Flow monitoring with IPFIX in MAPI

Results, compatibility, and further work

Technical report, v1.1, August 2006

Håvard Mork, Robin Eidissen
Uninett AS

Table of Contents

Abstract

This document is an overview of the IPFIX implementation in MAPI, which is developed by Uninett with the purpose of security and measuring quality of service in the norwegian research network.

The current system of flow monitoring in use at Uninett utilizes flow information from routers in order to track activity in the network. In order to collect more data regarding the protocols, applications using the network, and the quality of data transfer, Uninett is using probes in the network based on passive measurement probes at strategic locations in the backbone network. These probes are based on the MAPI (Monitoring API) toolkit, which is a device-independent programming interface to simplify the task of developing network monitoring software.

This report will further outline results from extensions to this library to provide quality-of-service (QoS) flow monitoring output from MAPI. The results demonstrate possibilities with the system, as well as what is missing for a complete IPFIX implementation.

Chapter 1 Introduction

Reasons for using passive monitoring of a network include accounting, security, and quality of service. This report focuses on the QoS (quality of service) aspects of flow monitoring, in the context of the MAPI toolkit and IPFIXFLIB flow reporting library. This chapter further introduces the architecture and technologies used.

1.1 Terminology and Architecture

Flow monitoring is based on the following entities and nomenclature:

– The *network* that is being monitored. The use of MAPI and IPFIXFLIB is based on passive measurements from a probe that is inserted between routers in the network's backbone. Typically, an optical mirror will capture the Ethernet or HDLC packets from the network, with i.e. the Endace 4.3S OC48 capture card.

– The *measurement process* (Flow Emitter, Emitter, or Probe) reads packets from the network, their source and destination addresses, size, and other attributes based on the contents of the transferred data. The emitter typically aggregates the information from the network into *flow records*. The emitter process relays the flow records to a different process, the collector, which is responsible for storing the information.

– The *collecting process* (Flow Collector, or Collector) is a process that typically receives flow records from one or more measurement processes, and stores the flow records for later use by applications.

– A *flow* is defined as a "set of IP packets passing an Observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties" [ipfix-protocol-22]. Typical properties that identify a flow is a set of source IP, destination IP, source port, and destination port.

The main data flow will therefore be (conceptually):
Network → Flow Emitter → Flow Collector → Applications

The Flow Emitters will have to be configured to collect the data that is required for the applications. The information flow, according to IPFIX, is downstream, which means that configuration of Flow Emitters are manual. However, there exist the possibility that IPFIX-conformant applications may introduce other configuration means.

1.2 Implementation overview and the tools

The following tools are central to this IPFIX implementation:

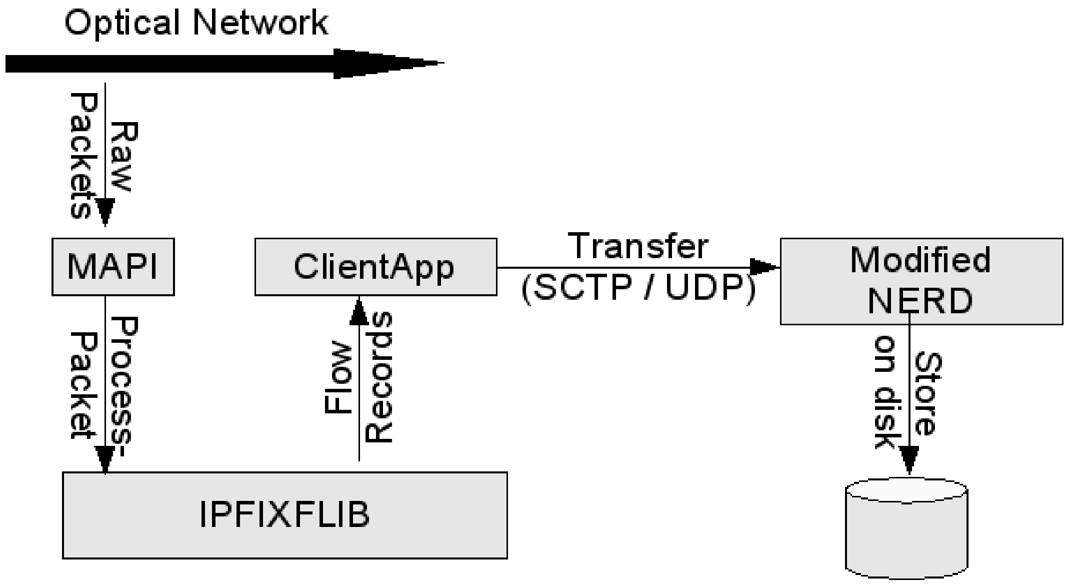
MAPI: Emitter
MAPI, and the IPFIX reporting library IPFIXLIB, generates flow records based on raw Ethernet or HDLC packets on the network. The MAPI library reads packets off a packet capture device, and relays the information to IPFIXFLIB.

MAPI Client Program
A MAPI Client Program is used to connect to MAPI, and relay flow records to a receiver.

Modified version of NERD: Collector
The collecting process is based on a flow collector that has been extended to read IPFIX flow records. The flow records are stored on disk in a format that is similar to how they are emitted. The collector is also responsible for error handling, reporting of errors, and handling of output files (log rotation).

FlowStat: Reporting Tool
FlowStat is a tool that is developed to read NERD output files, to aggregate or display information from flows. It supports calculating sums, averages etc. over IP addresses, handling netmasks, AS numbers, etc.

A sample information pipeline of tools is the following:
DAG interface card → MAPI Interface → MAPI Client Program → NERD → FlowStat



1.3 Report Outline

Chapter 2 presents the current state of the IPFIX draft documents. Chapter 3 shows implementation details of MAPI, modified NERD version, and the FlowStat tool. Chapter 4 presents some example aggregate reports generated with extended IPFIX information fields, with a focus on quality of service, while Chapter 5 concludes the state of the implementation.

Chapter 2 IPFIX

This chapter addresses the IPFIX implementation, and its adherence to the latest IPFIX proposals.

The following versions of the IPFIX documents are used, from the IP Flow Information Export WG[1]:

- Architecture for IP Flow Information Export, v11
- Information Model for IP Flow Information Export, v12
- IPFIX Protocol Specification, v22
- IPFIX Applicability, v9

2.1 Comparison to Netflow v5 and v9

IPFIX is a proposed new version of the Cisco Netflow standard for flow information export. One common version of this is Netflow v5, which uses a static structure of information fields when exporting. IPFIX is an attempt to make this dynamic, and allow its users to specify which information to export. This makes the flow information useful for other applications, such as quality of service.

Netflow v9 is similar to IPFIX in that it allows to dynamically configure which information fields to export. The following significant differences exist between Netflow v9 and IPFix:

Header
IPFix packets have different header:
- "count" (number of packets) replaced by "length" (num bytes) in ipfix
- "sysuptime" (secs since router booted) is removed from the header

Field definitions
IPFix encourages adding new information elements. Fields that are not official must be above 32768 (MSB in 16-bit int is set). In templates sent in IPFix, all fields with MSB set, is followed by a 32-bit "enterprise ID".

IPFix, in constrast to Netflow v9, also supports information elements with dynamic sizes. Dynamic field size is determined by using length 0xFFFF in the template. This is rarely used.

Deprecated fields
Some information elements from Netflow v9 are not used in IPFix. Others may be replaced with other fields with similar semantics. Fields like i.e. number of packets (2) and number of bytes (2) have changed maximum size from 4 bytes to 8 bytes. This should not create problems for v9/ipfix collectors, as they must use the lengths specified in template anyway.

IPFix information element Ids will in the future be managed by IANA. For "custom" information elements, the "enterprise ID" must be set.

New information elements
IPFix defines a large set of information fields. The LIBIPFIX implementation only supports emission of a subset of those.

