Network Working Group
Request for Comments: 3176
Category: Informational

P. Phaal
S. Panchen
N. McKee
InMon Corp.

September 2001

InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks

Status of this Memo

Copyright Notice

Abstract

This memo defines InMon Coporation's sFlow system. sFlow is a technology for monitoring traffic in data networks containing switches and routers. In particular, it defines the sampling mechanisms implemented in an sFlow Agent for monitoring traffic, the sFlow MIB for controlling the sFlow Agent, and the format of sample data used by the sFlow Agent when forwarding data to a central data collector.

Table of Contents

1. Overview

sFlow is a technology for monitoring traffic in data networks containing switches and routers. In particular, it defines the sampling mechanisms implemented in an sFlow Agent for monitoring traffic, the sFlow MIB for controlling the sFlow Agent, and the format of sample data used by the sFlow Agent when forwarding data to a central data collector.

The architecture and sampling techniques used in the sFlow monitoring system are designed to provide continuous site-wide (and network-wide) traffic monitoring for high speed switched and routed networks.

The design specifically addresses issues associated with:

o Accurately monitoring network traffic at Gigabit speeds and higher.

o Scaling to manage tens of thousands of agents from a single point.

o Extremely low cost agent implementation.

The sFlow monitoring system consists of an sFlow Agent (embedded in a switch or router or in a stand alone probe) and a central data collector, or sFlow Analyzer.

The sFlow Agent uses sampling technology to capture traffic statistics from the device it is monitoring. sFlow Datagrams are used to immediately forward the sampled traffic statistics to an sFlow Analyzer for analysis.

This document describes the sampling mechanisms used by the sFlow Agent, the SFLOW MIB used by the sFlow Analyzer to control the sFlow Agent, and the sFlow Datagram Format used by the sFlow Agent to send traffic data to the sFlow Analyzer.

2. Sampling Mechanisms

The sFlow Agent uses two forms of sampling: statistical packet-based sampling of switched flows, and time-based sampling of network interface statistics.

2.1 Sampling of Switched Flows

A flow is defined as all the packets that are received on one interface, enter the Switching/Routing Module and are sent to another interface. In the case of a one-armed router, the source and destination interface could be the same. In the case of a broadcast or multicast packet there may be multiple destination interfaces. The sampling mechanism must ensure that any packet involved in a flow has an equal chance of being sampled, irrespective of the flow to which it belongs.

Sampling flows is accomplished as follows: When a packet arrives on an interface, a filtering decision is made that determines whether the packet should be dropped. If the packet is not filtered a destination interface is assigned by the switching/routing function. At this point a decision is made on whether or not to sample the packet. The mechanism involves a counter that is decremented with each packet. When the counter reaches zero a sample is taken. Whether or not a sample is taken, the counter Total_Packets is incremented. Total_Packets is a count of all the packets that could have been sampled.

Taking a sample involves either copying the packet's header, or extracting features from the packet (see sFlow Datagram Format for a description of the different forms of sample). Every time a sample is taken, the counter Total_Samples, is incremented. Total_Samples is a count of the number of samples generated. Samples are sent by the sampling entity to the sFlow Agent for processing. The sample includes the packet information, and the values of the Total_Packets and Total_Samples counters.

When a sample is taken, the counter indicating how many packets to skip before taking the next sample should be reset. The value of the counter should be set to a random integer where the sequence of random integers used over time should be such that

(1) Total_Packets/Total_Samples = Rate

An alternative strategy for packet sampling is to generate a random number for each packet, compare the random number to a preset threshold and take a sample whenever the random number is smaller than the threshold value. Calculation of an appropriate threshold value depends on the characteristics of the random number generator, however, the resulting sample stream must still satisfy (1).

2.1.1 Distributed Switching

The SFLOW MIB permits separate sampling entities to be associated with different physical or logical elements of the switch (such as interfaces, backplanes or VLANs). Each sampling engine has its own independent state (i.e., Total_Packets, Total_Samples, Skip and Rate), and forwards its own sample messages to the sFlow Agent. The sFlow Agent is responsible for packaging the samples into datagrams for transmission to an sFlow Analyzer.

2.1.2 Random Number Generation

The essential property of the random number generator is that the mean value of the numbers it generates converges to the required sampling rate.

A uniform distribution random number generator is very effective. The range of skip counts (the variance) does not significantly affect results; variation of +-10% of the mean value is sufficient.

The random number generator must ensure that all numbers in the range between its maximum and minimum values of the distribution are possible; a random number generator only capable of generating even numbers, or numbers with any common divisor is unsuitable.

A new skip value is only required every time a sample is taken.

2.2 Sampling of Network Interface Statistics

The objective of the counter sampling is to efficiently, periodically poll each data source on the device and extract key statistics.

For efficiency and scalability reasons, the sFlow System implements counter polling in the sFlow Agent. A maximum polling interval is assigned to the agent, but the agent is free to schedule polling in order maximize internal efficiency.

Flow sampling and counter sampling are designed as part of an integrated system. Both types of samples are combined in sFlow Datagrams. Since flow sampling will cause a steady, but random, stream of datagrams to be sent to the sFlow Analyzer, counter samples may be taken opportunistically in order to fill these datagrams.

