INVITED PAPER *Special Section on Deployment and Operation of New Internet Technology: Challenges and Approaches*

Flow-Based Measurement: IPFIX Development and Deployment

Nevil BROWNLEE[†a)], ***Nonmember***

SUMMARY This paper presents a history of the IPFIX working group, from initial chartering through development and testing, re-chartering and further development, and looking ahead to future developments. As a standardised way of exporting information about traffic flows, IPFIX has attracted a growing number of implementors, who have continued to develop it in useful ways without changing its underlying architecture. This makes it a good example of how to develop a new technology. Further, it demonstrates widespread recognition of the importance of network measurement in the development, deployment and production stages of networks and the applications that depend on them.

key words: *traffic flows, network data export, IPFIX*

1. Introduction

From time to time every organisation will need to deploy or modify a system to achieve some important goal. For example, a company might need a new online ordering system, a news organisation might upgrade their web servers, or an Internet Service Provider (ISP) might upgrade their routers or link capacities. When that happens, it's important that the system is monitored, and that its performance is *measured.*

Again, measurement is needed during every part of a system's life. We begin by deciding what characteristics of the system will be most useful for indicating its behaviour; for example, an ISP may need to know how many packets or bytes flow through a link each minute, the news company needs to know how many requests its system handles each second, how long it takes before a user sees a response to her mouse clicks, and so on. During system development, we use our measurements to determine what happens when we change system parameters. While testing we check to see that everything is working as it should. In production we may need to verify that performance is within the boundaries set by the system's SLA. In short, measurement is vital for any operational system — *if we don't measure, we don't know what's happening!*

Values of system characteristics can be measured at Observation Points within one or more hosts, routers or switches, either by the host operating system (for interface- network- and transport-level characteristics), or within applications running on the host(s). We need to be able to gather that data from each host so that it can be archived and analysed. Network management and monitoring, and any applications that are distributed across many hosts, will need to collect data from every host involved in the system.

Once our data is measured, we need a system to carry the measurement data from the hosts where it's generated to our monitoring and analysis points. One method for doing that is the Simple Network Management Protocol, SNMP. SNMP uses a large set of *MIBs* for various kinds of devices; these are essentially databases that contain a tree of Objects and their current values. For example, interface packet and byte counts show the total number of packets and bytes that have passed in or out through an interface. Values of SNMP Objects can be read by a management or monitoring system, e.g. MRTG, an Open Source system that reads SNMP data, archives them and plots them using various timescales.

Another widely-used method of monitoring Internet traffic is *flow analysis,* which is based on grouping packets into sets that have common properties. This field's seminal paper [1] was published in 1995; it used the set {*IP protocol, source IP address, source port, destination IP address, destination port*} as the common properties, together with a fixed timeout period. Such flows are usually described as *5-tuple flows;* they have the same transport-layer start and end points, are unidirectional and time out if no new packets are seen for 60 seconds. A more general approach to flow measurement was taken by the IETF's RTFM working group in 1999. RTFM [2] generalised flows, allowing a larger set of 'flow attributes' to be used to specify flow endpoints. RTFM also introduced bidirectional flows, having two sets of attributes one for the *forward* (source-to-destination), the other for the *reverse* direction. Flows provide a useful picture of Internet traffic for network operators. For example, the largest flows indicate the most-active ('heavy-hitter') users, and ISPs can use source/destination-based flows to produce a traffic matrix, allowing them to see the busy paths within their network.

One example of a flow measurement system is Cisco's NetFlow, which was introduced in 1996. Because Cisco made the NetFlow data format freely available by distributing the `flowdata.h` file that defined its message formats, it became widely-used, and many NetFlow analysis tools became available. Early versions of NetFlow were implemented in routers, maintaining a table of flow information, and exporting flow entries to a NetFlow collector. This was a simple system, pushing the flow data over UDP transport (unlike SNMP which pulls its data from hosts). However, by the end of the 1990s there were several other flow-based

Manuscript received December 1, 2010.
Manuscript revised March 2, 2011.

[†] The author is with the Faculty of Science, The University of Auckland, New Zealand. He is a co-chair of the IETF's IPFIX Working Group.
a) E-mail: n.brownlee@auckland.ac.nz

DOI: 10.1587/transcom.E94.B.2190

systems, each with its own useful features. Hardware and software implementors, as well as application developers, had a difficult choice as to which flow measurement system they should use.

In August 2000 the IETF held a Birds-Of-a-Feather (BOF) session to consider working on flow-based measurement. Late in 2000 the IP Flow Information eXport (IPFIX) Working Group (WG) was chartered, with goals including:

- Define the notion of a "standard IP flow." The flow definition will be a practical one, similar to those currently in use by existing non-standard flow information export protocols which have attempted to achieve similar goals but have not documented their flow definition.
- Devise data encodings that support analysis of IPv4 and IPv6 unicast and multicast flows traversing a network element at packet header level and other levels of aggregation as requested by the network operator according to the capabilities of the given router implementation.
- Ensure that the flow export system is reliable in that it will minimize the likelihood of flow data being lost due to resource constraints in the exporter or receiver and to accurately report such loss if it occurs.

2. IPFIX: Architecture and Information Model

Rather than attempting to develop a completely new system the IPFIX WG began its activity by surveying existing flow-based systems, partly to help frame the IPFIX Requirements document, and partly so as to help determine which would become the starting point for the IPFIX system. In 2004 that work produced two RFCs, 'IPFIX Requirements' [3] and 'IPFIX Candidate Evaluation' [4].