1 http://www.ietf.org/internet-drafts/draft-ietf-ipfix-architecture-11.txt

2.2 Compatibility of MAPI, NERD

This section lists important points, primarily mandatory implementation issues with ipfix, along with information on the support level in MAPI/IPFIXFLIB and NERD.

Mandatory elements are presented in boldface. Elements that are marked as '*should*' are marked as so. Some elements of the IPFIX documents are not included, as they are not relevant for the QoS domain, and also are not mandatory for implementations.

According to RFC2119, the word *should* should be interpreted as *recommended*. If an element, that is marked as *should,* is replaced with an alternate approach, then the implications of this should be understood and carefully weighted.

IPFIX Architecture

	Issue	*MAPI*	*NERD*
A1	Implementors must provide an effective way to configure their IPFIX devices.	YES	
A2	Maintain database(s) of all the Flow Records from an Observation Domain.	YES	
A3	Maintain statistics about the Metering Process itself	YES	
A4	Flow expiration: - If no packets belonging to the Flow have been observed for a period - Expiration time *should* be configurable at the Metering Process - Expiration time value 0 means immediate export - Premature expiration if resource constraints - Long-running flows should be exported on a regular basis - Timeout value for long-running flows *should* be configurable - Flow records from long-running flows may still be maintained after export	 YES YES NO[2] NO YES YES NO	
A5	The IPFIX Device *should* count the number of packet losses	YES	
A6	The IPFIX Device *should* report all non-IPFIX errors and the error duration	NO	
A7	Control information about flows must be sent before the flows		YES
A8	Transfer of control information *should* be available on a reliable transport		NO
A9	Collector failure detection and recovery - The IPFIX device *should* set a keepalive timeout or heartbeat for tcp/sctp - Collector failure is detected at exporting process. Connection *should* be attempted re-established. - Backup session may be opened in the case of failure. Control information must be re-sent.	 NO NO NO	
A10	Security requirements - Authentication of IPFIX packets *should* be supported - Encyption (mandatory if unsafe transport) - An IPFIX system *should* authenticate endpoints	 NO NO NO	
A11	Collectors should be able to detect loss of exported flow records		NO

2 This is not useful for this IPFIX implementation (?)

	Issue	MAPI	NERD
A12	Denial of Service: - The IPFIX system *should* try to collect as much information as possible - IPFIX specific attacks can be solved by encryption.	 YES NO	

IPFIX Protocol

Issue	MAPI	NERD
All integers must be big-endian	YES	YES
Sequence number *should* identify number of IPFIX data records		
Collecting process *should* distinguish data sources by IP address and Observation domain ID - Support for Ipv6 for the collector (note: transfer of flow records over ipv6)		YES NO
IPFIX Header structure	YES	YES
Enterprise-specific field specifiers - Handling of enterprise numbers	YES YES	YES NO[3]
Option templates - Scope fields (mandatory, may not be 0) - Scope fields that should be supported: LineCardId, TemplateId, exporterIPv4Address, exporterIPv6Address, and ingressInterface. - Scope fields that are optional: meteringProcessId, exportingProcessId, observationDomainId, (etc.)	**NO** NO NO NO	**NO** NO
Specific reporting requirements (options templates, may be implemented) - The Metering Process Statistics Option Template - The Metering Process Reliability Statistics Option Template - The Exporting Process Reliability Statistics Option Template - The Flow Keys Option Template	 NO NO NO NO	 NO NO NO NO
IPFIX Export Time header field	YES	
Linkage with information model - Signed integers in two's complement. - Float32 and Float64 according to IEEE.754.1985 - Strings as unicode - Date/time milli/microsec values in NTP format, RFC1305 - Information fields (signedXX, unsignedXX, Float64) may be transmitted with fewer octets than ipfix-info specifies, if assumed more bits not needed.	 NO[4] YES[5] YES[6] YES YES	
Variable Length Information Elements	YES[7]	YES

3 NERD doesn't distinguish between enterprise numbers (yet)
4 Signed values not used
5 Conforms to IEEE754 if the compiler does so
6 Strings are not used
7 Easy enough to implement on a case-to-case basis.

Issue	***MAPI***	***NERD***
Template management for SCTP: - A newly created template is assigned an unused template ID - Template sets and option template sets must only be sent once, when reliable - Restart of exporting process must re-assign template Ids	 YES[8] **NO** YES	
Template Withdraw Message - TWM must not be sent before a process delay time has elapsed - This process delay time must be configurable (5 seconds is suggested) - Withdrawn template Ids must not be reused until sufficient time has passed - TWM for options templates	**NO** NO NO NO NO	**NO** NO
SCTP [RFC2960] and PR-SCTP [RFC3758] MUST be implemented by all compliant implementations: - Collecting process must support atleast two associations per connection - Reception of malformed message must reset connection, discard message, and report error - If a duplicate template ID is received, that has not been withdrawn, association must be shut down - When closing an SCTP association, all templates and data records from that association must be deleted. - The collector *should* provide a logging mechanism for out-of-sequence sequence numbers in IPFix headers (exhaustion, packet injection, etc.?) - Template Withdraw Messages for non-existant templates must reset the association, discard the IPFIX message, and log the error. - Collector *should* listen for SCTP connections on port 4739 - When collector shuts down, it *should* wait until exporting process shuts down its end before the SCTP connection is terminated. - Exporting process *should* try to re-establish SCTP connection if it is lost. - Association timeouts *should* be configurable	 NO NO	 YES[9] **NO** **NO** **NO** NO **NO** YES NO
Collector must be able to handle IPFIX messages up to 64KB		YES
Exporting process must request at least two outbound associations per connection. Stream 0 transfers templates and option templates, and must be reliable. Stream 1 transfers ipfix records.	**NO**	**NO**
UDP requirements, if UDP is used: - Collector *should* listen to port 4739 - Templates should be transferred at regular intervals - A lifetime should be associated with all templates - Information maintained for each flow template: <Exporting Process, Observation Domain ID, Template ID, Template Definition, Last Received>	 YES	 YES NO NO

8 Assigned template ID is random, and may not always be unique in a time period if i.e. exporting process is restarted. This is mitigated with the assumption that exporting processes will be configured with unique observation domain ID.

9 PR-SCTP not used because not using reliable channel

2.3 SCTP Status

Using SCTP in application-level requires that certain libraries are installed, and linked to both NERD and the client application that connects to MAPI. Furthermore, using such application-level SCTP requires that the applications are run as 'root', since the normal transport protocols are bypassed. This has security implications, and is in many cases not an option.

The preferred use of SCTP involves the use of a kernel that has SCTP Socket support. In addition to the kernel, some operating system files (such as header files) are needed.

Supported kernels
Recent versions of the Linux kernel does have built-in support for SCTP. Kernels from version 2.6.16 are usable. Check your kernel version with the command "cat /proc/version". Kernel can be downloaded from www.kernel.org.

Installation of SCTP-capable kernel
Installation is the following steps:

- Download kernel 2.6.16 or newer
- Read README, and follow instructions, or if you don't want to: (note: Author is not reponsible!)
  - Unzip, to i.e. /usr/newkernel
  - make xconfig, no modifications necessary
  - fakeroot make-kpkg --initrd --revision=custom.1.0 kernel_image
  - cd ..
  - dpkg -i kernel-image-2.6.17.7_custom.1.0_i386.deb (note: change filename to correct name)
- Reboot (and keep in mind that you can use the ESC key to select kernel in GRUB, in case of kernel panics)

Installation of SCTP tools
For compiling SCTP support, you need the kernel SCTP tools (if you already have /usr/include/netinet/sctp.h, then you don't need it!):

- Download lksctp-tools at http://sourceforge.net/project/showfiles.php?group_id=26529
- Run ./bootstrap
  Note that if you get errors on bootstrap, then remove all and unpack again, edit bootstrap and replace aclocal with aclocal-1.9 and automake with automake-1.9
- Run ./configure --prefix=/usr --enable-shared –enable-static
- Run make
- cd src/func_test
- make v4test (to verify the installation and kernel)
- cd ../../
- sudo make install

Copiling NERD with SCTP
Compiling NERD with SCTP-support is automatic, as long as proper kernel is in place, and lksctp-tools is present. The configure script will automatically find netinet/sctp.h and compile SCTP support if found.