One strategy for counter sampling has the sFlow Agent keep a list of counter sources being sampled. When a flow sample is generated the sFlow Agent examines the list and adds counters to the sample datagram, least recently sampled first. Counters are only added to the datagram if the sources are within a short period, 5 seconds say,

of failing to meet the required sampling interval (see sFlowCounterSamplingInterval in SFLOW MIB). Whenever a counter source's statistics are added to a sample datagram, the time the counter source was last sampled is updated and the counter source is placed at the end of the list. Periodically, say every second, the sFlow Agent examines the list of counter sources and sends any counters that need to be sent to meet the sampling interval requirement.

Alternatively, if the agent regularly schedules counter sampling, then it should schedule each counter source at a different start time (preferably randomly) so that counter sampling is not synchronized within an agent or between agents.

3. sFlow MIB

The sFlow MIB defines a control interface for an sFlow Agent. This interface provides a standard mechanism for remotely controlling and configuring an sFlow Agent.

3.1 The SNMP Management Framework

The SNMP Management Framework presently consists of five major components:

- o An overall architecture, described in RFC 2571 [2].

- o Mechanisms for describing and naming objects and events for the purpose of management. The first version of this Structure of Management Information (SMI) is called SMIv1 and described in STD 16,

  RFC 1155 [3], STD 16, RFC 1212 [4] and RFC 1215 [5]. The second version, called SMIv2, is described in STD 58, RFC 2578 [6], STD 58, RFC 2579 [7] and STD 58, RFC 2580 [8].

- o Message protocols for transferring management information. The first version of the SNMP message protocol is called SNMPv1 and described in STD 15, RFC 1157 [9]. A second version of the SNMP message protocol, which is not an Internet standards track protocol, is called SNMPv2c and described in RFC 1901 [10] and RFC 1906 [11]. The third version of the message protocol is called SNMPv3 and described in RFC 1906 [11], RFC 2572 [12] and RFC 2574 [13].

- o Protocol operations for accessing management information. The first set of protocol operations and associated PDU formats is described in STD 15, RFC 1157 [9]. A second set of protocol operations and associated PDU formats is described in RFC 1905 [14].

- o A set of fundamental applications described in RFC 2573 [15] and the view-based access control mechanism described in RFC 2575 [16].

A more detailed introduction to the current SNMP Management Framework can be found in RFC 2570 [17].

Managed objects are accessed via a virtual information store, termed the Management Information Base or MIB. Objects in the MIB are defined using the mechanisms defined in the SMI.

This memo specifies a MIB module that is compliant to the SMIv2. A MIB conforming to the SMIv1 can be produced through the appropriate translations. The resulting translated MIB must be semantically equivalent, except where objects or events are omitted because no translation is possible (use of Counter64). Some machine readable information in SMIv2 will be converted into textual descriptions in SMIv1 during the translation process. However, this loss of machine readable information is not considered to change the semantics of the MIB.