The WG began by considering commonly-used (in 2000) flow monitoring systems, choosing not to consider systems that did not handle flows as they are defined above. For example, sFlow[†] captures samples of packets as they pass through switches, together with SNMP interface counters. The protocols evaluated by the WG were CRANE, Diameter, LFAP, NetFlow version 9 and Streaming IPDR. RFC 3955 [4] gives brief summaries for each of them.

The WG reached consensus for a generalised IP flow export system, rather than a "carrier-grade accounting protocol that could also be used to export flow information." On that basis, we chose NetFlow version 9 as our starting point, and began work to develop:

- An Information Model that describes its Information Elements (IEs), i.e. attributes to be exported, in a clear and unambiguous manner.
- A simple protocol that exports flow data in compact (binary) form as sets of IE values, with the IEs themselves specified in a list called a 'Template.' IPFIX messages are sent from *IPFIX Exporters* to *IPFIX Collectors.*

IPFIX messages needed a congestion-aware and secure transport, and a means of reporting any IPFIX data losses that might occur, As well, since we could see that IPFIX had many possible uses, it needed to be easily extensible.

Table 1 Example information element, showing its defining properties.

Name:	tcpSynTotalCount
Description:	Total number of packets of this Flow with their TCP SYN flag set
Abstract Data Type:	unsigned64
Data Type Semantics:	totalCounter
ElementId:	218
Units:	packets

The WG had important contributions from members of DMTF (Desktop Management Task Force), resulting in our using XML to describe the IPFIX Information Model. Apart from these early interactions, the IPFIX WG has not had any interactions with other standards bodies.

2.1 IPFIX Information Model

IPFIX's 'Information Model' is essentially just a list of all the IEs currently known to IPFIX. Its RFC [5] lists 238 IEs, each with a name, element ID (an integer), description and type. An example IE entry is shown in Table 1.

In an IE entry like this, the description must specify exactly what quantity it represents, either directly or by referring to some other document. Note that IPFIX does not make any measurements itself, an IPFIX Exporter must get IE values from other systems that have access to them. IE names begin with a lower-case letter, and have upper-case letters for the first letter of each component, e.g. tcpSynTotalCount above. They appear in that form throughout this paper.

In this example an unsigned64 may seem larger than necessary, and therefore wasteful of space in IPFIX messages. However, 'Abstract Data Type' describes the IE's maximum possible value. A Data Template that uses an integer-valued IE must specify its length in octets, allowing implementors to decide on the actual maximum length needed in their system.

The 'standard' IE list is maintained in an IANA Registry [6] in XML, so that implementors can download it and use XML-based tools to help them generate code. If new IEs are needed, they can be added using the procedure documented in [5], i.e. submit a request to IANA who will call for an expert review. Again, individual organisations can develop a set of their own IEs; these are identified in IPFIX messages by having the organisation's Private Enterprise Number (PEN) appended as a suffix. In this way a new set of IEs can be developed and tested — if they prove useful one could ask for them to be added to the IPFIX IE list.

2.2 IPFIX Protocol

The IPFIX protocol RFC [7], published in 2008, explains

[†] http://www.sflow.org/

the IPFIX terminology. In brief:

- packets are observed at an Observation Point,
- Flows are sets of packets that share a common set of properties, passing an Observation Point during a certain time interval,
- Flows are defined using Flow Keys, i.e. a set of IEs and their allowed values, and
- a Template is 'an ordered sequence of <type, length> pairs' that specifies 'the structure and semantics of a particular set of information' that can be exported.

IPFIX messages have a simple basic structure, beginning with a header that specifies the message length, and metadata about this particular message. The header is followed by one or more 'Sets,' which may be

- Template Sets, containing one or more data templates,
- Data Sets, carrying data using already-defined templates, or
- Options Template Sets, carrying information such as Flow Keys and Sampling Parameters for a specified Exporter-related scope.

Other authors such as [8] give a more detailed description of IPFIX messages. For this paper, I have used `ripfix` [9] to create a minimal IPFIX Exporter and Collector. Their Ruby source code, and examples of its 'on the wire' packets are shown in Appendix.

Finding a congestion-aware transport for IPFIX took the WG quite some time. TCP was the obvious choice, but — since it is a *reliable* protocol — it could prove memory-hungry where a router with multiple 10 Gb/s interfaces was exporting data over an unreliable path. The best choice proved to be SCTP, using its 'partial reliability' extension [10], which allows one to say "resend a packet at most n times," so that transmission cannot be indefinitely blocked by packet losses. Unfortunately, SCTP is often not included by default in operating systems, so using it is a non-trivial task; IPFIX can therefore use TCP instead. Again, in environments where a high-rate UDP flow of IPFIX data can be tolerated, e.g. within an ISP Point Of Presence, UDP can also be used. IPFIX uses port 4739 (IPFX on a telephone keypad) for SCTP, TCP and UDP.

Two other issues were important for IPFIX: reliability and security. The WG decided that IPFIX need not be completely reliable, as explained above, but that it should be possible for a Collector to detect when IPFIX messages have been lost. IPFIX does that using a sequence number in its message header. This gives an IPFIX user the choice of full reliability (using either SCTP or TCP) for 'billing' or 'security monitoring' applications, or partial reliability (using SCTP or UDP) for monitoring network and application behaviour. Each of the IPFIX RFCs have Security Considerations sections that explore questions concerning security when using IPFIX. Where data confidentiality or integrity are important, TLS can be used over TCP, and DTLS over SCTP or UDP.

