priority function (e.g., routers), simply ignore fragmentation. For TCP or UDP, these would treat the first four bytes below the IP layer as the source and destination port of the flow regardless of whether the packet contained the first fragment; all fragments other than the first per packet are accounted to the wrong flows. While this may be acceptable for some applications, given the relatively low prevalence of fragmented traffic on the Internet [21], it presents a simple attack against any flow meter ignoring fragmentation: most packets can be accounted to the wrong flows simply by aggressive fragmentation.

At the same time, full fragment reassembly is resource-intensive, especially when most of the information stored and reassembled will then be discarded, as is the case with flow key extraction from fragments.

For this reason, YAF supports just enough defragmentation: a fragment table designed very much like the flow table (i.e., using pickable queues, slab allocation, and dynamic timeouts for defense against resource overuse) which keeps track of the flow a fragment belongs to, and defragments only enough payload per flow to support the

other features selected at runtime. Defragmentation occurs between de-encapsulation and decoding.

Defragmentation is enabled by default, but can be disabled at runtime to save resources (e.g., on networks where fragmented packets are known not to be present) or for compatibility with flow meters not supporting defragmentation.

3.6 Packet Clock

YAF is designed to accept data both from live capture as well as from files containing ordered packet traces. For that reason, designed into the core of YAF is the concept of the packet clock: YAF in effect "pretends" that the current time is the timestamp of the packet it is presently processing. This implies that timeouts are evaluated against the data, and not against the system realtime clock of the machine running YAF. This is important to ensure repeatability: that the same packet trace processed multiple times will lead to identical output data, as well as that YAF will produce identical data whether from live capture or from playback.

3.7 Per-flow Payload Capture

If so configured at compile-time, YAF supports per-flow payload capture. Payload capture is limited to the first n bytes of each flow, configured on the command-line, in order to provide the administrator control over YAF's resource consumption; payload capture can significantly increase YAF's requirements. Payload capture for TCP flows provides full TCP reassembly.

Since each YAF record must fit within an IPFIX record, and IPFIX imposes as 65515-byte content limit on records, this maximum exportable payload is somewhat under 64kB for uniflows and somewhat under 32kB for biflows. As well as supporting direct export of flow payload, YAF payload capture can be used to support entropy calculation and application identification, as described in the following subsections.

3.8 Flow Payload Entropy

YAF can calculate the Shannon [22] entropy of the captured payload for use in understanding the nature of the traffic within a flow. The Shannon entropy is calculated by scanning through the first n bytes of captured payload (the "banner") byte-by-byte and creating a histogram distribution within a 256-entry array. The partial entropy of each histogram value x is then summed to compute the total entropy H as follows:

$$H = \sum_{i=0}^{255} \frac{x_i}{n} * \frac{\log x_i}{\log 2.0}$$

The entropy is scaled to the range 0-255, for single byte export, as follows:

$$H_{export} = -1 * \frac{H}{8.0} * 256$$

The same operation is done for the reverse payload to generate the reverse entropy.

This method of entropy calculation results in a measure of entropy in bits per byte (log base 2); i.e., a number between 0 and 8 in 8-bit fixed-point representation. In pure terms, a value of 255 would indicate a *perfectly* random set of data, while a value close to 0 would indicate an extremely redundant set of data with almost no information content. High entropy values indicate data that is either compressed or encrypted. Lower values likely to indicate something such as an ASCII-based protocol, or English text.

As a guide to the actual usage of the numbers from YAF, values above approximately 230 indicate compressed or encrypted traffic. Values centered around 140 are likely to be English text. A quirk in SSL/TLS is that it commonly zeros its packets out before sending them. This leads to extremely low numbers often indicating an SSL/TLS encrypted flow.

3.9 Application Labeling

YAF can analyze the banner on each flow that it captures in order to recognize the protocols above layer 4 in captured flows, and to label each flow with the application it is running. Labeling runs at flush-and-export time (i.e., once the flow payload is known to be complete). YAF independently evaluates each direction of a biflow during labeling. Application labeling is designed to be transport port neutral in recognizing the protocol being used; however, port information is used as a hint to which protocol match to attempt first, for efficiency purposes. For example, if YAF has captured a biflow from two hosts with ports 5238 and 80, YAF will attempt to match HTTP first, due to the presence of port 80 in the flow. Application labeling is first match wins; in the example above, the flow would be labeled HTTP and labeling would stop.