3.2 Definitions

```
SFLOW-MIB DEFINITIONS ::= BEGIN

IMPORTS

MODULE-IDENTITY, OBJECT-TYPE, Integer32, enterprises
        FROM SNMPv2-SMI
SnmpAdminString
        FROM SNMP-FRAMEWORK-MIB
OwnerString
        FROM RMON-MIB
InetAddressType, InetAddress
        FROM INET-ADDRESS-MIB
MODULE-COMPLIANCE, OBJECT-GROUP
        FROM SNMPv2-CONF;

sFlowMIB  MODULE-IDENTITY
  LAST-UPDATED "200105150000Z"   -- May 15, 2001
  ORGANIZATION "InMon Corp."
  CONTACT-INFO
```

```
         "Peter Phaal
          InMon Corp.
          http://www.inmon.com/

          Tel:  +1-415-661-6343
          Email: peter_phaal@inmon.com"
  DESCRIPTION
          "The MIB module for managing the generation and transportation
           of sFlow data records."

  --
  -- Revision History
  --
  REVISION    "200105150000Z"     -- May 15, 2001
  DESCRIPTION
          "Version 1.2

           Brings MIB into SMI v2 compliance."

  REVISION    "200105010000Z"      -- May 1, 2001
  DESCRIPTION
           "Version 1.1

            Adds sFlowDatagramVersion."
  ::= { enterprises 4300 1 }

sFlowAgent OBJECT IDENTIFIER ::= { sFlowMIB 1 }

sFlowVersion OBJECT-TYPE
     SYNTAX      SnmpAdminString
     MAX-ACCESS  read-only
     STATUS      current
     DESCRIPTION
       "Uniquely identifies the version and implementation of this MIB.
        The version string must have the following structure:
           <MIB Version>;<Organization>;<Software Revision>
        where:
           <MIB Version>  must be '1.2', the version of this MIB.
           <Organization> the name of the organization responsible
                          for the agent implementation.
           <Revision>     the specific software build of this agent.

        As an example, the string '1.2;InMon Corp.;2.1.1' indicates
        that this agent implements version '1.2' of the SFLOW MIB, that
        it was developed by 'InMon Corp.' and that the software build
        is '2.1.1'.

        The MIB Version will change with each revision of the SFLOW
```

```
        MIB.

        Management entities must check the MIB Version and not attempt
        to manage agents with MIB Versions greater than that for which
        they were designed.

        Note: The sFlow Datagram Format has an independent version
              number which may change independently from <MIB Version>.
              <MIB Version> applies to the structure and semantics of
              the SFLOW MIB only."
     DEFVAL { "1.2;;" }
     ::= { sFlowAgent 1 }

sFlowAgentAddressType OBJECT-TYPE
     SYNTAX      InetAddressType
     MAX-ACCESS  read-only
     STATUS      current
     DESCRIPTION
       "The address type of the address associated with this agent.
        Only ipv4 and ipv6 types are supported."
     ::= { sFlowAgent 2 }

sFlowAgentAddress OBJECT-TYPE
     SYNTAX      InetAddress
     MAX-ACCESS  read-only
     STATUS      current
     DESCRIPTION
       "The IP address associated with this agent.  In the case of a
        multi-homed agent, this should be the loopback address of the
        agent.  The sFlowAgent address must provide SNMP connectivity
        to the agent.  The address should be an invariant that does not
        change as interfaces are reconfigured, enabled, disabled,
        added or removed.  A manager should be able to use the
        sFlowAgentAddress as a unique key that will identify this
        agent over extended periods of time so that a history can
        be maintained."
    ::= { sFlowAgent 3 }

sFlowTable OBJECT-TYPE
     SYNTAX      SEQUENCE OF SFlowEntry
     MAX-ACCESS  not-accessible
     STATUS      current
     DESCRIPTION
       "A table of the sFlow samplers within a device."
     ::= { sFlowAgent 4 }

sFlowEntry OBJECT-TYPE
     SYNTAX      SFlowEntry
```

```
     MAX-ACCESS  not-accessible
     STATUS      current
     DESCRIPTION
       "Attributes of an sFlow sampler."
     INDEX { sFlowDataSource }
     ::= { sFlowTable 1 }

SFlowEntry ::= SEQUENCE {
     sFlowDataSource               OBJECT IDENTIFIER,
     sFlowOwner                    OwnerString,
     sFlowTimeout                  Integer32,
     sFlowPacketSamplingRate       Integer32,
     sFlowCounterSamplingInterval  Integer32,
     sFlowMaximumHeaderSize        Integer32,
     sFlowMaximumDatagramSize      Integer32,
     sFlowCollectorAddressType     InetAddressType,
     sFlowCollectorAddress         InetAddress,
     sFlowCollectorPort            Integer32,
     sFlowDatagramVersion          Integer32
}

sFlowDataSource OBJECT-TYPE
     SYNTAX      OBJECT IDENTIFIER
     MAX-ACCESS  read-only
     STATUS      current
     DESCRIPTION
       "Identifies the source of the data for the sFlow sampler.
       The following data source types are currently defined:

       - ifIndex.<I>
       DataSources of this traditional form are called 'port-based'.
       Ideally the sampling entity will perform sampling on all flows
       originating from or destined to the specified interface.
       However, if the switch architecture only permits input or
       output sampling then the sampling agent is permitted to only
       sample input flows input or output flows.  Each packet must
       only be considered once for sampling, irrespective of the
       number of ports it will be forwarded to.

       Note: Port 0 is used to indicate that all ports on the device
             are represented by a single data source.
             - sFlowPacketSamplingRate applies to all ports on the
               device capable of packet sampling.
             - sFlowCounterSamplingInterval applies to all ports.

       - smonVlanDataSource.<V>
       A dataSource of this form refers to a 'Packet-based VLAN'
       and is called a 'VLAN-based' dataSource.  <V> is the VLAN
```

```
       ID as defined by the IEEE 802.1Q standard.  The
       value is between 1 and 4094 inclusive, and it represents
       an 802.1Q VLAN-ID with global scope within a given
       bridged domain.
       Sampling is performed on all packets received that are part
       of the specified VLAN (no matter which port they arrived on).
       Each packet will only be considered once for sampling,
       irrespective of the number of ports it will be forwarded to.

       - entPhysicalEntry.<N>
       A dataSource of this form refers to a physical entity within
       the agent (e.g., entPhysicalClass = backplane(4)) and is called
       an 'entity-based' dataSource.
       Sampling is performed on all packets entering the resource (e.g.
       If the backplane is being sampled, all packets transmitted onto
       the backplane will be considered as single candidates for
       sampling irrespective of the number of ports they ultimately
       reach).

       Note: Since each DataSource operates independently, a packet
             that crosses multiple DataSources may generate multiple
             flow records."
     ::= { sFlowEntry 1 }

sFlowOwner OBJECT-TYPE
     SYNTAX      OwnerString
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
       "The entity making use of this sFlow sampler.  The empty string