Chapter 3 Information Elements

This chapter presents the information elements in use in MAPI, and their support.

3.1 IPFIX Information Elements

The **ID** column specifies the IPFIX ID of the information element. For Ids below 128, the Ids will be similar to their respective Netflow v9 values.
The **Size** column specifies the size (in bytes) of the maximum size of information elements. Elements may be transferred with a lower number of bits if it can be assumed no precision is lost.
The **MAPI Name** column specifies the name of the information field, as used when asking MAPI to export that field. For information fields with no MAPI Name, exporting of that information is not available (yet).
The **IPFIX Name** specifies the name as presented in the IPFIX documents.
Shortname is an alternate, shorter name that can i.e. be used in the reporting tool, for convenience.
The **Type field** specifies the type of the field, as *int*eger, *ipv4* address, histogram, etc.

For documentation concerning individual information elements, please see the IPFIX Information Model available at http://www.ietf.org/html.charters/ipfix-charter.html

ID	SIZE	MAPI NAME	IPFIX NAME	SHORTNAME	TYPE
1	8	BYTES	octetDeltaCount	octets	int
2	8	PKTS	packetDeltaCount	packets	int
4	1	PROT	protocolIdentifier	prot	int
5	1	TOS	ipClassOfService	tos	int
6	1	TCP_FLAGS	tcpControlBits	flags	int
7	2	L4_SRC_PORT	sourceTransportPort	srcport	int
8	4	IP_SRC_ADDR	sourceIPv4Address	src	ipv4
9	1		sourceIPv4PrefixLength	srcmask	int
10	2	INGRESS	ingressInterface	ingress	int
11	2	L4_DST_PORT	destinationTransportPort	dstport	int
12	4	IP_DST_ADDR	destinationIPv4Address	dst	ipv4
13	1		destinationIPv4PrefixLength	dstmask	int
14	2	EGRESS	egressInterface	egress	int
15	4		ipNextHopIPv4Address	nexthop	ipv4
16	2	SRC_AS	bgpSourceAsNumber	src_as	int
17	2	DST_AS	bgpDestinationAsNumber	dst_as	int
18	4		bgpNexthopIPv4Address	nexthop	ipv4
19	4		postMCastPacketDeltaCount		int
20	4		postMCastOctetDeltaCount		int
21	4	LAST_SWITCHED	flowEndSysUpTime		int
22	4	FIRST_SWITCHED	flowStartSysUpTime		int
23	4		postOctetDeltaCount		int
24	4		postPacketDeltaCount		int
25	2	MIN_PKT_LEN	minimumPacketLength		int
26	2	MAX_PKT_LEN	maximumPacketLength		int
27	16	IPV6_SRC_ADDR	sourceIPv6Address	srcv6	ipv6

28	16	IPV6_DST_ADDR	destinationIPv6Address	dstv6	ipv6
29	1		sourceIPv6PrefixLength		int
30	1		destinationIPv6PrefixLength		int
31	4		flowLabelIPv6		int
32	2		icmpTypeCodeIPv4		int
33	1		igmpType		int
36	2	ACTIVE_TIMEOUT	flowActiveTimeout		int
37	2	INACTIVE_TIMEOUT	flowInactiveTimeout		int
40	8	BYTES_EXP	exportedOctetTotalCount		int
41	8	PKTS_EXP	exportedMessageTotalCount		int
42	8	FLOWS_EXP	exportedFlowRecordTotalCount		int
44	1		sourceIPv4Prefix		int
45	1		destinationIPv4Prefix		int
46	1		mplsTopLabelType		int
47	4		mplsTopLabelIPv4Address		int
52	1	MIN_TTL	minimumTTL	minttl	int
53	1	MAX_TTL	maximumTTL	maxttl	int
54	2		fragmentIdentification		int
55	1		postIpClassOfService		int
56	6		sourceMacAddress		int
57	6		postDestinationMacAddr		int
58	2		vlanID		int
59	2		postVlanId		int
60	1	IP_PROTO_VER	ipVersion	ipver	int
62	16		ipNextHopIPv6Address		int
63	16		bgpNexthopIPv6Address		int
64	4	IPV6_OPT_HDR	ipv6ExtensionHeaders		int
70	3		mplsTopLabelStackEntry		int
71	3		mplsLabelStackEntry2		int
72	3		mplsLabelStackEntry3		int
73	3		mplsLabelStackEntry4		int
74	3		mplsLabelStackEntry5		int
75	3		mplsLabelStackEntry6		int
76	3		mplsLabelStackEntry7		int
77	3		mplsLabelStackEntry8		int
78	3		mplsLabelStackEntry9		int
79	3		mplsLabelStackEntry10		int
80	1		destinationMacAddress		int
81	1		postSourceMacAddress		int
85	1		octetTotalCount		int
86	1		packetTotalCount		int
88	1		fragmentOffset		int
90	0		mplsVpnRouteDistinguisher		binary
128	2		bgpNextAdjacentAsNumber		int
129	1		bgpPrevAdjacentAsNumber		int
130	4	EXPORTER_IPV4	exporterIPv4Address		ipv4

131	12	EXPORTER_IPV6	exporterIPv6Address		ipv6
132	1		droppedOctetDeltaCount		int
133	1		droppedPacketDeltaCount		int
134	1		droppedOctetTotalCount		int
135	1		droppedPacketTotalCount		int
136	1	FLOW_END_REASON	flowEndReason		int
137	8		commonPropertiesId		int
138	4		observationPointId		int
139	1		icmpTypeCodeIPv6		int
140	1		mplsTopLabelIPv6Address		int
141	4		lineCardId		int
142	4		portId		int
143	4		meteringProcessId		int
144	4		exportingProcessId		int
145	2	TEMPLATE_ID	templateId	templ	int
146	1		wlanChannelId		int
147	1		wlanSSID		int
148	4	FLOW_ID	flowId		int
149	4	OBSERV_DOMAIN	observationDomainId	obsdom	int
150	4	FLOW_START_SEC	flowStartSeconds	fssec	int
151	4	FLOW_END_SEC	flowEndSeconds	fesec	int
152	8	FLOW_START_MS	flowStartMilliseconds	fsms	int
153	8	FLOW_END_MS	flowEndMilliseconds	fems	int
154	8	FLOW_START_US	flowStartMicroseconds	fsus	int
155	8	FLOW_END_US	flowEndMicroseconds	feus	int
156	8	FLOW_START_NS	flowStartNanoseconds	fsns	int
157	8	FLOW_END_NS	flowEndNanoseconds	fens	int
158	8		flowStartDeltaMicroseconds		int
159	8		flowEndDeltaMicroseconds		int
160	8	SYSINIT_MS	systemInitTimeMillisecond		int
161	8	FLOW_DUR_MS	flowDurationMilliseconds	dur	int
162	8	FLOW_DUR_US	flowDurationMicroseconds	duru	int
163	8	NUM_FLOWS_OBSERVED	observedFlowTotalCount		int
164	8	NUM_IGNORED_PKT	ignoredPacketTotalCount		int
165	8	NUM_IGNORED_OCTETS	ignoredOctetTotalCount		int
166	8	NOTSENT_FLOWS	notSentFlowTotalCount		int
167	8	NOTSENT_PKTS	notSentPacketTotalCount		int
168	8	NOTSENT_OCTETS	notSentOctetTotalCount		int
169	1		destinationIPv6Prefix		int
170	1		sourceIPv6Prefix		int
171	1		postOctetTotalCount		int
172	1		postPacketTotalCount		int
173	1		flowKeyIndicator		int
174	1		postMCastPacketTotalCount		int
175	1		postMCastOctetTotalCount		int
176	1		icmpTypeIPv4		int