The overall IPFIX Architecture is described in detail in RFC 5470, an overview is shown here in Fig. 1. RFC 5470 did not include any standardised way to configure an IPFIX Exporter, since that was outside the WG's original charter. Configuration is included in current IPFIX work, see Sect. 4.4.

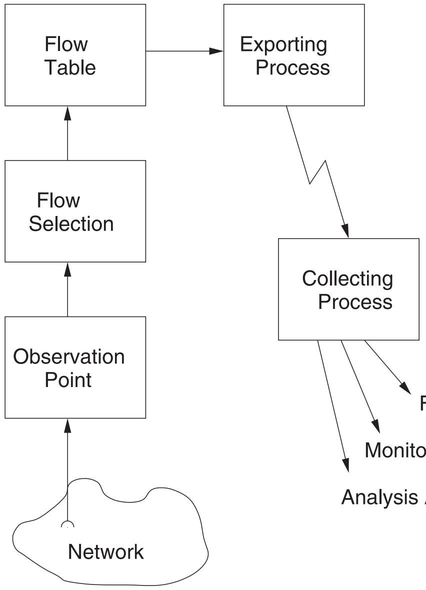

Fig. 1 IPFIX architecture overview. Packets are examined at an observation point, processed into flows by a selection process, and stored in a flow table. From time to time they are exported over the network to a collector for processing and archival.

3. Developing IPFIX

During the development of the IPFIX Protocol and Information Model several different groups were working on implementing it. Those groups met three times in 2005 and 2006 and held IPFIX Interoperability Events. These events were important because they revealed parts of the IPFIX Internet Drafts that were unclear, and overall helped to improve the system design. This early implementation experience was documented in [11] and gave rise to a set of IPFIX Testing Guidelines [12].

Since then other work has continued steadily, improving IPFIX by adding further features, as presented below. Note that all these developments required new IEs, but did not require changes to the IPFIX protocol.

3.1 Bidirectional Flows

IPFIX could originally only export one-directional flows. However, experience with earlier systems (e.g. RTFM) had shown that *biflows* (bidirectional flows) were useful since they simplify the analysis of connection behaviour.

Adding biflows to IPFIX meant having IEs that could be used for counters for each direction. That could have been done by adding 'reverse' versions of IEs like octetDeltaCount and packetDeltaCount, but that would have required adding 'reverse' versions of *all* such IEs. IPFIX

Field Specifiers [7] contain one bit that indicates whether the Field's IE is 'standard' (in the IANA Registry), or belongs to a Private Enterprise. The WG decided to register a PEN for the "IPFIX Reverse Information Element Private Enterprise" (29305) as a general mechanism for creating 'reverse' versions of any IE that might be needed in a biflow. Some have described this approach as "a hack," but is a simple and effective way to introduce 'reverse IEs,' without adding an arbitrary number of new IEs.

One new aspect of biflows is the question "how do we know what 'forward direction' means for this flow?" IPFIX handled this using a single IE, biflowDirection, with values indicating how an exported flow's direction was determined. Biflows are described in detail in [13].

3.2 Reducing Redundancy

Another potential problem for IPFIX was that flows with common properties (e.g. same source IP address and port) had to export values of those properties for every flow, thus wasting export bandwidth. The WG addressed this issue in [14] by introducing another IE, commonPropertiesID.

The idea here is that an IPFIX Exporter will maintain a table of values for a specified set of common properties, and that the commonPropertiesID value is an index into that table. Values from that table will be exported as they are created and deleted. Once the table is active, a Data Template can use commonPropertiesID to indicate which set of common property values belongs to a particular Exported data set. This is described in [14] as allowing "Common and Specific Properties" to be Exported separately.

The scheme depends on having an Exporter configured so that it knows which Common Properties templates are used in any particular Flow Key. However, it did not require any changes to the IPFIX protocol, and it can clearly be useful in minimising the number of bytes to be exported.

3.3 Packet Sampling: PSAMP

From time to time a Network Operator may need to observe a subset of the packets passing an Observation Point, and to export some information from each packet. For example, one might want to observe only DNS response messages from nameservers within a single network prefix. One could do that using `tcpdump`, but tcpdump can only write a trace file; an Operator may wish to export sampled packets from several Observation Points at different locations, and collect them at a central point.

PSAMP is an IETF WG chartered to "define a standard way to sample subsets of packets." PSAMP was developed in parallel with IPFIX, allowing it to influence the IPFIX protocol early on. Since IPFIX simply exports specified sets of IE values, it makes no difference whether those values come from a single packet or a flow. Indeed, PSAMP regards a single packet as a very short flow, with the IPFIX protocol providing message headers that have timestamps and sequence numbers.

PSAMP packet selection is described in [15]. As well as selection by matching packet properties (as in the example above), PSAMP offers hash-based selection. Further, it can select packets using one of five different sampling methods, e.g. count-based, time-based, etc. The PSAMP protocol [16] extends the IPFIX architecture by selecting packets as they arrive at an Observation Point, then exporting them using the IPFIX protocol. Note that this mechanism can be used to select packets passed to a Metering Process for both IPFIX (flows) and PSAMP (packets).

The PSAMP Information Model [17] adds IEs 301-338 to IPFIX. Some of these are used to specify what kind of filtering or sampling is being used, some carry observation times (when a packet was observed, with ms or ns precision), and others carry packet properties (e.g. a specified number of payload bytes).