        indicates that the sFlow sampler is currently unclaimed.
        An entity wishing to claim an sFlow sampler must make sure
        that the sampler is unclaimed before trying to claim it.
        The sampler is claimed by setting the owner string to identify
        the entity claiming the sampler.  The sampler must be claimed
        before any changes can be made to other sampler objects.

        In order to avoid a race condition, the entity taking control
        of the sampler must set both the owner and a value for
        sFlowTimeout in the same SNMP set request.

        When a management entity is finished using the sampler,
        it should set its value back to unclaimed.  The agent
        must restore all other entities this row to their
        default values when the owner is set to unclaimed.

        This mechanism provides no enforcement and relies on the
        cooperation of management entities in order to ensure that
```

```
        competition for a sampler is fairly resolved."
     DEFVAL { "" }
     ::= { sFlowEntry 2 }

sFlowTimeout OBJECT-TYPE
     SYNTAX      Integer32
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
       "The time (in seconds) remaining before the sampler is released
        and stops sampling.  When set, the owner establishes control
        for the specified period.  When read, the remaining time in the
        interval is returned.

        A management entity wanting to maintain control of the sampler
        is responsible for setting a new value before the old one
        expires.

        When the interval expires, the agent is responsible for
        restoring all other entities in this row to their default
        values."
     DEFVAL { 0 }
     ::= { sFlowEntry 3 }

sFlowPacketSamplingRate OBJECT-TYPE
     SYNTAX      Integer32
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
       "The statistical sampling rate for packet sampling from this
        source.

        Set to N to sample 1/Nth of the packets in the monitored flows.
        An agent should choose its own algorithm introduce variance
        into the sampling so that exactly every Nth packet is not
        counted.  A sampling rate of 1 counts all packets.  A sampling
        rate of 0 disables sampling.

        The agent is permitted to have minimum and maximum allowable
        values for the sampling rate.  A minimum rate lets the agent
        designer set an upper bound on the overhead associated with
        sampling, and a maximum rate may be the result of hardware
        restrictions (such as counter size).  In addition not all values
        between the maximum and minimum may be realizable as the
        sampling rate (again because of implementation considerations).

        When the sampling rate is set the agent is free to adjust the
        value so that it lies between the maximum and minimum values
```

```
        and has the closest achievable value.

        When read, the agent must return the actual sampling rate it
        will be using (after the adjustments previously described).  The
        sampling algorithm must converge so that over time the number
        of packets sampled approaches 1/Nth of the total number of
        packets in the monitored flows."
     DEFVAL { 0 }
     ::= { sFlowEntry 4 }

sFlowCounterSamplingInterval OBJECT-TYPE
  SYNTAX      Integer32
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
       "The maximum number of seconds between successive samples of the
        counters associated with this data source.  A sampling interval
        of 0 disables counter sampling."
     DEFVAL { 0 }
     ::= { sFlowEntry 5 }

sFlowMaximumHeaderSize OBJECT-TYPE
     SYNTAX      Integer32
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
       "The maximum number of bytes that should be copied from a
        sampled packet.  The agent may have an internal maximum and
        minimum permissible sizes.  If an attempt is made to set this
        value outside the permissible range then the agent should
        adjust the value to the closest permissible value."
     DEFVAL { 128 }
     ::= { sFlowEntry 6 }

sFlowMaximumDatagramSize OBJECT-TYPE
     SYNTAX      Integer32
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
        "The maximum number of data bytes that can be sent in a single
         sample datagram.  The manager should set this value to avoid
         fragmentation of the sFlow datagrams."
     DEFVAL { 1400 }
     ::= { sFlowEntry 7 }

sFlowCollectorAddressType OBJECT-TYPE
     SYNTAX      InetAddressType
     MAX-ACCESS  read-write
```

```
     STATUS      current
     DESCRIPTION
       "The type of sFlowCollectorAddress."
     DEFVAL { ipv4 }
     ::= { sFlowEntry 8 }

sFlowCollectorAddress OBJECT-TYPE
     SYNTAX      InetAddress
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
       "The IP address of the sFlow collector.
        If set to 0.0.0.0 all sampling is disabled."
     DEFVAL { "0.0.0.0" }
     ::= { sFlowEntry 9 }

sFlowCollectorPort OBJECT-TYPE
     SYNTAX      Integer32
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
       "The destination port for sFlow datagrams."
     DEFVAL { 6343 }
     ::= { sFlowEntry 10 }

sFlowDatagramVersion OBJECT-TYPE
     SYNTAX      Integer32
     MAX-ACCESS  read-write
     STATUS      current
     DESCRIPTION
       "The version of sFlow datagrams that should be sent.

        When set to a value not support by the agent, the agent should
        adjust the value to the highest supported value less than the
        requested value, or return an error if no such values exist."
     DEFVAL { 4 }
     ::= { sFlowEntry 11 }

 --
 -- Compliance Statements
 --

sFlowMIBConformance OBJECT IDENTIFIER ::= { sFlowMIB 2 }
sFlowMIBGroups      OBJECT IDENTIFIER ::= { sFlowMIBConformance 1 }
sFlowMIBCompliances OBJECT IDENTIFIER ::= { sFlowMIBConformance 2 }

sFlowCompliance MODULE-COMPLIANCE
     STATUS      current
```

```
     DESCRIPTION
       "Compliance statements for the sFlow Agent."

     MODULE -- this module
         MANDATORY-GROUPS { sFlowAgentGroup }
         OBJECT     sFlowAgentAddressType
         SYNTAX     InetAddressType { ipv4(1) }
         DESCRIPTION
           "Agents need only support ipv4."

         OBJECT sFlowCollectorAddressType
         SYNTAX InetAddressType { ipv4(1) }
         DESCRIPTION
           "Agents need only support ipv4."

     ::= { sFlowMIBCompliances 1 }

sFlowAgentGroup OBJECT-GROUP
     OBJECTS { sFlowVersion, sFlowAgentAddressType, sFlowAgentAddress,
               sFlowDataSource, sFlowOwner, sFlowTimeout,
               sFlowPacketSamplingRate, sFlowCounterSamplingInterval,
               sFlowMaximumHeaderSize, sFlowMaximumDatagramSize,
               sFlowCollectorAddressType, sFlowCollectorAddress,
               sFlowCollectorPort, sFlowDatagramVersion }
      STATUS current
      DESCRIPTION
        "A collection of objects for managing the generation and
         transportation of sFlow data records."
       ::= { sFlowMIBGroups 1 }

END
```