177	1		icmpCodeIPv4		int
178	1		icmpTypeIPv6		int
179	1		icmpCodeIPv6		int
180	1		udpSourcePort		int
181	1		udpDestinationPort		int
182	1		tcpSourcePort		int
183	1		tcpDestinationPort		int
184	1		tcpSequenceNumber		int
185	1		tcpAcknowledgementNumber		int
186	2	TCP_WINDOW	tcpWindowSize		int
187	1		tcpUrgentPointer		int
188	1		tcpHeaderLength		int
189	1		ipHeaderLength		int
190	2	PKTLEN_IPV4	totalLengthIPv4	len	int
191	4	PAYLOADLEN_IPV6	payloadLengthIPv6	lenv6	int
192	1		ipTTL		int
193	1		nextHeaderIPv6		int
194	1		mplsPayloadLength		int
195	1		ipDiffServCodePoint		int
196	1		ipPrecedence		int
197	1		fragmentFlags		int
198	1		octetDeltaSumOfSquares		int
199	1		octetTotalSumOfSquares		int
200	1		mplsTopLabelTTL		int
201	1		mplsLabelStackLength		int
202	1		mplsLabelStackDepth		int
203	1		mplsTopLabelExp		int
204	8		ipPayloadLength		int
205	2		udpMessageLength		int
206	1		isMulticast		int
207	1	HDRLEN_IPV4	ipv4IHL		int
208	8	IPV4_OPT	ipv4Options		int
209	8	TCP_OPT	tcpOptions		int
210	1	PADDING	paddingOctets		int
211	4		collectorIPv4Address		ipv4
212	12		collectorIPv6Address		ipv6
213	4		collectorInterface		int
214	1		collectorProtocolVersion		int
215	1		collectorTransportProtocol		int
216	2		collectorTransportPort		int

3.2 UNINETT Enterprise-Specific Information Elements

The following enterprise-specific information elements are defined for UNINETT. After each logical block of information elements, a short description follows.

ID	SIZE	MAPI NAME	IPFIX NAME	SHORTNAME	TYPE
32769	9	HIST_PKT_LEN	pktLenHistogram	hpktsz	hist8
32770	10	HIST_PKT_DIST	pktDistHistogram	hpktdst	hist8

These two are histograms of packet sizes, and distance respectively. Histograms have pre-defined bucket sizes, and a fraction in each byte's bucket to determine its frequency. The best-rated bucket will have 0xFF. A bucket with half the frequency will have 0x7F, etc.
HIST_PKT_LEN measures the sizes of packets. This may characterize the traffic as either being high-speed (large packets), or realtime/slower (small packet sizes)
HIST_PKT_DIST measures the distance between packets, in milliseconds. This shows how frequent packets are, or how much of it is waiting, or ACK streams, etc.

32776	16	PAYLOAD	pktPayload	payload	raw

Payload export is used to export the first few bytes of a flow.

32790	4	VAR_PKT_DIST	pktDistVar	var_dist	float
32791	4	EXPVAL_PKT_DIST	pktDistExpval	ev_dist	float
32792	4	VAR_PKT_LENGTH	pktLengthVar	var_len	float
32793	4	EXPVAL_PKT_LENGTH	pktLengthExpval	ev_len	float
32794	8	SUM_PKT_DIST	pktDistSum	s_pdist	int
32795	8	SUM_PKT_LENGTH	pktLengthSum	s_plen	int
32796	8	QSUM_PKT_DIST	pktDistSumQuadratic	qs_pdist	int
32797	8	QSUM_PKT_LENGTH	pktLengthSumQuadratic	qs_plen	int

Packet sizes and distances, in statistical numbers.

32798	1	CONN_DIRECTION	direction	dir	int

Direction of the flow. Value 0 for a connection request, 1 for a connection response, or 0xFF for unknown (may be due to that a flow is observed after flow started)

32799	4	PKT_REORDERED	reordered	reord	int

Number of reordered packets. Currently monitors TCP sequence numbers, and give the number of *bytes* that have less sequence number than highest one observed. Potential reasons for this is retransmission, or network packet reordering(?).

32800	2	SERVICE	service	serv	int

Experimental service discovery. Performs packet search inside packets to detect protocols that have unreliable port numbers. Currently detects the P2P file sharing clients DC++ (value 1), and BitTorrent (value 2). Value 0 indicates unknown.

32801	4	RTP_JITTER	rtpJitter	rtpj	int
32802	1	RTP_LOSTFRAC	rtpLostFraction	rtplf	int
32803	4	RTP_LOSTPKTS	rtpLostPackets	rtplp	int
32804	2	RTP_SEQCYCLES	rtpSequenceCycles	rtpsq	int

Parameters read from RTCP packets. These may in 5-10% be false positives(?). These are values communicated from various media clients back to a streaming server on reception quality. The most recent value received from the server is presented here.

33805	1	PIM_PACKETS	-	pim	int

The number of packets of this stream that are detected to be encapsulated in PIM payloads. This happens for multicast packets transferred out of its multicast domain, i.e. RTP. Max value is 255. This is mostly used for diagnostic purposes.

33000	4	MAXRATE_1SEC	maxTransfer1sec	max1s	int
33001	4	MINRATE_1SEC	minTransfer1sec	min1s	int
33002	4	MAXRATE_100MS	maxTransfer100ms	max100ms	int
33003	4	MINRATE_100MS	minTransfer100ms	min100ms	int
33004	4	MAXRATE_10MS	maxTransfer10ms	max10ms	int
33005	4	MINRATE_10MS	minTransfer10ms	min10ms	int
33006	4	MAXRATE_1MS	maxTransfer1ms	max1ms	int
33007	4	MINRATE_1MS	minTransfer1ms	min1ms	int

These information elements calculate the peak maximum and minimum bandwidth usage of flows. Note there is a performance overhead by enabling several of the time intervals. Export of only one time interval is recommended if packet loss is observed.

33008	4	TCPWIN_MAX	maxTcpWindow	maxwin	int
33009	4	TCPWIN_MIN	minTcpWindow	minwin	int
33010	4	TCPWIN_EFF	effTcpWindow	effwin	int

The maximum, minimum, and effective(?) TCP window sizes of observed TCP flows. These flags, in contrast to the tcpWindowSize IPFIX flag, take into account RFC1323 window scaling for high-performance links. The effective TCP window is a product of the flow's bandwidth, multiplied by the latency. This is the *maximum* observed effective TCP window, and is calculated as a *sample* from the flow.

3.3 Performance of IPFIX Information Element Export

The export of some information elements may have larger performance impact than others. This is a tentative study of the performance of information element export.

Element (per-packet)	*Instructions*
MAPI Library cost	*200*
IPFIXFLIB with minimal options	*680*
Additional cost: HIST_PKT_LEN	30
Additional cost: HIST_PKT_DIST	495
Additional cost: VAR/EV of DIST/LEN	100
Additional cost: SERVICE	1760
Additional cost: MAXRATE (one interval)	300

This implies that the basic cost, with MAPI compiled with no optimization, is around 1000 instructions per packet. Circumstantial evidence may point towards that packet loss may occur at 4000 instructions/packet on a 500-1000 MBit/sec link. No link with higher bandwith has been available for testing during the summer season.

Reducing the number of bytes exported per flows is probably one of the best ways of increasing the performance, as moving data is the main cost.

Chapter 4 Implementation Details

This chapter details the implementations of IPFIXFLIB, and the algorithms used.

4.1 Algorithms

This section describes the algorithms used to generate some of the QoS information element values used in MAPI.

Time-consuming algorithms are only performed when they are explicitly requested from the user process. Due to the real-time needs for MAPI in production networks, this strategy was needed.

Effective TCP window size

The effective TCP window size is based on the assumption that a TCP sender (S) will have a particular byte-range in transit to the destination (D) at any given time in the flow. Measuring the TCP window is based on the time-range from a TCP sequence number is detected, to the corresponding acknowledgment is observed.