YAF application labeling currently recognizes the following protocols: HTTP, SSH, SMTP, Gnutella, Yahoo Messenger, DNS, FTP, SSL/TLS, SLP, IMAP, IRC, RTSP, SIP, RSync, PPTP, NNTP, TFTP, MySQL, and POP3. Additional protocols are actively being added. Areas of future work include an experimental integration of the OpenDPI project [16], including the capabilities of that engine as a plug-in. Additionally, it is possible for users to add some recognizers for some simple protocols simple by text editing a configuration file, which is described below.

3.9.1 Implementation Details

YAF uses multiple mechanisms for protocol recognition. For some network protocols, e.g. DNS, YAF includes a shared library written in C which can decode the structure of the DNS packets. The DNS protocol uses a well-formatted binary structure making the tests for determining a valid DNS packet relatively easy in C. Additionally, when exporting the extended packet details as in 3.9.2, the C library has the added advantages of handling the binary fields easily. In addition for cases like DNS implementing name "decompression" is also relatively easy.

Conversely, other protocols, such as SMTP, IMAP, and SIP, are ASCII or UTF-8 based protocols; these lend themselves instead to textual analysis based on regular expressions. YAF provides a method for defining labeling of these protocols based on the widely used PCRE engine [13]. This provides two distinct mechanisms for protocol recognition within YAF. This creates an advantage of allowing application labeling to be applied using two different mechanisms for the two different types of protocols. Another advantage of using a regular expression system is that it allows easy in-the-field experimentation, without recompiling anything, to find new protocols.

In order to illustrate how the recognition system is configured, a small sample of the application labeler configuration file is shown:

```
# HTTP
label 80 regex HTTP/\d\.\d\b
# SSH
label 22 regex ^SSH-\d\.\d
# SMTP
label 25 regex (?i)^(HE|EH)LO\b
# DNS
label 53 plugin dnsplugin \
         dnsplugin_LTX_ycDnsScanScan
```

The structure of the entries is keyword `label` followed by the port number. This port number is used as both the label that YAF will put into the record it produces as the output record as well as the port number for hinting based on the source and destination ports. The next keyword is either `regex` for a regular expression based rule or `plugin` for a C-callable plugin. In the case of the `regex` expression, everything beyond the `regex` keyword will be interpreted as the regular expression.

The C plugin requires a set of functions to be defined and a standard naming convention to be used. There are 11 functions to be defined required of every plugin, and optionally, a twelfth function used for the deep packet inspection. These functions are defined in the source file `yafhooks.c`. Advanced users of YAF may also be capable of implementing a YAF extension in this way.

As previously mentioned the label which YAF applies is defined as the primary port number on which the application is expected to be seen. This is also used at runtime to determine the most likely matching application for a given flow, based on the source and destination ports of the flow. If there is no match, the rules are evaluated in order, which allows performance tuning by ordering the rules in the order of their expected precedence on the network. However, regular expression-based flow labeling still presents a performance risk, and users should take care that regular expressions are kept simple in order to minimize negative impact on performance.

3.9.2 Extended Application Information Export

YAF can export extended information about a large number of protocols in its application labeling capability. Many of these fields are relatively innocuous and detail the general workings of the network and its protocols. However, some fields may contain sensitive information. Depending on jurisdiction, captured payload data or data derived from payload capture may be considered Personally Identifiable Information (PII), such that turning on these features may require special handling of the flow record output.

As previously mentioned, the recently released YAF 1.2 can identify via deep packet inspection (DPI) the following protocols: HTTP, SSH, SMTP, Gnutella, Yahoo Messenger, DNS, FTP, SSL/TLS, SLP, IMAP, IRC, RTSP, SIP, RSYNC, PPTP, NNTP, TFTP, Teredo, MySQL, and POP3. In addition to identifying those protocols, YAF may (or may not) export extended information depending on the various protocols. YAF will currently export extended information about the following protocols: HTTP, SSH, SMTP, FTP, IMAP, RTSP, SIP, and DNS. Futhermore, we collect user name type information in SMTP, FTP, IMAP, and SIP.