3.4 IPFIX MIB

The WG developed a MIB module for monitoring IPFIX [18]. The MIB contains tables for each component of an IPFIX device, and gives an interesting overview of how one might implement IPFIX. All the MIB's objects have read-only access clauses, since the MIB was intended only for monitoring.

4. Current IPFIX Work

Once the IPFIX Protocol and Information Model standards were published, in mid-2008 the Working Group rechartered and began work on improving IPFIX's infrastructure, as described in this section.

4.1 IPFIX File Format

Users who collect IPFIX data on a long-term basis need to have an effective way to store and archive it for later analysis, or for sharing with other users, e.g. flow data sharing within the research community. Since the IPFIX Protocol is compact and efficient, with the capacity to Export many kinds of data 'on the wire,' it seemed natural to also use it for storage. The IPFIX File Format RFC [19] sets out the requirements for IPFIX file storage, describing IPFIX files as "another form of transport between IPFIX Exporters and Collectors." A File Writer can be co-located with an Exporter, and a File Reader can be used as input to a Collector.

An IPFIX file is regarded as a complete IPFIX transport session, so Templates must appear in the file before Data Sets that use them. The files are intended to be freestanding, so [19] introduces many new IEs that should be used to specify when and where the File was written. IPFIX files may use CMS detached signatures [20] for integrity checking and ensuring the identity of IPFIX File writers.

4.2 Exporting Type Information

To handle any IE properly, IPFIX Exporters and Collectors need to know the IE's specification. For standard IEs they can do that by simply reading the IANA IE Registry [6]. For Enterprise IEs, however, that information may not be readily available. To solve this problem, the WG added eight new IEs [21] that can be used to export the defining information for an Enterprise IE. The ability to export type information is specifically meant to address the question "how is this IE encoded?" for archival data, when details for Enterprise IEs may have been lost. This ability was developed in conjunction with the IPFIX file format.

Type definitions for Enterprise IEs may be exported in an Options Template that uses the informationElementDataType/privateEnterpriseNumber IEs as its scope, with other IEs including informationElementId and informationElementDataType that specify its properties. After a Collector has received such Option Templates it can use them to determine how it will handle data values for the IEs they describe.

4.3 IPFIX Mediation

When monitoring small networks, one may use a single IPFIX Collector to receive and process that data from many remote IPFIX Exporters. Unfortunately, such an approach scales poorly. Large networks, say those with more than about 1000 exporting nodes, are too big to have just a single collector. Instead one needs a set of Collectors, each reducing the incoming data in some way, e.g. for particular geographic regions. The WG has generalised this idea as 'IPFIX Mediation.'

The IPFIX Mediation Problem Statement RFC [22] defines *Mediation* as "the manipulation and conversion of a record stream for subsequent export using the IPFIX protocol." An *IPFIX Mediator* is a device that receives IPFIX records, processes them using one or *Intermediate Processes,* then Exports them using IPFIX. Figure 2 shows a simplified overview of the IPFIX Mediation architecture.

Intermediate Processes considered so far are Anonymisation, Flow Selection and Aggregation[†]

- The *Anonymisation* document [23] categorises ways to anonymise data exported by IPFIX, for example by encrypting IP addresses, and discusses the trade-off between protecting data privacy and loosing information detail
- *Flow Selection* means 'selecting a subset of flows from those appearing at an IPFIX Mediator (or Exporter).' This is described in [24], together with a set of IEs for describing how flows are selected
- The *Aggregation* draft [25] describes ways to combine data either from a set of flows with common properties *(spatial)* or over a period of time *(temporal)*

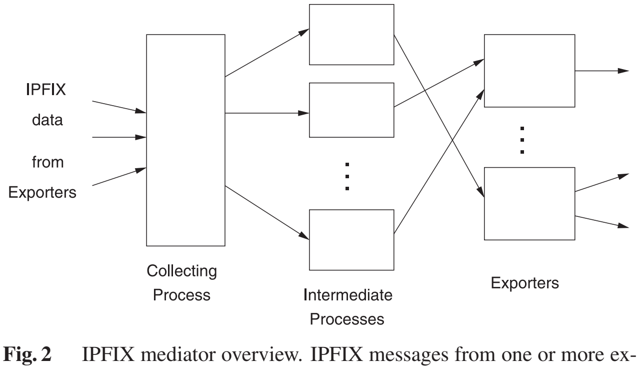

Fig. 2 IPFIX mediator overview. IPFIX messages from one or more exporters arrive at a collecting process. They pass through one or more intermediate processes and are then exported via one or more exporting processes.

Other Intermediate Processes could be added when the need for them becomes apparent; this remains as possible future work for the IPFIX WG.

4.4 IPFIX Configuration Model

The WG's original charter explicitly disallowed work on configuring IPFIX devices — instead configuration was left to individual implementors. That proved to be a sensible approach, allowing the WG to focus on developing, testing and refining the IPFIX Architecture and Information Model. Now that these were well-defined, the WG began work on a Configuration Data Model.

The IPFIX Configuration Data Model [26] will provide a standard way to "configure and monitor Selection Processes, Caches, Exporting Processes, and Collecting Processes of IPFIX and PSAMP compliant Monitoring Devices using the NETCONF protocol [27]. The data model is defined using UML (Unified Modeling Language) class diagrams and formally specified using YANG [28]." Further, the Configuration Data Model (currently an Internet Draft) provides a clear description of how the various parts behave, and how they interact within IPFIX devices.