The assumption for this measurement is that the data stream's capacity is fully utilized by the sender, S. It is assumed that the sender can fully exhaust the transfer window, and have to wait for the positive acknowledgment in order to continue sending. Using this assumption, we “know” that the entire transfer window has passed the probe, on its way to the destination. We have that the effective TCP window size, Weff, equals the product of the bitrate and the latency of the ACK:

$$W_{eff} = BR * S = \frac{ObservedBytes}{Latency} \times Latency = ObservedBytes$$

If the measurement process (the probe) is close to S, and the real transfer window being artificially low, then this number will probably be accurate. If the measurement process is closer to the destination, this number will have less value.

Histogram of packet sizes, and distance

Histograms that are generated in the IPFIXLIB library, are generated in an array of 8-bit integers, each bucket specifying the ratio of hits. The largest bucket always have value 0xFF, while a bucket with half the frequency of the largest one has 0x7F.

Histogram of packet sizes are created by simply matching the packet sizes against the buckets, which are hard-coded with the tentative bucket sizes:

[0, 50, 100, 200, 750, 1300, 1400, 1450, 1500, infinity] (bytes)
(9 buckets)

The extremities of the packet sizes are more interesting than the middle-sized packets, since most of the traffic on a network usually is between 1400-1500 and 0-100 bytes.

Packet distance, the number of microseconds elapsed between packets in one direction, is measured in buckets with threshold values:

[0, 25, 50, 75, 100, 200, 300, 400, 500, 1000, infinitite] (microseconds, us)
(10 buckets).

Bitrate calculation

Bitrate calculation is performed for three types of time intervals, both on a max and minimum basis:
- Min/max for 1 second
- Min/max for 0.1 second
- Min/max for 0.01 second

The calculation is per-direction. In other words, bitrate is calculated only for a unidirectional set of endpoints. Bitrate calculation is performed by dividing the time interval into 10 slots. The count of bytes that pass the probe is added to the first slot. When a number of milliseconds has passed, the contents of slot 1 is moved to slot 2, and so on. The minimal bandwith calculation is not performed until sufficient data is collected, because the minimum is always 0 at the beginning of a flow.

Variance and expectancy value

Variance and expectancy value are measured in real-time with the formula:

$$S^2 = \frac{n \cdot \sum x_i^2 - (\sum x_i)^2}{n \cdot (n-1)} \quad \text{(which follows from} \quad S^2 = \frac{\sum (x_i - \bar{x}^2)}{n-1} \quad)$$

To perform the calculation, the probe only needs to keep track of two values for each flow:
- E(x) – the sum of all the values, which is divided on the number of observations at the end.
- E(x^2) – the quadratic sum of observed values

The formula for S^2 can then be used after the flow has been completed, in order to calculate variance and expectancy value. The S^2 formula is made by Simon Jonassen at UNINETT.

Packet sequence reordering detection

The sequence of TCP packets are read by the probe, and the number of out-of-order packets are registered. When a TCP packet with lower sequence number is discovered, a counter is increased. There is no handling of sequence discontinuities, i.e. when a packet has been lost. The retransmission will compensate for this.

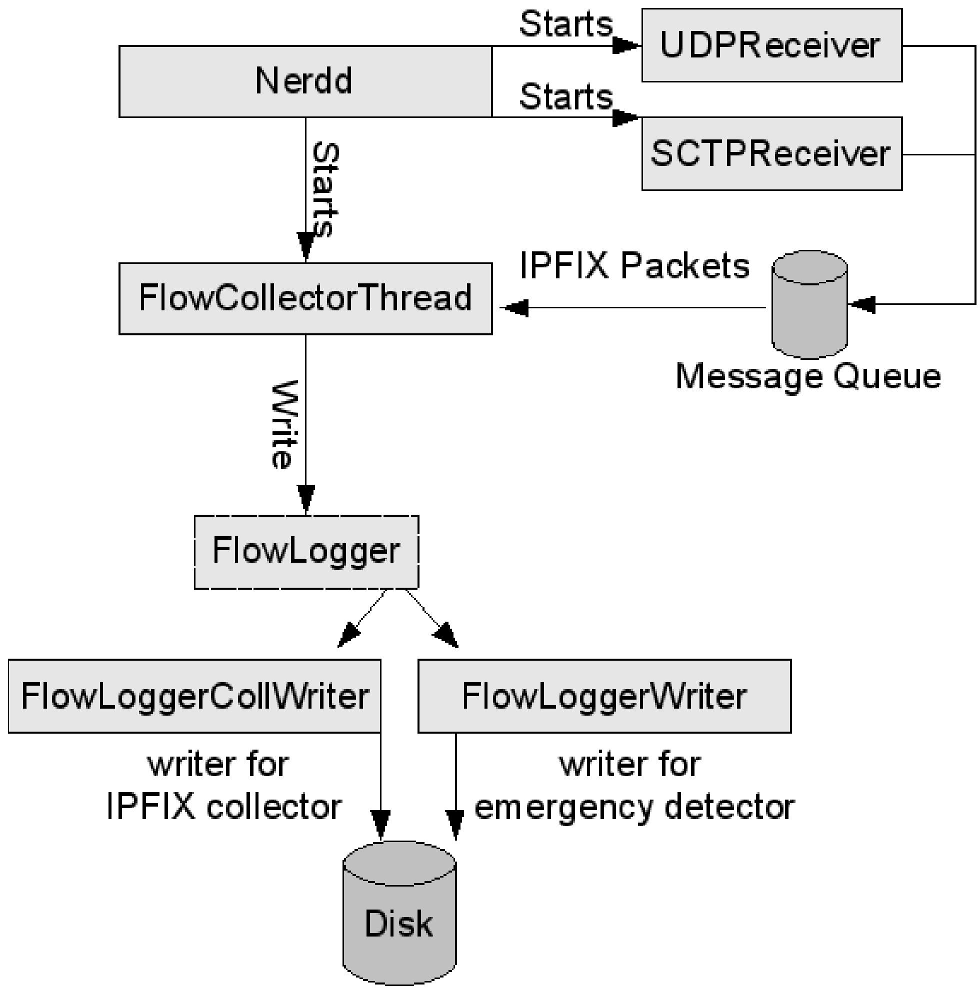
Reordering detection may also be performed for other connection-based IP protocols, but is currently only implemented for TCP.

4.2 Modified version of NERD

The modified version of NERD is available at the UNINETT SVN repository, and can be checked out with the command (from an empty directory):

```
svn co https://svn.testnett.uninett.no/nerd/trunk
```

The NERD system is a network emergency detector that has been adapted to support IPFIX collection of data. This is not the primary task of the system, and therefore it is not streamlined.



The modified NERD supports collection of Netflow v5, v9 and IPFIX data. When NERD operates in collector mode, it stores elements to disk in much the same format as received IPFIX packets, and therefore operates independently of data content/semantics.

4.3 MAPI, IPFIXFLIB

The MAPI library and the IPFIXFLIB library reads packets from the network, and emits flow records that describes the flows in the network. IPFIXFLIB has support for both Ethernet and HDLC networks.

The overall program flow of IPFIXLIB is the following:

Packet processing pipeline:

1. Register library with MAPI (ipfixprobe.c)
2. Initialize IPFIX template record (util.c)
3. Unwrap Ethernet/HDLC, IP, TCP/UDP (npktproc.c)
4. Recognize flows, do flow metering (engine.c)
5. Flow export: Find flows to expire, and initialize IPFIX headers (nprobe.c)
6. Format information elements in the IPFIX flow (util.c)

Adding new enterprise-specific information elements to IPFIXFLIB may require changes to the following files:

1. util.c for adding the information element name and ID
2. nprobe_bucket.h for adding the information element's data storage to each flow metering.
3. util.c for adding export bindings for the information element.
4. engine.c for setting the information field value appropriately
5. npktproc.c if the new information element is dependent on IP/TCP header fields.