As an example of the extended information collected, we will consider a single protocol, SIP [19]. For SIP messages, YAF will optionally capture, identify, and export the following fields: the `Via`, `Max-Forwards`, `To`, `From`, `Contact`, and `Content-Length` headers, as well as the SIP method.

3.10 Performance

YAF was designed and developed with the goal of building a high-performance flow meter, while still maintaining a cleanliness of design allowing future maintainability and extensibility. Performance is measured throughout development and maintenance using profile-based measurement tools such as Shark and Instruments on the Mac OS X development platform.

YAF's performance depends as well upon the performance of the underlying fixbuf IPFIX library, since IPFIX transcoding on export is a significant portion of the work YAF does. fixbuf is also designed to be a high-performance IPFIX implementation, operating on in-memory buffers containing messages and records, and exploiting natural alignment of data structures in order to speed the rearranging and copying from application internal data structures to IPFIX records on the wire.

Subject to the capabilities of the underlying capture device, YAF is tested for performance using various carrier line speeds and encapsulations during live capture. It is tested on Ethernet systems from 100Mbit to 10Gbit, and on optical carrier lines from OC-3 to OC-192, using generated traffic from a dedicated load generator.

YAF is designed to perform well on generic PC hardware running most UNIX variants as well as Apple OS X. For many types of links, a capable PC will be sufficient. YAF is also tested and tuned to run on various custom capture systems including Endace DAG capture cards and Bivio appliances. A future release of YAF will also be developed with specific enhancements for Napatech capture cards.

4 Comparing YAF to Other Flowmeters

The name Yet Another Flowmeter plays on the old UNIX-community joke of prefixing the nth instance of a particular type of tool "Yet Another x"; however, we designed YAF to meet a combination of requirements that were at the time not generally available: we needed high performance, easy extensibility of both the output record format and the flow-level measurement capabilities, and compliance to an emerging standard to ensure a long operational lifetime. Here we compare YAF to existing flow meters, or flow-meter-like systems.

4.1 Software NetFlow Meters/Exporters

While there is a wide variety of both free and commercial software designed to operate as NetFlow collectors and analyzers, there is a smaller number of available NetFlow meter/exporters, which generate flow data from packet data and export via NetFlow or IPFIX. Two popular examples are softflowd [15] and nProbe [8].

`softflowd` does semi-stateful assembly of flows and export via NetFlow v5 and v9; a related tool `pfflowd` uses the OpenBSD packetfilter flow table instead. It supports raw capture from `libpcap` only. It was designed to be fast and simple, and as such supports none of YAF's flexibility or advanced features. Development appears to have been inactive since 2006.

nProbe is an "all-in-one" tool for handling flow data as part of the nTop [9] network measurement suite. One of the features it provides is flow generation and export from packet data. Its feature set is much more comparable to YAF's: it supports IPFIX export, high-speed collection from dedicated capture cards such as Endace DAG and Napatech devices, and protocol inspection . In addition, it does a few things YAF doesn't: operating as an IPFIX Mediator to translate older NetFlow versions to IPFIX, and storing flow data directly into MySQL or sqlite databases, for example. Development is active as of summer 2010.

YAF and nProbe have to some extent been developed in parallel; features showed up in one or another at roughly the same time, and the authors indeed tested the interoperability the underlying IPFIX export implementation as early as 2006. However, in contrast to YAF, while nProbe is published under the GNU GPL, it is not generally freely available, with source download behind a donation paywall and limited mirroring of older versions.

4.2 Argus

Argus [17] is a flow meter and analysis toolkit in a single set of tools. While Argus does contain a set of powerful analysis tools, similar in some ways to SiLK, that is beyond the scope of this paper. Instead, here we focus on its flow measurement and export protocol type.

Argus is designed to measure bidirectional flows in the network control plane. In order to complete that task, Argus will attempt to merge relevant control plane information into a control flow via its monitoring points independent of the link types monitored. For example, if the goal is to monitor a high-performance cluster system using Infiniband, Argus can monitor the control plane on the cluster as well as the external link running an Ethernet or optical carrier connection. In addition to monitoring both of those links, Argus will attempt to match a DNS query on the external link with activity on the Infiniband connection.