The sFlow MIB references definitions from a number of existing RFCs [18], [19], [20] and [21].

4. sFlow Datagram Format

The sFlow datagram format specifies a standard format for the sFlow Agent to send sampled data to a remote data collector.

The format of the sFlow datagram is specified using the XDR standard [1]. XDR is more compact than ASN.1 and simpler for the sFlow Agent to encode and the sFlow Analyzer to decode.

Samples are sent as UDP packets to the host and port specified in the SFLOW MIB. The lack of reliability in the UDP transport mechanism does not significantly affect the accuracy of the measurements obtained from an sFlow Agent.

- If counter samples are lost then new values will be sent during the next polling interval. The chance of an undetected counter wrap is negligible. The sFlow datagram specifies 64 bit octet counters, and with typical counter polling intervals between 20 to 120 seconds, the chance of a long enough sequence of sFlow datagrams being lost to hide a counter wrap is very small.

- The net effect of lost flow samples is a slight reduction in the effective sampling rate.

The use of UDP reduces the amount of memory required to buffer data. UDP also provides a robust means of delivering timely traffic information during periods of intense traffic (such as a denial of service attack). UDP is more robust than a reliable transport mechanism because under overload the only effect on overall system performance is a slight increase in transmission delay and a greater number of lost packets, neither of which has a significant effect on an sFlow-based monitoring system. If a reliable transport mechanism were used then an overload would introduce long transmission delays and require large amounts of buffer memory on the agent.

While the sFlow Datagram structure permits multiple samples to be included in each datagram, the sampling agent must not wait for a buffer to fill with samples before sending the sample datagram. sFlow sampling is intended to provide timely information on traffic. The agent may at most delay a sample by 1 second before it is required to send the datagram.

The agent should try to piggyback counter samples on the datagram stream resulting from flow sampling. Before sending out a datagram the remaining space in the buffer can be filled with counter samples. The agent has discretion in the timing of its counter polling, the specified counter sampling intervals sFlowCounterSamplingInterval is a maximum, so the agent is free to sample counters early if it has space in a datagram. If counters must be sent in order to satisfy the maximum sampling interval then a datagram must be sent containing the outstanding counters.