4.4 Operationalization

This section details the factors that may be missing from MAPI/IPFIXFLIB and the modified NERD collector in order to make the system operational in an environment to collect research data from the research network.

Tool support

- A discussion on whether the NERD collector should be used, or a different collector/toolkit, should be discussed before being used. There may be a lock-in effect if fixes are applied to one toolkit.

Communication

- Communication is currently bound to UDP, with early support for SCTP. The SCTP support should be improved after/before the system goes into production, in order to minimize risk of any impact faults, and for the purpose of IPFIX conformance.

Failure prevention

Failure prevention are best implemented with these two mechanisms (this happens to coincide with minimal effort implementation):

- Graceful degradation of collector and emitter upon resource constraints. For reasons of processing overhead in the measurement process (emitter), the degradation in case of too much data are best implemented as: (1) allow packet drop to be reported by emitter, (2) let MAPI client program selectively export flows “at random”, (3) allow collector to report on gaps in IPFIX message sequence numbers.
- Eficient error logging mechanism in the collector needs to be implemented. Currently, Syslog is used. Syslog messages, especially error messages or reports on connection drops from measurement probes, need to be accessible for system administrators

Chapter 5 Reports

These are some example reports exported by the modified NERD collector's FlowStat tool. FlowStat is developed by Uninett.

5.1 FlowPrint

This is an example of information present in flow records:

```
                   flowId, ID:   148, len=4, 22480
         sourceIPv4Address, ID:     8, len=4, 195.140.XXX.XXX
       sourceTransportPort, ID:     7, len=2, 443
    destinationIPv4Address, ID:    12, len=4, 129.241.XXX.XXX
  destinationTransportPort, ID:    11, len=2, 57796
        protocolIdentifier, ID:     4, len=1, 6
           octetDeltaCount, ID:     1, len=8, 45026
          packetDeltaCount, ID:     2, len=8, 37
             flowEndReason, ID:   136, len=1, 1
     flowStartNanoseconds, ID:   156, len=8, 1154615059907194848
       flowEndNanoseconds, ID:   157, len=8, 1154615060191744800
          ipClassOfService, ID:     5, len=1, 0
                   service, ID: 32800, len=2, 0
           maxTransfer1sec, ID: 33000, len=4, 0
           minTransfer1sec, ID: 33001, len=4, 4294967295
          maxTransfer100ms, ID: 33002, len=4, 393620
          minTransfer100ms, ID: 33003, len=4, 33300
           maxTransfer10ms, ID: 33004, len=4, 2286200
           minTransfer10ms, ID: 33005, len=4, 0
            maxTransfer1ms, ID: 33006, len=4, 13862000
            minTransfer1ms, ID: 33007, len=4, 0
                 reordered, ID: 32799, len=4, 0
                 direction, ID: 32798, len=1, 1
                tcpOptions, ID:   209, len=8, 5
                 rtpJitter, ID: 32801, len=4, 0
            rtpLostPackets, ID: 32803, len=4, 0
                pktDistVar, ID: 32790, len=4, 205393840.000000
             pktDistExpval, ID: 32791, len=4, 9017.000000
             tcpWindowSize, ID:   186, len=2, 8760
              maxTcpWindow, ID: 33008, len=4, 24820
              minTcpWindow, ID: 33009, len=4, 0
              effTcpWindow, ID: 33010, len=4, 434
 flowDurationMilliseconds, ID:   161, len=8, 333
       exporterIPv4Address, ID:   130, len=4, 158.38.XX.XXX
```

5.2 *Enterprise-Specific Information Elements*

5.2.1 Bitrate calculation

Bitrate calculation. This presents information from a few selected flows, with maximum max1ms field values:

./FlowPrint -s “srcport dstport octets packets max100ms max10ms max1ms”
-o “7 desc” flowfile

```
# srcport dstport octets  packets  max100ms  max10ms   max1ms
   80, 34273,  5389500,     3600,   840000,  6600000, 66000000,
 2010,   443, 15437710,    10913,  1030080,  6624000, 64456000,
37971,    25, 74586809,    49800, 10167200, 13200000, 62752000,
 2008,   443, 15440840,    10984,  1073840,  6624000, 60524000,
 2007,   443,  1941110,     1383,   801040,  6624000, 57808000,
 1935,  3191,  2099229,     1468,   659190,  6591900, 49691000,
   80,  3771,  3547534,     2384,  2388600,  6717200, 42000000,
   80, 47611,  2618760,     1752,   660000,  6150000, 34500000,
```

5.2.2 RTP Jitter value

RTP Jitter is calculated for media streams. Here obsering a test stream with VLC to a media streaming server (media.hiof.no):

```
# src           dst srcport dstport packets octets  rtpJitter pim prot
 158.39.160.xxx,   224.15.32.xxx,  5001,  5001,  3, 252, 1610, 0, 17,
 158.39.160.xxx,   224.15.32.xxx,  5003,  5003,  3, 252,  694, 0, 17,
129.241.132.xxx,    224.15.26.xx,  5003,  5003,  2, 152,  671, 1, 17,
129.241.132.xxx,    224.15.26.xx,  5001,  5001,  4, 304,  626, 2, 17,
```

Few such test streams are present in the network. For unknown reason, this detection may during certain periods have many false positives.

5.2.3 RFC1323 TCP Options

TCP options is now supported.

./FlowStat -F -f "tcpOptions&8" -f "tcpOptions&256" -s "src dst srcport dstport octets packets service maxTransfer10ms maxTransfer1ms tcpOptions" /raw/havardm/flows2006-07-27-19-38-04
A list of TCP options: http://www.iana.org/assignments/tcp-parameters
RFC1323 here: http://rfc.net/rfc1323.html

```
# src               dst              srcport dstport maxTransfer10ms  OPTS
206.161.206.xxx,   129.241.32.xx,     80, 49355,   600000,      258,
  129.241.32.xx, 206.161.206.xxx, 49355,    80,    10400,      258,
129.241.124.xxx,    88.203.44.xx, 19747,  2622,     5200,      258,
   88.203.44.xx, 129.241.124.xxx,  2635, 19747,    12000,      286,
129.241.124.xxx,    88.203.44.xx, 19747,  2635,     9800,      286,
129.241.131.xxx,  213.128.136.xx,  1521,    80,    88700,      278,
    70.87.52.xx,  129.241.176.xx,    80, 40545,  3750000,      258,
```

Filtering on 0x08 and 0x0100 is similar to finding all RFC1323 options. The window scaling option is further set by extended information field Ids:

```
./FlowStat -f "prot=6" -f "octets > 100000" -f "tcpOptions&8"
           -s "src dst packets tcpWindowSize maxwin minwin"
           -o "5 desc" /raw/havardm/flows2006-07-28-17-30-59
```

```
# src               dst               packets tcpWindowSize maxwin  minwin
  82.211.81.xxx, 129.241.157.xx,     234,  5792,  1482752,     6912
    8.10.160.xx, 129.241.157.xx,     155,  5792,  1482752,     6912
    8.10.160.xx, 129.241.157.xx,     130,  5792,  1482752,     6912
    8.10.160.xx, 129.241.157.xx,     267,  5792,  1482752,     6912
 129.241.139.xx, 81.225.189.xxx,     995,  5840,   747520,     5888
  217.30.180.xx, 129.241.176.xx,      77,  5792,   741376,     6912
  217.30.180.xx, 129.241.176.xx,      76,  5792,   741376,     6912
```

Note that tcpWindowSize is the old 16-bit value, which is not precise for RFC1323 flows.

5.2.4 AS numbers

AS numbers implemented by Robin.