Argus is unique in its fundamentally "philosophically" different approach to flow metering. While Argus attempts to merge and relate flow information at the sensor from related flows, YAF and most other flow sensors attempt to capture the flow information in an IPFIX standard way and allow the back end analysis tools, such as SiLK or others, find the relationships among the various flows.

Argus also has a proprietary "sensor-centric" export protocol in contrast to IPFIX and its predecessors, and provides flexibility at the record level as opposed to the informaiton element level. Argus "clients" (collectors) initiate connections to the sensor and pull flow data off the sensor.

Each approach has its pros and cons. In our appli-

cations, flow meters are generally deployed in environments where inbound connections are forbidden, and where software maintenance is often difficult; meters must be stable and not change very often; therefore, we prefer to centralize the harder correlation work, and any analysis which may see further innovation, while keeping the edge fast and stable, and the bandwidth from the edge to the data center as small as practical.

4.3 sflow

sflow [23] is a protocol which takes a different approach to the same problems solved by flow collection and analysis. The key design decision here is that attempting full flow assembly on high rate packet data requires too many resources, and for many applications (e.g., traffic matrix generation) sampled packet data is sufficient. Despite the "flow" in the name, sFlow is not a flow metering or export technology: it simulates flows with packet sampling. There is a free reference implementation, as well as support in nTop in addition to switches and routers from a variety of manufacturers [20].

4.4 Netflow, Flexible NetFlow, and other router-based approaches

The key difference between YAF and flow metering processes running on routers is one of application: NetFlow and related technologies run as secondary processes on routers, and as such an important design consideration is that packet forwarding performance must not be impeded by flow metering. This leads to reduced data fidelity during peak traffic times, as the router allocates its limited resources to forwarding instead of monitoring. When high data fidelity in flow metering is a primary requirement, such as for security, flow meters such as YAF can be used on a switch span port or optical tap to offload the task from the router.

A key difference between YAF and Flexible NetFlow is that, although both utilize the template functionality in IPFIX or NetFlow V9, YAF uses it only for export efficiency while Flexible NetFlow uses it for flow key flexibility: it exports different record types, aggregating packets into flow records on other than the traditional flow key. In this way it is more akin to YAF followed by aggregation operations in SiLK, or NAF [26].

5 Applying YAF

YAF was initially released in 2006, although it was marked as an alpha-quality release for quite some time. The YAF 0.7 release of August 2007 market the first release ready for operational deployment. Since then, YAF has been adopted by several organizations as their main software flow meter platform, and has been in production use for quite some time. YAF is still used in the experimental side of network flow analysis as well. In the following two sections we will describe our experiences using YAF for those various purposes.

5.1 YAF and the Security Suite

SiLK [12] is designed to allow very large scale collection and analysis of network flow data. It provides a set of command-line tools modeled after the standard UNIX command-line tools (e.g. sort(1), uniq(1), cut(1)) to analyze the collected data. A typical SiLK command-line to query a SiLK data repository is then piped into another set of SiLK tools to further process the results. The data record format for SiLK is a proprietary format, but the data fields are fundamentally similar to the NetFlow v5 record, as SiLK was originally designed to process NetFlow v5 data.

Large SiLK deployments include one with more than 50 geographically distributed sensors monitoring everything from DS3 to 10 gigabit Ethernet links. In this case, YAF is run on one of two types of sensors: one a Linux-based PC server with an Endace DAG card installed, for monitoring a DS3 link; the other a Bivio 7500 series appliance using multiple blades to listen to a 10 Gigabit Ethernet link.

Each deployed sensor sends data back to a centralized data center via secure, encrypted connection. At that data center, each record from the sensors is collected, tagged, and stored onto a large SAN system for analysis by various analysis groups. This system collects many tens of gigabytes of flow data per day, allowing analysts to see the large-scale picture of network activity occurring across the largest of enterprise networks.

A typical configuration for a PC with an Endace card installed is to have YAF start at system boot. For this purpose, YAF includes a set of startup scripts that can be installed on a typical Linux system to manage YAF. These scripts start YAF via the included `airdaemon` utility, which ensures that YAF will restart on abnormal shutdown due to hardware issues. YAF then typically exports via IPFIX to a local instance of the `rwflowpack` process, part of the SiLK packing system. `rwflowpack` then packs the received IPFIX record into a SiLK format, and then compresses the records to make them smaller still. After the records are fully compressed to an average of about 15 bytes per flow, `rwflowpack` passes the records to `rwsender` for transmission back to the data center.