The following is the XDR description of an sFlow Datagram:

```
/* sFlow Datagram Version 4 */

/* Revision History
   - version 4 adds support BGP communities
   - version 3 adds support for extended_url information
*/

/* sFlow Sample types */
```

```
/* Address Types */

typedef opaque ip_v4[4];
typedef opaque ip_v6[16];

enum address_type {
   IP_V4    = 1,
   IP_V6    = 2
}

union address (address_type type) {
   case IP_V4:
     ip_v4;
   case IP_V6:
     ip_v6;
}

/* Packet header data */

const MAX_HEADER_SIZE = 256;   /* The maximum sampled header size. */

/* The header protocol describes the format of the sampled header */
enum header_protocol {
   ETHERNET-ISO8023     = 1,
   ISO88024-TOKENBUS    = 2,
   ISO88025-TOKENRING   = 3,
   FDDI                 = 4,
   FRAME-RELAY          = 5,
   X25                  = 6,
   PPP                  = 7,
   SMDS                 = 8,
   AAL5                 = 9,
   AAL5-IP              = 10, /* e.g., Cisco AAL5 mux */
   IPv4                 = 11,
   IPv6                 = 12,
   MPLS                 = 13
}

struct sampled_header {
   header_protocol protocol;       /* Format of sampled header */
   unsigned int frame_length;      /* Original length of packet before
                                      sampling */
   opaque header<MAX_HEADER_SIZE>; /* Header bytes */
}

/* Packet IP version 4 data */

struct sampled_ipv4 {
```

```
   unsigned int length;     /* The length of the IP packet excluding
                               lower layer encapsulations */
   unsigned int protocol;   /* IP Protocol type
                               (for example, TCP = 6, UDP = 17) */
   ip_v4 src_ip;            /* Source IP Address */
   ip_v4 dst_ip;            /* Destination IP Address */
   unsigned int src_port;   /* TCP/UDP source port number or
                               equivalent */
   unsigned int dst_port;   /* TCP/UDP destination port number or
                               equivalent */
   unsigned int tcp_flags;  /* TCP flags */
   unsigned int tos;        /* IP type of service */
}
/* Packet IP version 6 data */

struct sampled_ipv6 {
   unsigned int length;     /* The length of the IP packet excluding
                               lower layer encapsulations */
   unsigned int protocol;   /* IP next header
                               (for example, TCP = 6, UDP = 17) */
   ip_v6 src_ip;            /* Source IP Address */
   ip_v6 dst_ip;            /* Destination IP Address */
   unsigned int src_port;   /* TCP/UDP source port number or
                               equivalent */
   unsigned int dst_port;   /* TCP/UDP destination port number or
                               equivalent */
   unsigned int tcp_flags;  /* TCP flags */
   unsigned int priority;   /* IP priority */
}


/* Packet data */

enum packet_information_type {
   HEADER  = 1,      /* Packet headers are sampled */
   IPV4    = 2,      /* IP version 4 data */
   IPV6    = 3       /* IP version 6 data */
}

union packet_data_type (packet_information_type type) {
   case HEADER:
      sampled_header header;
   case IPV4:
      sampled_ipv4 ipv4;
   case IPV6:
      sampled_ipv6 ipv6;
}
```

```
/* Extended data types */

/* Extended switch data */

struct extended_switch {
   unsigned int src_vlan;     /* The 802.1Q VLAN id of incoming frame */
   unsigned int src_priority; /* The 802.1p priority of incoming
                                 frame */
   unsigned int dst_vlan;     /* The 802.1Q VLAN id of outgoing frame */
   unsigned int dst_priority; /* The 802.1p priority of outgoing
                                 frame */
}

/* Extended router data */

struct extended_router {
   address nexthop;         /* IP address of next hop router */
   unsigned int src_mask;   /* Source address prefix mask bits */
   unsigned int dst_mask;   /* Destination address prefix mask bits */
}

/* Extended gateway data */

enum as_path_segment_type {
   AS_SET      = 1,           /* Unordered set of ASs */
   AS_SEQUENCE = 2            /* Ordered set of ASs */
}

union as_path_type (as_path_segment_type) {
   case AS_SET:
      unsigned int as_set<>;
   case AS_SEQUENCE:
      unsigned int as_sequence<>;
}

struct extended_gateway {
   unsigned int as;           /* Autonomous system number of router */
   unsigned int src_as;       /* Autonomous system number of source */
   unsigned int src_peer_as;  /* Autonomous system number of source
                                 peer */
   as_path_type dst_as_path<>; /* Autonomous system path to the
                                  destination */
   unsigned int communities<>; /* Communities associated with this
                                  route */
   unsigned int localpref;    /* LocalPref associated with this
                                 route */
}
```

```
/* Extended user data */

struct extended_user {
   string src_user<>;          /* User ID associated with packet
                                  source */
   string dst_user<>;          /* User ID associated with packet
                                  destination */

}

/* Extended URL data */

enum url_direction {
   src    = 1,                 /* URL is associated with source
                                  address */
   dst    = 2                  /* URL is associated with destination
                                  address */
}

struct extended_url {
   url_direction direction;    /* URL associated with packet source */
   string url<>;               /* URL associated with the packet flow */
}

/* Extended data */
enum extended_information_type {
   SWITCH    = 1,      /* Extended switch information */
   ROUTER    = 2,      /* Extended router information */
   GATEWAY   = 3,      /* Extended gateway router information */
   USER      = 4,      /* Extended TACACS/RADIUS user information */
   URL       = 5       /* Extended URL information */
}

union extended_data_type (extended_information_type type) {
   case SWITCH:
      extended_switch switch;
   case ROUTER:
      extended_router router;
   case GATEWAY:
      extended_gateway gateway;
   case USER:
      extended_user user;
   case URL:
      extended_url url;
}

/* Format of a single flow sample */
```

```
struct flow_sample {
unsigned int sequence_number;    /* Incremented with each flow sample
                                    generated by this source_id */
unsigned int source_id;          /* sFlowDataSource encoded as follows:
                                    The most significant byte of the
                                    source_id is used to indicate the
                                    type of sFlowDataSource
                                    (0 = ifIndex,
                                    1 = smonVlanDataSource,
                                    2 = entPhysicalEntry) and the
                                    lower three bytes contain the
                                    relevant index value.*/

unsigned int sampling_rate;      /* sFlowPacketSamplingRate */
unsigned int sample_pool;        /* Total number of packets that could
                                    have been sampled (i.e., packets
                                    skipped by sampling process + total
                                    number of samples) */
unsigned int drops;              /* Number times a packet was dropped
                                    due to lack of resources */

unsigned int input;               /* SNMP ifIndex of input interface.
                                     0 if interface is not known.  */
unsigned int output;              /* SNMP ifIndex of output interface,
                                    0 if interface is not known.
                                    Set most significant bit to