4.5 Exporting Structured Data

At present IPFIX can export any one of its Information Elements. It cannot, however, export any kind of structure made up of IEs. Such an ability would be useful, for example one could export a variable length set of IP addresses for hosts that had sent TCP SYN packets to a server. The WG is producing a method of exporting structured data by introducing IPFIX encodings for Lists, a 'semantic' that describes the relationship among the IEs in a List, and three new IEs: basicList, subTemplateList and subTemplateMultiList. Examples of how to construct structured data objects in Templates are given in [29].

These will allow IPFIX to export arbitrarily complex data structures, though its main use, at least initially, will be for simple basicLists. This has been possible without changing the IPFIX Protocol or Information Model — it was achieved simply by finding new ways to use them for this purpose.

[†] Anonymisation and Flow Selection are current IPFIX work, Aggregation is a proposal for future work.

5. Future IPFIX Work

At the time of writing (November 2010) the WG is close to completing its charter, but it is still very active. Possible work items for a new charter include those discussed in this section.

5.1 IE Doctors: Managing the Information Model

Since the IPFIX Information Model was published in 2008 [5], it has been extended in three ways — in RFCs that define new sets of IEs for a particular application area, such as PSAMP, by RFCs that define a few new IEs such as File Format, and by requests to IANA for new IEs a few at a time, subject to Expert Review. Recently, other IETF Working Groups such as SIPCLF have become interested in using IPFIX to export measurement data for new application areas. Such efforts have generated requests for help on how to use IPFIX, how to design new sets of IEs, and on how to register them with IANA. This need is addressed in an Internet Draft [30], "IE Doctors," a set of guidelines about IPFIX Information Elements for Draft authors and IE reviewers.

The first attempt to use these guidelines is an Internet Draft on "IEs for Flow Performance Measurement" [31]. It proposes sets of performance related IEs at the Transport, User and Application layers of the networking stack. This draft was discussed at IETF 80; WG consensus was that "the IPFIX WG should not define any metrics or ways to measure them. That should be done in other WGs," e.g. IP Performance Measurement (IPPM).

In future, when network devices have IPFIX Exporters built-in, it would be useful for developers to create new sets of IEs for their applications areas. For example, one obvious area would be IEs for usage-based billing. These might be developed either within the IPFIX WG, in other WGs, or by the WG in co-operation with other standards organisations.

5.1.1 Rethinking the Definition of Flows

The IPFIX WG was originally chartered to "standardise existing practice for exporting IP flow data," hence that was the main goal of its Protocol and Information Model RFCs. However, the Information Model contains many IEs that are not IP-related. For example:

- ingressInterface, egressInterface
- mplsLabelStackSection, mplsLabelStackSection2, etc
- wlanChannelID, wlanSSID

Recently there has been considerable interest in exporting more information about level-2 network behaviour, motivated by [32]. That Draft has generated considerable discussion, suggesting that IPFIX's definition of Flow should be widened further. Since flow information is exported at regular intervals, its most important feature is that it produces time sequences of the flow data. Therefore, any process that generates data at intervals may be considered to be a flow. Having a new Flow definition could make it simpler for other applications to use IPFIX to export measurement data.

Another application that generates sequential data is logging of event data. This is current work in the IETF's SIP Common Log Format (SIPCLF) Working Group. Two formats proposed were 'Indexed ASCII' and IPFIX. 'Indexed ASCII' aims to make searching the logs easier, alternatively the application receiving SIPCLF IPFIX data could simply store the collected IPFIX records in a database. A possible set of IPFIX IEs for SIPCLF was presented in [33]. Although IPFIX was ultimately not adopted for SIPCLF, work is in progress elsewhere on extending IPFIX for SIP layer 7 export.

It might also be possible to create a set of IPFIX IEs for system logging, as an alternative to `syslog`. Such a logging system would need to coexist with `syslog` for many years.

5.1.2 Exporting MIB Variables

Another interesting use for IPFIX could be to export MIB variables. SNMP is a 'pull' protocol, so applications such as MRTG must read the values of MIB variables at regular intervals, say every five minutes. When Flows are being measured and exported, it would be useful to also export values of relevant MIB variables. As well, this could be an effective way for an application to 'push' data to a remote collector. A draft proposing this has already been published [34].

6. Deployment and Use

More than two years have elapsed since the first Standards-Track IPFIX RFCs were published. This section briefly surveys current usage of IPFIX, and considers how it is likely to be deployed.

Firstly, researchers find IPFIX attractive because it provides a simple and effective way to export almost any kind of data. Several research groups have produced open-source implementations, for example:

- *libipfix* from Fokus, a C library for constructing Exporters and Collectors[†]
- *Maji* from WAND, an IPFIX meter that supports 50 standard IEs, and allows user-defined templates[††]
- *Vermont* from Berlios, an IPFIX meter and collector [35], supporting IPFIX and PSAMP[†††]

IPFIX has been used to collect data in FP7 PRISM, "a European Union-funded research project that addresses privacy and scalability in network monitoring" [8]. That project used three separate IPFIX implementations, and used IPFIX files in its data processing infrastructure. Other research work using IPFIX is reported in papers such as [36].