YAF's enhanced flow metering has been applied in production in combination with SiLK in order to solve operational problem. As a simple example, flows containing traffic running on nonstandard ports can be de-

tected simply by enabling application labeling, then running an `rwfilter` query to compare the application label (exported in the `silkAppLabel` information element) with the ports in the flow. Enhanced information can also be applied to inventory problems: DNS support has been used to identify and patch DNS servers vulnerable to the Kaminsky [10] exploit.

5.2 PRISM: Data Reduction in a Multi-Stage Monitoring Architecture

YAF was also deployed in 2010 as part of the integrated trial of the European Union Seventh Framework PRISM [11, 2] project at a regional network service provider in Italy. The aim of the PRISM project is to apply semantically-aware access control, a multi-stage monitoring architecure, aggressive data reduction, and data protection and anonymisation techniques to enable privacy-aware and privacy-preserving network monitoring. The PRISM architecture is split into front-end (FE) components which observe a packet stream and reject packets unlikely to be of interest, back-end (BE) components which further reduce, analyze, and store data received from the FE, and beyond-the-back-end (BBE) components.

A key insight of the PRISM project is that data reduction, such as reducing a packet stream down to a flow stream early in the monitoring pipeline, removes potentially privacy-relevant information; in the case of flow data generation, the elimination of payload data significantly reduces the potential privacy impact of the content in the observed data stream.

YAF was used as one of the data reduction components in the back-end of the integrated trial for a Skype-detection application. In this scenario, higher levels of privilege (e.g., a network administrator debugging a specific connection issue for a customer) would allow full dissection of the Skype traffic from one host to another, using a packet-based Skype analyzer [1] beyond-the-back-end, while a lower level of privilege (e.g., a junior administrator preparing a report on the volume of Skype traffic on the network) would use a flow-based method, reducing the fidelity of data seen beyond the back-end. The PRISM access control system would automatically select the correct reduction component based upon the privilege and purpose of the request.

In this deployment, YAF was used as a straight flow meter - payload inspection and capture were specifically disabled. YAF was invoked on `libpcap` dumpfiles generated by the `capfix` utility developed as part of the PRISM project, which provides for the framing of packet traces in IPFIX/PSAMP, and configured to export to a `snack` instance beyond the back-end, which in turn generated a list of Skype connection events used to count distinct hosts running Skype.

YAF was selected for this application due in large part to the effortlessness of integration. PRISM had selected a data plane based entirely on IPFIX early in development, in order to maximize the potential to leverage off-the-shelf standards-compliant components within the PRISM architecture. When the project decided on a split packet/flow based approach to Skype traffic analysis, YAF was an obvious choice for the flow meter, and its selection reduced the implementation of this stage of the data plane to simply writing a little glue.

6 Availability

YAF is distributed by the Software Engineering Institute as free software under a dual-license system. The general public may download YAF (and all of the Network Situational Awareness team's software, including SiLK) from

`http://tools.netsa.cert.org`

under the terms of the GNU General Public License version 2. The US government maintains separate rights in the software, and may use the software under the terms of the Government Purpose License Rights of DFARS. Support for building and using YAF and all the Network Situational Awareness tools is available by sending email to

`netsa-help@cert.org`

YAF should work on most flavors of Unix, and is developed and tested on Mac OS X, Linux, FreeBSD, OpenBSD, and Solaris.

7 Acknowledgments

The authors would like to express their deep gratitude to the members of the Network Situational Awareness team at CERT for helping build and support YAF, especially Emily Ecoff, Michael Duggan, and Tony Cebzanov. Funding and support for the development of YAF has come from the United States Department of Defense, which supports the Software Engineering Institute as a federally funded research and development center.

References

[1] ADAMI, D., CALLEGARI, C., GIORDANO, S., PAGANO, M., AND PEPE, T. A real-time algorithm for Skype traffic detection and classification. In *9th International Conference on Wired/Wireless Networking* (Sept. 2009).

[2] BIANCHI, G., TEOFILI, S., AND POMPOSINI, M. New directions in privacy-preserving anomaly detection for network traffic. In *NDA '08: Proceedings of the 1st ACM workshop on Network data anonymization* (New York, NY, USA, 2008), ACM, pp. 11–18.