./FlowStat -o 3 -f "octets > 100" -s "src_as dst_as avg(reordered) count(reordered) sum(packets) sum(reordered)" /raw/havardm/flows2006-07-31-12-42-43

```
#srcas dstas avg(reord) ANTALL-FLYT sum(pkts) sum(reord)
 2119,   224,         0,    14337,   960441,     5428 Telenor-NEXTEL
  224,  2119,         0,    14160,  1108934,     7451
15659,   224,         0,     9307,  1072145,     3058 NEXTGENTEL
  224, 15659,         0,     8728,  1488217,     2385
 8642,   224,        13,     2929,  5930645,    38395 Bredband AB
  224,  8642,         1,     4203,  3571883,     4264
 6467,   224,        87,       28,    10683,     2441 (diverse)
 3593,   224,        99,        8,     5592,      794
  224, 30798,       103,       18,    61786,     1860
29124,   224,       121,       13,     7402,     1579
35586,   224,       180,        4,     1667,      723
```

Note how the **sum(reordered)** column demonstrate out-of-sequence packet delivery. Out-of-sequence delivery may i.e. be due to hardware that prioritizes packets. It may also be due to packet retransmissions.

5.3 Potential use

Numbers reported on with IPFIXFLIB extended information fields may i.e. give information on:

– IPFIX data export has the advantage that flow sampling is not performed, and implementors are free to select an export strategy that fits the use of the data. This gives the potential that data with little interest, i.e. one-packet flows, can be chosen to be ignored, or aggregated into "meta" flows. Meta flows ***may*** also be a solution to the DDOS problem, allowing precise reporting on who is transmitting, while removing the massive data amounts produced by varying sources and port numbers.

– The quality (in speed, burst speeds in i.e. 1ms, TCP retransmission, TCP max/effective window size) that maximum can be accomplished between networks. This may indicate networks that may require hardware upgrades, or have traffic problems.

– RTP parameters may give clues to the user experienced quality for video and audio streaming. Especially if SIP RTP parameters can be observed (which currently does not happen), this may demonstrate network areas with problems.

Chapter 6 Modifications to IPFIXFLIB and NERD

This chapter summarizes modifications to IPFIXFLIB and NERD during a 4-week summer job at UNINETT:

MAPI:

- BUGFIX: Minor protocol changes per latest IPFIX drafts
- BUGFIX: Autoconf: IPFIXFLIB now uses Autoconf's config.h, and also MAPI configuration file.
- BUGFIX: Valgrind errors fixed
- BUGFIX: Calculation of expectancy/variance of packet distance/sizes may have been incorrect. They previously were bidirectional, now they're unidirectional. Probably more useful!
- Optimization: Between 10-13% performance on packet processing.
- Added: Observation domain ID is now configurable Observation domain ID should be manually set to an unique value for each probe using one single collector, in order to avoid conflicts.
- Added: Service discovery for Torrent and DC++, experimental string-searching implementation. Tried to improve performance, but failed.
- Added: packet drop counter for DAG cards (hw_info). This is using a direct query against the DAG card. Using the value that is reported on top of each captured packet block does not work. MAPI reports in console if packet loss.
- Added: Configurable ingress and egress ifnum information elements at the probe.
- Added: 1ms-interval for bitrate calculation. Generalized the code for modifiability.
- Added: Look inside PIM Register-encapsulated multicast packages.
- Added: RTCP detection. Parses packet drop and jitter information out of RTCP Receiver reports.
- Added: HDLC data link header support
- Added: TCP Options information element
- Added: Enterprise-specific TCP window information elements.
- Added: Automatic configuration entry for IPFIXFLIB with default settings.

NERD:

- BUGFIX: Saving 16 bytes per flow record by not storing redundant timestamps (NERD legacy)
- BUGFIX: Minor protocol changes per latest IPFIX drafts
- BUGFIX: Corrected the sequence number for IPFIX
- BUGFIX: Improved some error messages, added some, and fixed valgrind errors.
- BUGFIX: Copied MAPI information element names into nerdtype.cfg for user-friendliness.
- BUGFIX: Removed debug info for major performance boost.
- Changed: FlowStat improvements to 'order by' parameter.
- Added: GZIP compression to flow records. Now only using 25% of the space previously required.
- Added: FlowStat may filter on bit mask: FlowStat -f “tcpOptions & 32” ...
- Added: Autoconf: Automatically install configuration file to etc directory. Automatically download the AS number database.

Chapter 7 User Guide

This chapter describes a few tools.

7.1 FlowStat usage

This chapter introduces the basic functionality of FlowStat, a tool included in the "local" version of NERD to make aggregated statistics from NERD data files.

Installation
FlowStat is installed together with *nerdd*. When running *make install*, it is copied. Also, a file *datatypes.cfg* need to be located in the working-directory when running FlowStat.

Data extraction
FlowStat may be used the following way to extract data:

```
FlowStat -s "src dst srcport dstport" flow_filename
```

flow_filename is here a file, a list of files, or name with wild card.

The -s parameter takes a list of fields to extract. Some of them are:

src	Source IP (ipv4)
dst	Destination IP
dstport	Destination port
octetDeltaCount	Number of bytes (64-bits)
packets	Number of packets
flowDurationMS	Duration in milliseconds

See Chapter 3, of the nerdtype.txt file, for a complete overview of information fields. To display the information fields present in a flow file, use the FlowPrint tool. Attemts to select information fields from a flow file that does not contain that particular information field may yield only zero values for that column.

Filtering
Filtering may be performed on one or more criteria:

```
FlowStat -f "octetDeltaCount > 10000"
         -s "src dst dstport octetDeltaCount"
```

These operators are allowed:

Operator	*Description*	*Example*
= or ==	Equality,	src == 129.241.132.150
= or ==	Equality (netmasks)	dst == 129.240.0.0/15
~=	Compare value to a set, i.e.	dstport ~= 20,21,22,80,114
> og <. !=	Compare difference or not-equal	octets > 100000
&	Bit-wise comparison,	tcpOptions & 32

All filtering is "AND"-based. If the FlowStat parameter -F is given (in uppercase!), FlowStat will "OR" the filters instead.
I.e.:

```
FlowStat -F -f "src=129.240.0.0/15" -f "dst=129.240.0.0/15"
```

... will filter out all rows that have 129.240 in either their source or destination addresses (due to -F)

Aggregation
FlowStat may aggregate over data:

```
FlowStat -s “src dst sum(octetDeltaCount)”
```

The aggregation operators that are permitted, are:

sum(x)	sums all values of an information element
avg(x)	calculates the average of an element
min(x)	least x
max(x)	largest x
count(x)	number of rows of x (the total number of rows)
prefix(x)	converts an IP address its corresponding prefix (if mask.cfg is present)

Aggregation works like this: All fields to the *left* of aggregated fields are regarded as the key. All fields at the *right* side are aggregated, all fields to the *left* are constant. There are no limits on the combination of aggregation operators, i.e. this works:

```
-s “src dst count(octetDeltaCount)
    sum(octetDeltaCount) avg(octetDeltaCount)”
```

This will for each combination of {src,dst} aggregate number of octets (sum and avg).

IP prefixes
Statistics of traffic between subnets may be created. A file *mask.cfg* need to be located in the current working directory, The syntax is similar to flow-tools's mask.cfg file:

```
mask-definition uninett-prefixes
 prefix 128.39.0.0/24 24
 prefix 128.39.1.0/24 24
 prefix 128.39.2.0/24 24
 prefix 128.39.3.0/24 24
 prefix 0.0.0.0/0 0
```

Data extraction can then be done this way:

```
FlowStat -s “prefix(src) prefix(dst) avg(reordered)”
```

The IP addresses in *src* and *dst* will then be translated to their respective subnet addresses.

Sorting, and limiting rows
FlowStat may limit the number of outputted rows, and do sorting on certain fields. For instance, this can be used to extract IP addresses with most outbound traffic:

```
FlowStat -s “src sum(octetDeltaCount)”
        -o “2 desc” -l 10
```

This command will restrict output to 10 rows, and sort descending on the second column.
Ordering can also be specified as i.e. `-o src`. However, this syntax does not work if the field name is ambiguous, i.e. if aggregation is being used.