                                    indicate multiple destination
                                    interfaces (i.e., in case of
                                    broadcast or multicast)
                                    and set lower order bits to
                                    indicate number of destination
                                    interfaces.
                                    Examples:
                                       0x00000002  indicates ifIndex =
                                                   2
                                       0x00000000  ifIndex unknown.
                                       0x80000007  indicates a packet
                                                   sent to 7
                                                   interfaces.
                                       0x80000000  indicates a packet
                                                   sent to an unknown
                                                   number of interfaces
                                                   greater than 1. */

   packet_data_type packet_data;       /* Information about sampled
                                          packet */
   extended_data_type extended_data<>; /* Extended flow information */
}
```

```
/* Counter types */

/* Generic interface counters - see RFC 2233 */

struct if_counters {
   unsigned int ifIndex;
   unsigned int ifType;
   unsigned hyper ifSpeed;
   unsigned int ifDirection;    /* derived from MAU MIB (RFC 2668)
                                   0 = unknown, 1=full-duplex,
                                   2=half-duplex, 3 = in, 4=out */
   unsigned int ifStatus;       /* bit field with the following bits
                                   assigned
                                   bit 0 = ifAdminStatus
                                     (0 = down, 1 = up)
                                   bit 1 = ifOperStatus
                                     (0 = down, 1 = up) */
   unsigned hyper ifInOctets;
   unsigned int ifInUcastPkts;
   unsigned int ifInMulticastPkts;
   unsigned int ifInBroadcastPkts;
   unsigned int ifInDiscards;
   unsigned int ifInErrors;
   unsigned int ifInUnknownProtos;
   unsigned hyper ifOutOctets;
   unsigned int ifOutUcastPkts;
   unsigned int ifOutMulticastPkts;
   unsigned int ifOutBroadcastPkts;
   unsigned int ifOutDiscards;
   unsigned int ifOutErrors;
   unsigned int ifPromiscuousMode;
}

/* Ethernet interface counters - see RFC 2358 */

struct ethernet_counters {
   if_counters generic;
   unsigned int dot3StatsAlignmentErrors;
   unsigned int dot3StatsFCSErrors;
   unsigned int dot3StatsSingleCollisionFrames;
   unsigned int dot3StatsMultipleCollisionFrames;
   unsigned int dot3StatsSQETestErrors;
   unsigned int dot3StatsDeferredTransmissions;
   unsigned int dot3StatsLateCollisions;
   unsigned int dot3StatsExcessiveCollisions;
   unsigned int dot3StatsInternalMacTransmitErrors;
   unsigned int dot3StatsCarrierSenseErrors;
   unsigned int dot3StatsFrameTooLongs;
```

```
   unsigned int dot3StatsInternalMacReceiveErrors;
   unsigned int dot3StatsSymbolErrors;
}

/* FDDI interface counters - see RFC 1512 */
struct fddi_counters {
  if_counters generic;
}

/* Token ring counters - see RFC 1748 */

struct tokenring_counters {
  if_counters generic;
  unsigned int dot5StatsLineErrors;
  unsigned int dot5StatsBurstErrors;
  unsigned int dot5StatsACErrors;
  unsigned int dot5StatsAbortTransErrors;
  unsigned int dot5StatsInternalErrors;
  unsigned int dot5StatsLostFrameErrors;
  unsigned int dot5StatsReceiveCongestions;
  unsigned int dot5StatsFrameCopiedErrors;
  unsigned int dot5StatsTokenErrors;
  unsigned int dot5StatsSoftErrors;
  unsigned int dot5StatsHardErrors;
  unsigned int dot5StatsSignalLoss;
  unsigned int dot5StatsTransmitBeacons;
  unsigned int dot5StatsRecoverys;
  unsigned int dot5StatsLobeWires;
  unsigned int dot5StatsRemoves;
  unsigned int dot5StatsSingles;
  unsigned int dot5StatsFreqErrors;
}

/* 100 BaseVG interface counters - see RFC 2020 */

struct vg_counters {
  if_counters generic;
  unsigned int dot12InHighPriorityFrames;
  unsigned hyper dot12InHighPriorityOctets;
  unsigned int dot12InNormPriorityFrames;
  unsigned hyper dot12InNormPriorityOctets;
  unsigned int dot12InIPMErrors;
  unsigned int dot12InOversizeFrameErrors;
  unsigned int dot12InDataErrors;
  unsigned int dot12InNullAddressedFrames;
  unsigned int dot12OutHighPriorityFrames;
  unsigned hyper dot12OutHighPriorityOctets;
  unsigned int dot12TransitionIntoTrainings;
```

```
  unsigned hyper dot12HCInHighPriorityOctets;
  unsigned hyper dot12HCInNormPriorityOctets;
  unsigned hyper dot12HCOutHighPriorityOctets;
}

/* WAN counters */

struct wan_counters {
  if_counters generic;
}

/* VLAN counters */

struct vlan_counters {
  unsigned int vlan_id;
  unsigned hyper octets;
  unsigned int ucastPkts;
  unsigned int multicastPkts;
  unsigned int broadcastPkts;
  unsigned int discards;
}

/* Counter data */

enum counters_version {
   GENERIC      = 1,
   ETHERNET     = 2,
   TOKENRING    = 3,
   FDDI         = 4,
   VG           = 5,
   WAN          = 6,
   VLAN         = 7
}

union counters_type (counters_version version) {
   case GENERIC:
      if_counters generic;
   case ETHERNET:
      ethernet_counters ethernet;
   case TOKENRING:
      tokenring_counters tokenring;
   case FDDI:
      fddi_counters fddi;
   case VG:
      vg_counters vg;
   case WAN:
      wan_counters wan;
   case VLAN:
```

```
      vlan_counters vlan;
}

/* Format of a single counter sample */

struct counters_sample {
   unsigned int sequence_number;   /* Incremented with each counter
                                      sample generated by this
                                      source_id */
   unsigned int source_id;         /* sFlowDataSource encoded as
                                      follows:
                                       The most significant byte of the
                                       source_id is used to indicate the
                                       type of sFlowDataSource
                                       (0 = ifIndex,
                                       1 = smonVlanDataSource,
                                       2 = entPhysicalEntry) and the
                                           lower three
                                       bytes contain the relevant
                                       index value.*/

   unsigned int sampling_interval; /* sFlowCounterSamplingInterval*/
   counters_type counters;
}

/* Format of a sample datagram */

enum sample_types {
   FLOWSAMPLE  = 1,
   COUNTERSSAMPLE = 2
}

union sample_type (sample_types sampletype) {
   case FLOWSAMPLE:
      flow_sample flowsample;
   case COUNTERSSAMPLE:
      counters_sample counterssample;
}

struct sample_datagram_v4 {
   address agent_address           /* IP address of sampling agent,
                                      sFlowAgentAddress. */
   unsigned int sequence_number;  /* Incremented with each sample
                                     datagram generated */
   unsigned int uptime;           /* Current time (in milliseconds since
                                     device last booted).  Should be set
                                     as close to datagram transmission
                                     time as possible.*/
```

```
   sample_type samples<>;         /* An array of flow, counter and delay
                                     samples */
}

enum datagram_version {
   VERSION4 = 4
}

union sample_datagram_type (datagram_version version) {
   case VERSION4:
      sample_datagram_v4 datagram;
}

struct sample_datagram {
   sample_datagram_type version;
}
```