[3] BONWICK, J. The slab allocator: an object-caching kernel memory allocator. In *USTC'94: Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference* (Berkeley, CA, USA, 1994), USENIX Association, pp. 6–6.

[4] CISCO SYSTEMS, INC. Cisco IOS Netflow Introduction. http://www.cisco.com/go/netflow. [Accessed 19 August 2010].

[5] CISCO SYSTEMS, INC. Hyperconnectivity and the Approaching Zettabyte Era. Cisco VNI White Paper, June 2010.

[6] CLAISE, B., BRYANT, S., LEINEN, S., DIETZ, T., AND TRAMMELL, B. Specification of the IP Flow Information Export Protocol. RFC 5101 (Proposed Standard), Jan. 2008.

[7] CLAISE, B., SADASIVAN, G., VALLURI, V., AND DJERNAES, M. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), Oct. 2004.

[8] DERI, L. nprobe – netflow/ipfix network probe. http://www.ntop.org/nProbe.html, Oct 2006. [Accessed 9 August 2010].

[9] DERI, L., AND SUIN, S. Effective traffic measurement using ntop. *IEEE Communications Magazine* (May 2000), 138–143.

[10] DOUGHERTY, C. R. "Vulnerability Note VU#800113 multiple DNS implementations vulnerable to cache poisoning". http://www.kb.cert.org/vuls/id/800113, 2009. [Accessed 27 July 2010].

[11] FP7 PRISM PROJECT. PRIvacy-aware Secure Monitoring. http://www.fp7-prism.eu. [Accessed 6 August 2010].

[12] GATES, C., COLLINS, M., DUGGAN, M., KOMPANEK, A., AND THOMAS, M. More netflow tools for performance and security. In *LISA '04: Proceedings of the 18th USENIX conference on System administration* (Berkeley, CA, USA, 2004), USENIX Association, pp. 121–132.

[13] HAZEL, P. PCRE - Perl Compatible Regular Expressions. http://www.pcre.org.

[14] INTERNET ASSIGNED NUMBERS AUTHORITY. IP Flow Information Export (IPFIX) Information Elements. http://www.iana.org/assignments/ipfix/.

[15] MILLER, D. softflowd - fast software netflow probe. http://www.mindrot.org/projects/softflowd, Oct 2006. [Accessed 9 August 2010].

[16] OPENDPI. http://www.opendpi.org. [Accessed 6 August 2010].

[17] QOSIENT, LLC. Argus: Auditing Network Activity. http://www.qosient.com/argus/. [Accessed 19 August 2010].

[18] QUITTEK, J., BRYANT, S., CLAISE, B., AITKEN, P., AND MEYER, J. Information Model for IP Flow Information Export. RFC 5102 (Proposed Standard), Jan. 2008.

[19] ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., AND SCHOOLER, E. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630.

[20] SFLOW.ORG. sflow products - network equipment. http://www.sflow.org/products/network.php, 2010. [Accessed 9 August 2010].

[21] SHANNON, C., MOORE, D., AND CLAFFY, K. C. Beyond folklore: observations on fragmented traffic. *IEEE/ACM Trans. Netw. 10*, 6 (2002), 709–720.

[22] SHANNON, C. E. A mathematical theory of communication. *Bell System Technical Journal 27* (Jul and Oct 1948), 379–423,623–656.

[23] STEENBERGEN, R. A. sflow – why you should use it and like it. In *39th Meeting of the North American Network Operator's Group (NANOG 39)* (Feb 2007).

[24] TRAMMELL, B., AND BOSCHI, E. Bidirectional Flow Export using IP Flow Information Export. RFC 5103 (Proposed Standard), Jan. 2008.

[25] TRAMMELL, B., BOSCHI, E., MARK, L., ZSEBY, T., AND WAGNER, A. Specification of the IP Flow Information Export File Format. RFC 5655 (Proposed Standard), Oct. 2009.

[26] TRAMMELL, B., AND GATES, C. NAF: the NetSA Aggregate Flow Tool Suite. In *20th USENIX Large Installation System Administration Conference (LISA '06)* (Dec 2006), pp. 221–231.