[†] http://libipfix.sourceforge.net

[††] http://research.wand.net.nz/software/maji.php

[†††] http://vermont.berlios.de/start

Secondly, developers of network management systems have extended them to accept IPFIX input. These systems include *Scrutinizer* (Open Source)[†] and commercial offerings such as *Orion NetFlow Analyzer*[††] and *ntop + nProbe*[†††]. This move allows users of such systems, i.e. Network Operators, to continue using them, accepting IPFIX input from stand-alone probes and — as vendors implement IPFIX Export — from routers and switches. As well, implementors will be able to develop new applications that monitor new network devices and systems, using IPFIX as an easily-extensible standard infrastructure.

Lastly, IPFIX usage should increase as equipment vendors implement Exporters in more network devices. Of course, Network Operators will not deploy a new technology unless it provides some real benefit to them. For IPFIX, that benefit should come from being able to extend monitoring to new devices and systems. One area of particular interest may be network security monitoring, using PSAMP to observe 'unusual' features of packets and report their properties using IPFIX [35], [37].

7. Conclusion

IPFIX development started with the formation of the IETF Working Group in 2000. Background work was published in 2004 and the first IPFIX RFCs in 2008. Since then the working group has been re-chartered and has produced a continuing stream of IPFIX extensions, taking it into new areas a long way from simply 'standardising current practice.'

During its ten-year history the WG has had strong involvement from researchers (e.g. Fokus, ETH and WAND), from industry (e.g. Cisco), and from network operators (e.g. NTT). As a consequence, there were many implementations — in IETF terms 'running code' — early on. One IPFIX feature that has proved useful to implementors is its XML Information Model; that allows automated code generation for IE handling, simplifying implementation. Implementations were tested during repeated interoperability events, refining and improving the protocol, and producing RFCs on Implementation and Testing — a somewhat unusual output for an IETF WG.

Many implementations have been reported. Current Open Source implementations are discussed in Sect. 6, and at least one network equipment vendor (Nortel) has implemented IPFIX in their hardware. We trust that more vendors will implement IPFIX soon, that the developers of Flow Analysis tools will convert them to work with IPFIX, and that new tools will be developed using the IPFIX data and file formats.

This steady development over ten years shows that IPFIX fills a real need. I believe that demonstrates that the network community understands the need for measurement — particularly flow measurement — at all stages of the system life-cycle. Overall, IPFIX has proved to be an effective and easily extensible system; that can surely continue into the foreseeable future!

[†] http://www.plixer.com
[††] http://www.solarwinds.com
[†††] http://www.ntop.org

Acknowledgments

My activity in the IPFIX WG is generously supported by The University of Auckland (New Zealand), and by CAIDA (University of California, San Diego). Thanks to Brian Trammell for his helpful comments, and for his development and support of `ripfix`. Thanks also to the anonymous reviewers and to Brian Trammel; their feedback certainly improved this paper.

References

[1] K.C. Claffy, H.W. Braun, and G.C. Polyzos, "A parameterizable methodology for Internet traffic flow profiling," IEEE J. Sel. Areas Commun., vol.13, no.1, pp.1481–1494, 1995.
[2] N. Brownlee, C. Mills, and G. Ruth, "Traffic flow measurement: Architecture," RFC 2722 (Informational), Oct. 1999.
[3] J. Quittek, T. Zseby, B. Claise, and S. Zander, "Requirements for IP flow information export (IPFIX)," RFC 3917 (Informational), Oct. 2004.
[4] S. Leinen, "Evaluation of candidate protocols for IP flow information export (IPFIX)," RFC 3955 (Informational), Oct. 2004.
[5] J. Quittek, S. Bryant, B. Claise, P. Aitken, and J. Meyer, "Information model for IP flow information export," RFC 5102 (Proposed Standard), Jan. 2008.
[6] IANA, "IPFIX Information Element Registry," http://www.iana.org/assignments/ipfix/ipfix.xhtml
[7] B. Claise, "Specification of the IP flow information export (IPFIX) protocol for the exchange of IP traffic flow information," RFC 5101 (Proposed Standard), Jan. 2008.
[8] B. Trammell and E. Boschi, "An introduction to IP flow information export," IEEE Commun. Mag., vol.49, no.4, pp.89–95, April 2011.
[9] B. Trammell, "Ripfix: Rapid prototyping and debugging of IPFIX applications in ruby," Proc. Network Management Research Group, 2010. IETF 78, Maastricht.
[10] R. Stewart, M. Ramalho, Q. Xie, M. Tuexen, and P. Conrad, "Stream control transmission protocol (SCTP) partial reliability extension," RFC 3758 (Proposed Standard), May 2004.
[11] E. Boschi, L. Mark, J. Quittek, M. Stiemerling, and P. Aitken, "IP flow information export (IPFIX) implementation guidelines," RFC 5153 (Informational), April 2008.
[12] C. Schmoll, P. Aitken, and B. Claise, "Guidelines for IP flow information export (IPFIX) testing," RFC 5471 (Informational), March 2009.
[13] B. Trammell and E. Boschi, "Bidirectional flow export using IP flow information export (IPFIX)," RFC 5103 (Proposed Standard), Jan. 2008.
[14] E. Boschi, L. Mark, and B. Claise, "Reducing redundancy in IP flow information export (IPFIX) and packet sampling (PSAMP) reports," RFC 5473 (Informational), March 2009.
[15] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall, "Sampling and filtering techniques for IP packet selection," RFC 5475 (Proposed Standard), March 2009.
[16] B. Claise, A. Johnson, and J. Quittek, "Packet sampling (PSAMP) protocol specifications," RFC 5476 (Proposed Standard), March 2009.
[17] T. Dietz, B. Claise, P. Aitken, F. Dressler, and G. Carle, "Information model for packet sampling exports," RFC 5477 (Proposed Standard), March 2009.
[18] T. Dietz, A. Kobayashi, B. Claise, and G. Muenz, "Definitions of

managed objects for IP flow information export,” RFC 5815 (Proposed Standard), April 2010.