The sFlow Datagram specification makes use of definitions from a number of existing RFCs [22], [23], [24], [25], [26], [27] and [28].

5. Security Considerations

Deploying a traffic monitoring system raises a number of security related issues. sFlow does not provide specific security mechanisms, relying instead on proper deployment and configuration to maintain an adequate level of security.

While the deployment of traffic monitoring systems does create some risk, it also provides a powerful means of detecting and tracing unauthorized network activity.

This section is intended to provide information that will help understand potential risks and configuration options for mitigating those risks.

5.1 Control

The sFlow MIB is used to configure the generation of sFlow samples. The security of SNMP, with access control lists, is usually considered adequate in an enterprise setting. However, there are situations when these security measures are insufficient (for example a WAN router) and SNMP configuration control will be disabled.

When SNMP is disabled, a command line interface is typically provided. The following arguments are required to configure sFlow sampling on an interface.

```
-sFlowDataSource               <source>
-sFlowPacketSamplingRate       <rate>
-sFlowCounterSamplingInterval  <interval>
-sFlowMaximumHeaderSize        <header size>
-sFlowMaximumDatagramSize      <datagram size>
-sFlowCollectorAddress         <address>
-sFlowCollectorPort            <port>
```

5.2 Transport

Traffic information is sent unencrypted across the network from the sFlow Agent to the sFlow Analyzer and is thus vulnerable to eavesdropping. This risk can be limited by creating a secure measurement network and routing the sFlow Datagrams over this network. The choice of technology for creating the secure measurement network is deployment specific, but could include the use of VLANs or VPN tunnels.

The sFlow Analyzer is vulnerable to attacks involving spoofed sFlow Datagrams. To limit this vulnerability the sFlow Analyzer should check sequence numbers and verify source addresses. If a secure measurement network has been constructed then only sFlow Datagrams received from that network should be processed.

5.3 Confidentiality

Traffic information can reveal confidential information about individual network users. The degree of visibility of application level data can be controlled by limiting the number of header bytes captured by the sFlow agent. In addition, packet sampling makes it virtually impossible to capture sequences of packets from an individual transaction.

The traffic patterns discernible by decoding the sFlow Datagrams in the sFlow Analyzer can reveal details of an individual's network related activities and due care should be taken to secure access to the sFlow Analyzer.

6. References

[1] Sun Microsystems, Inc., "XDR: External Data Representation Standard", RFC 1014, June 1987.

[2] Harrington, D., Presuhn, R., and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks", RFC 2571, April 1999.

[3] Rose, M. and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", STD 16, RFC 1155, May 1990.

[4] Rose, M. and K. McCloghrie, "Concise MIB Definitions", STD 16, RFC 1212, March 1991.

[5] Rose, M., "A Convention for Defining Traps for use with the SNMP", RFC 1215, March 1991.

[6] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

[7] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[8] McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., Rose, M. and S. Waldbusser, "Conformance Statements for SMIv2", STD 58, RFC 2580, April 1999.

[9] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", STD 15, RFC 1157, May 1990.

[10] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[11] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[12] Case, J., Harrington D., Presuhn R. and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2572, April 1999.

[13] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2574, April 1999.

[14] Case, J., McCloghrie, K., Rose, M. and S. Waldbusser, "Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[15] Levi, D., Meyer, P. and B. Stewart, "SNMPv3 Applications", RFC 2573, April 1999.

[16] Wijnen, B., Presuhn, R. and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)", RFC 2575, April 1999.

[17] Case, J., Mundy, R., Partain, D. and B. Stewart, "Introduction to Version 3 of the Internet-standard Network Management Framework", RFC 2570, April 1999.

[18] Waldbusser, S., "Remote Network Monitoring Management Information Base", RFC 2819, May 2000.

[19] Waterman, R., Lahaye, B., Romascanu, D. and S. Waldbusser, "Remote Network Monitoring MIB Extensions for Switched Networks Version 1.0", RFC 2613, June 1999.

[20] Daniele, M., Haberman, B., Routhier, S. and J. Schoenwaelder, "Textual Conventions for Internet Network Addresses", RFC 2851, June 2000.

[21] Brownlee, N., "Traffic Flow Measurement: Meter MIB", RFC 2720, October 1999.

[22] Smith, A., Flick, J., de Graaf, K., Romanscanu, D., McMaster, D., McCloghrie, K. and S. Roberts, "Definition of Managed Objects for IEEE 802.3 Medium Attachment Units (MAUs)", RFC 2668, August 1999.

[23] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB using SMIv2", RFC 2233, November 1997.

[24] Flick, J. and J. Johnson, "Definition of Managed Objects for the Ethernet-like Interface Types", RFC 2358, June 1998.

[25] Case, J., "FDDI Management Information Base", RFC 1512, September 1993.

[26] McCloghrie, K. and E. Decker, "IEEE 802.5 MIB using SMIv2", RFC 1748, December 1994.

[27] Flick, J., "Definitions of Managed Objects for IEEE 802.12 Interfaces", RFC 2020, October 1996.

[28] Willis, S., Burruss, J. and J. Chu, "Definitions of Managed Objects for the Fourth Version of the Border Gateway Protocol (BGP-4) using SMIv2", RFC 1657, July 1994.

7. Authors' Addresses

Peter Phaal
InMon Corporation
1404 Irving Street
San Francisco, CA 94122

Phone: (415) 661-6343
EMail: peter_phaal@INMON.COM

Sonia Panchen
InMon Corporation
1404 Irving Street
San Francisco, CA 94122

Phone: (415) 661-6343
EMail: sonia_panchen@INMON.COM

Neil McKee
InMon Corporation
1404 Irving Street
San Francisco, CA 94122

Phone: (415) 661-6343
EMail: neil_mckee@INMON.COM

8. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

9. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.