Autonomous Systems
FlowStat will automatically look up AS numbers for IP addresses if the BGPTABLE.TXT file is present in the ETC directory. For instance:

FlowStat -s “src dst src_as dst_as” filename

... will present the AS numbers for source and destination, even when they are not present in the flow file. The IP addresses will be used as look-up values in the BGP database.

7.2 Nerdd usage

To compile NERD, the following packages are needed:

mysql-devel
libboost-dev
libboost-thread-dev
libboost-regex-dev
automake
autoconf

Installing NERD:

./configure
(or: ./configure –prefix=/home/your_user_name/install_dir)
make
make install

Configuration files (nerd.cfg, nerdtype.cfg, mask.cfg) will be automatically copied to the 'etc' directory when installing. The most recent AS number database from www.potaroo.org will be automatically downloaded when running 'make install'.

Before running NERD in collector mode, please modify output path in nerd.cfg

To run NERD in collector mode, the following parameters may be given:

nerdd -C 1234 -T -t 5678

where

-C 1234 means collector mode, with log rotation every 1234 sec.
-T means that a textual configuration file should be used (nerd.cfg), and not MySQL config.
-t 5678 means that the program should run for 5678 sec, and then terminate.
If this parameter is ommitted, NERD will run in daemon mode.

7.3 Installation and testing

Installing and running NERD together with MAPI can be accomplished using these steps:

MAPI installation

1. Check out a copy of MAPI from SVN repository `https://svn.testnett.uninett.no/mapi/trunk`
2. Compile MAPI by running `./configure --prefix=/your/dir --enable-dag --enable-ipfixflib`
3. Edit mapid.conf. Set correct enterprise ID. You may need to copy it from $SRC/etc/ to $PREFIX/etc
4. Create a MAPI user program (a program that connects to MAPI). Working versions may be copied from https://svn.testnett.uninett.no/nerd/trunk/src/t/ (test2.c, c, client_sctp.c, c_sctp, r, r_sctp)
    1. Use test2.c if running without SCTP support
    2. Use client_sctp.c if running with experimental SCTP support (non-conformant, requires SCTP kernel support)
    3. Change test2.c or client_sctp.c to set destination IP, destination port, and proper networking device (eth0, /dev/dag0, ath0, etc.). If you change port number, also change collector settings.
    4. Compile test2.c with command:

gcc -I [mapi-include-dir] test2.c -o test2 [mapi-lib-dir]/mapi.so
(also see the 'c_sctp' script, which compiles client_sctp.c)

5. Configure and run *mapid*. MAPI should be configured to load IPFIXLIB. The file /etc/mapid.conf need to be modified, with proper name of network interface card
(problems with loading IPFIXFLIB? Try to let it be alone in the LIBS= line in mapi.conf)

NERD installation

6. Check out a copy of NERD from SVN repository
`https://svn.testnett.uninett.no/nerd/trunk`
7. Install the dependencies for NERD. Most notably, the package *mysql-devel* must be installed.
8. SCTP SUPPORT: If you want SCTP support installed, get a SCTP kernel (2.6.16+) and the lksctp-tools). SCTP support is optional for running NERD (however, “mandatory” for IPFIX conformance)
9. Compile NERD from the directory ./nerd/trunk/src by using ./configure, make [-prefix=...], and make install. You may also compile NERD from its main directory ./nerd/trunk, but this also installs part which you don't want (like the PHP front end).
10. Modify nerd.cfg. Set preferred output directory for flow files. (note: the dir **must** exist!)

Running NERD

11. Start *nerdd.* Command line should be:

```
nerdd -C 3200 -T -t 3200
```

The -C 3200 parameter instructs log rotation every 3600 seconds
The -T parameter instructs nerdd to use text-file for configuration, instead of the default MySQL database.
The -t 3200 parameter tells nerdd to run for 3200 seconds in non-daemon-mode, instead of the default unconstrained daemon-mode.
If compiled with SCTP, then it will listen to both UDP and SCTP ports.

12. Run MAPI/IPFIXFLIB: Start the user program from command line (the program created in step 3), i.e. by typing ./test2. If using SCTP (client_sctp.c), then the user program must be run as root.
The user program should now be relaying data to the collector. If MAPI is reading from a low-traffic link, then you must wait a few minutes before relay messages are appearing.

11. Log files are created by the collector (NERD) in your designated log directory. When you want to display the contents of log files, do the following:
Go to the *log directory*, as it is specified in the nerd.cfg file.

12. Run ./FlowStat flows2005-xx-xx-xx-xx-xx
to display some basic statistics about the flows. For more advanced parameters, see the chapter about FlowStat.

13. To display all information contained in a flow file, use the command:
./FlowPrint flows2005-xx-xx-xx-xx-xx

Chapter 8 Conclusions

This chapter sums up the standards compliance and possible remaining tasks for an IPFIX compliant monitoring implementation.

8.1 Standards Compliance

There are some elements that are missing in order for the implementation to be fully IPFIX conform. Most notably are these:

- SCTP is not fully implemented. This is a requirement for IPFIX compliance.
- Support for Ipv6 is not well-developed in IPFIXFLIB. Various IPV6 options are not relayed from the Ipv6 header to their respective IPFIX information element fields, in particular Ipv6 options.
- The enterprise ID is currently not regarded in NERDD. The set {enterpriseid, templateid} is used to identify information elements that are non-standard. This ID could in the future be added to nerdtype.cfg to uniquely identify information elements.

8.2 Further Work

Some of the work that could be done on the MAPI/NERD flow monitoring system:

MAPI

- There is no support for long-living flows. These will be exported as a number of flows every 4 minutes. They constitute between 1-2% of a small data set sample. Not purging the buckets will allow for buckets to more precisely represent quality attributes, like TCP options, and speed, and also remember parameters carried in SYN packets.
- TCP out-of-sequence packets are now registered (field "reordered"). It may be interesting to detect what is retransmissions by looking at SACK (selective ACK) and other TCP events.
- The stability of MAPI should be examined. I.e. it may be interesting to try to run it for a week in a production environment.
- Support for IPFIX Options Templates/Records. Report packet loss, congestion etc. to the collector, in order for these numbers to be aggregated in a monitoring system with multiple probes.
- MAPI should support detection of new high-performance TCP protocol (new TCP version, not RFC1323)
- Better service discovery (based on Trackflib?). The current packet-search strategy is not good, and does not scale. Should find a better approach. Also, should add Gnutella/etc support if service discovery should be useful for P2P protocol detection.
- RTP Media Time
- WSOPT/RFC1323: Does MAPI handle re-negotiation of WSOPT correctly?
- BUG: TCP Effective Window Size is currently based on the ACK field. The SACK option may also be necessary to look at.
- BUG: mapi.conf not automatically copied. "libs" in mapi.conf doesn't work unless ipfixflib.so is the last entry, or the only entry. The libpath and drvpath of default mapi.conf is incorrect, possibly.
- BUG: This is not good: `minTransfer1sec, ID: 33001, length: 4,  4294967295` minTransfer1sec should be 0 if a measurement is not possible, not MAX_INT (?)

MAPI Client Process

- There currently does not exist a generic application for exporting data from MAPI. Such an application should consider strict SCTP conformance, DDOS protection, and data integrity.

NEDR

- **WARNING:** NERD's autconfig scripts are evil! Don't depend on them for seeing dependencies. Find out why 'make clean' sometimes is necessary to avoid strange errors.
- Rename the product, remove the alarm part of it and make it collector specific?
- The behaviour of FlowStat and other tools should be considered when under DDOS attacks, and other large data volume situations. **Currently, a possible situation is that the memory usage of the collector will grow until it crashes.** Need to implement a limit to the number of queued packets (FlowPacketQueue), and a reporting mechanism for when packets are ignored.
- NERD need to provide a useful logging mechanism for many events, such as packet loss, malformed packets, and connection changes. This may be syslog (currently implemented).