[19] B. Trammell, E. Boschi, L. Mark, T. Zseby, and A. Wagner, “Specification of the IP flow information export (IPFIX) file format,” RFC 5655 (Proposed Standard), Oct. 2009.

[20] R. Housley, “Cryptographic message syntax (CMS),” RFC 3852 (Proposed Standard), July 2004. Obsoleted by RFC 5652, updated by RFCs 4853, 5083.

[21] E. Boschi, B. Trammell, L. Mark, and T. Zseby, “Exporting type information for IP flow information export (IPFIX) information elements,” RFC 5610 (Proposed Standard), July 2009.

[22] A. Kobayashi and B. Claise, “IP flow information export (IPFIX) mediation: Problem statement,” RFC 5982 (Informational), Aug. 2010.

[23] E. Boschi and B. Trammell, “IP flow anonymization support,” draft-ietf-ipfix-anon: Internet Draft — Work in Progress.

[24] L. Peluso, T. Zseby, S. D’Antonio, and C. Henke, “Flow selection techniques,” draft-ietf-ipfix-flow-selection-tech: Internet Draft — Work in Progress.

[25] B. Trammell, E. Boschi, A. Wagner, and B. Claise, “Exporting aggregated flow data using the IP flow information export (IPFIX) protocol,” draft-trammell-ipfix-a9n: Internet Draft — Work in Progress.

[26] G. Muenz, B. Claise, and P. Aitken, “Configuration data model for IPFIX and PSAMP,” draft-ietf-ipfix-configuration-model: Internet Draft — Work in Progress.

[27] R. Enns, “NETCONF configuration protocol,” RFC 4741 (Proposed Standard), Dec. 2006.

[28] M. Bjorklund, “YANG — A data modeling language for the network configuration protocol (NETCONF),” RFC 6020 (Proposed Standard), Oct. 2010.

[29] B. Claise, G. Dhandapani, P. Aitken, and S. Yates, “Export of structured data in IPFIX,” draft-ietf-ipfix-structured-data: Internet Draft — Work in Progress.

[30] B. Trammell and B. Claise, “Guidelines for authors and reviewers of IPFIX information elements,” draft-trammell-ipfix-ie-doctors: Internet Draft — Work in Progress.

[31] A. Akhter, “Information elements for flow performance measurement,” draft-akhter-ipfix-perfmon: Internet Draft — Work in Progress.

[32] S. Kashima and A. Kobayashi, “Information elements for data link layer traffic measurement,” draft-kashima-ipfix-data-link-layer-monitoring: Internet Draft — Work in Progress.

[33] S. Niccolini, B. Claise, B. Trammell, and H. Kaplan, “A common log format for sip using IPFIX files,” Internet Draft — Work in Progress.

[34] A. Johnson, B. Claise, and P. Aitken, “Exporting MIB variables using the IPFIX protocol,” draft-johnson-ipfix-mib-variable-export: Internet Draft — Work in Progress.

[35] R. Christoph, C. Sommer, G. Mnz, and F. Dressler, “Vermont — A Versatile Monitoring Toolkit for IPFIX and PSAMP,” Proc. Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2006), Sept. 2006. Tuebingen, Germany.

[36] F. Fatemipour and M. Yaghmaee, “Design and implementation of a monitoring system based on IPFIX protocol,” Telecommunications, 2007. AICT 2007. The Third Advanced International Conference on, May 2007.

[37] T. Zseby, E. Boschi, T. Hirsch, and L. Mark, “Ipfix/psamp: What future standards can offer to network security,” Proc. FloCon 2006, 2006.

Appendix: Ripfix Example

To demonstrate the simplicity of IPFIX, this appendix gives an example of a simple system using IPFIX to transfer data from a probe monitoring DNS behaviour, together with a commented version of the first few IPFIX messages sent by the Exporter.

Table A· 1 Example set of Information Elements for exporting DNS performance data.

Information Element	Number	Type
dnsServerNumber	1	unsigned32
dnsResponseType	2	unsigned8
dnsResponseLength	3	unsigned16
dnsNbrQuestions	4	unsigned8
dnsTLD	5	string
dnsRttSeconds	6	float32

For this example we defined a set of IEs using our Private enterprise Number, 411. Our probe maintains a table of nameserver addresses, we refer to them by their index in that table, dnsServerNumber. For each observed request/response DNS transaction the probe produces a set of measures, as shown in Table A· 1.
We used `ripfix` to create a Ruby IPFIX Exporter and Collector for this data. An outline of the Ruby code is shown below:

A.1 `Ripfix` Code Outline

```
require ’ipfix/model’  # Use IPFIX module
require ’ipfix/message’
require ’ipfix/exporter’
require ’ipfix/collector’

#Set up Information Model for our DNS IEs
model = IPFIX::InfoModel.new
model.add_spec(’dnsServerNumber(411/1)<unsigned32>[4]’)
model.add_spec(’dnsResponseType(411/2)<unsigned8>[1]’)
model.add_spec(’dnsResponseLength(411/3)<unsigned16>[2]’)
model.add_spec(’dnsNbrQuestions(411/4)<unsigned8>[1]’)
model.add_spec(’dnsTLD(411/5)<string>[65535]’) # variable length
model.add_spec(’dnsRttSeconds(411/6)<float32>[4]’)

# Create symbols we can use to refer to our IEs
model.ie_for_spec(’dnsServerNumber’).hashkey = :server_nbr
model.ie_for_spec(’dnsResponseType’).hashkey = :resp_type
model.ie_for_spec(’dnsResponseLength’).hashkey = :resp_length
model.ie_for_spec(’dnsNbrQuestions’).hashkey = :n_questions
model.ie_for_spec(’dnsTLD’).hashkey = :tld
model.ie_for_spec(’dnsRttSeconds’).hashkey = :rtt

templates = Array.new  # Single template (531) in templates
templates << (Template.new(model, 531) <<
  ’dnsServerNumber’ <<
  ’dnsResponseType’ <<
  ’dnsResponseLength’ <<
  ’dnsNbrQuestions’ <<
  ’dnsTLD’ <<
  ’dnsRttSeconds’)

 # Code for an IPFIX Exporter
 ep = TCPExporter.new(’collector.auckland.ac.nz’,
        4739,  # Collector listens on port 4739
        model, # Our data model
        7654)  # Observation domain
 ep.message.mtu = 576

 templates(model).each { |t| ep << t }  # Export templates

 loop do  # Export data for DNS transactions
          #   (constants shown in statement below)
   h = { :_ipfix_tid => 531,
     :server_nbr => s, :resp_type => d.rc,
     :resp_length => msg.size, :n_questions => d.n_questions,
     :tld => tld_q1, :rtt => r }
   ep << h  # Export h
 end

 ep.close  # Fimished exporting
```

```
  # Code for an IPFIX Collector
  cps = TCPCollectorServer.new(nil, 4739, model)
  cp = cps.next_collector

  cp.each do |h|
    # hash h is received here, exactly as sent above
  end

  cp.close;  cps.close
```

A.2 Example IPFIX Session

To observe the IPFIX messages generated by `ripfix` we used `tcpdump` to record them as they were sent by the Exporter. The first few messages (after the TCP connection was established) are shown below.

```
TCP Packet 4: length=637, data_len=567
  00 0a 02 37 4d 61 b8 80 00 00 00 00 00 00 1d e6
  Message Header:
    Version 10, Length 567,
    Export Time 2011-02-21 13:57:36
    Sequence Number 0, Observation Domain ID 7654

  00 02 00 38
  Template Set, Length 56
    Template 531, Field Count 6
  80 01 00 04 00 00 01 9b
     IE Number 1, PEN 411, Field Length 8
  80 02 00 01 00 00 01 9b
     IE Number 2, PEN 411, Field Length 8
  80 03 00 02 00 00 01 9b
     IE Number 3, PEN 411, Field Length 8
  80 04 00 01 00 00 01 9b
     IE Number 4, PEN 411, Field Length 8
  80 05 ff ff 00 00 01 9b
     IE Number 5, PEN 411, Field Length 8
  80 06 00 04 00 00 01 9b
     IE Number 6, PEN 411, Field Length 8

  02 13 01 ef
  Data Set 531, Length 495
  00 00 00 0a 03 00 67 01 02 6e 7a
    seq 0:  10  3  103  1  ’nz’  0.0019
  00 00 00 14 00 00 8a 01 03 6e 65 74
    seq 1:  20  0  138  1  ’net’  0.1375

  ...

  00 00 00 32 00 00 8a 01 04 61 72 70 61
    seq 30:  50  0  138  1  ’arpa’  0.3325

TCP Packet 5: ACK

TCP Packet 6: length=633, data_len=563
  00 0a 02 33 4d 61 b8 80 00 00 00 1f 00 00 1d e6
  Message Header:
    Version 10, Length 563,
    Export Time 2011-02-21 13:57:36
    Sequence Number 31, Observation Domain ID 7654

  02 13 02 23
  Data Set 531, Length 547
  00 00 00 22 03 00 61 01 03 6f 72 67
    seq 31:  34  3  97  1  ’org’  0.2886

  ...

  00 00 00 46 00 00 8a 01 04 61 72 70 61
    seq 64:  70  0  138  1  ’arpa’  0.2782

TCP Packet 7: ACK

...
```

`ripfix` provides a simple and effective way to develop IPFIX systems. Since Ruby is an interpreted language, `ripfix` Exporters and Collectors run more slowly than those written in C. However, it is Open Source code, making it very useful for developing prototype systems. See [9] for more information about `ripfix`.

Nevil Brownlee has been an Associate Professor in The University of Auckland's Computer Science Department since 2004; his teaching and research focuses on the Internet, especially on Internet data collection and Measurement. Nevil managed the University's campus network from its beginnings in 1985, its connection to the Internet in 1989, and its further development to about 1998, thus gaining experience in operating a medium-sized (14,000 hosts) network. He has been active in the IETF since 1992, chairing its RTFM and IPFIX (IP Flow Information Export) Working Groups. Since 2000 Nevil has been associated with CAIDA (the Cooperative Association for Internet Data Analysis); his work there includes measurement and analysis of Internet traffic flows, and of the behaviour of the global Domain Name System (DNS). He is also associated with the WAND Network Research group at the University of Waikato.