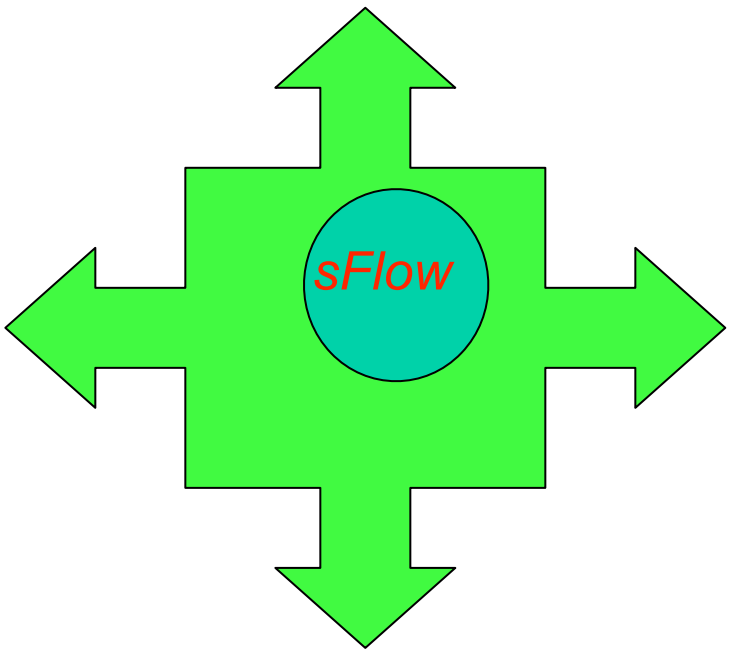
sFlow (http://www.sflow.org)

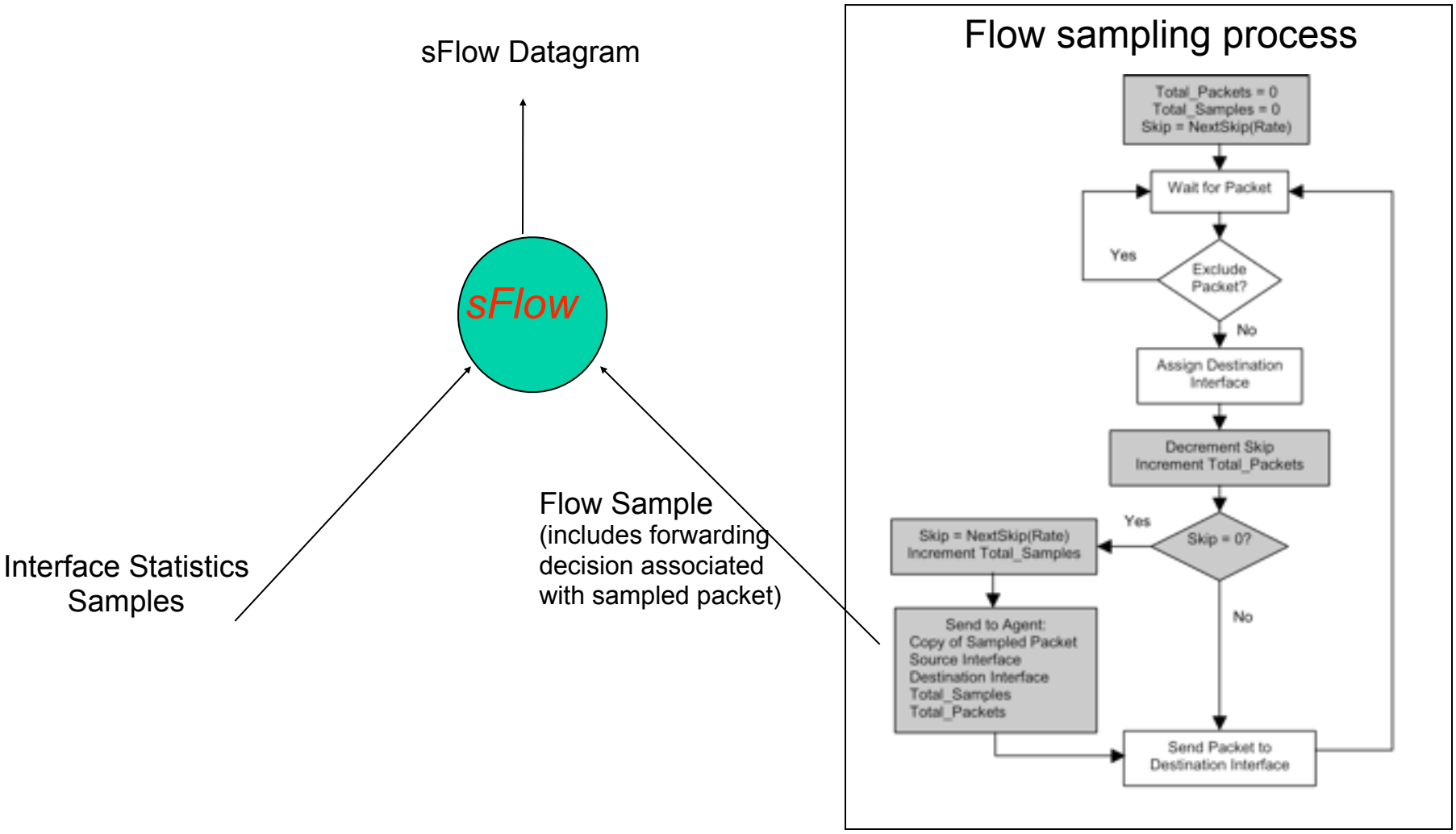


Agent Software Description

This slide set is intended as a guide to InMon's example sFlow agent software. The concepts and design choices are illustrated. The intention is to make it easier to read and understand the source code, so that the process of developing and integrating an sFlow agent into existing switch/router firmware is as rapid and painless as possible.

The language is plain C, but the design is essentially object-oriented. The idea was to make the basic sFlow-encoding engine so simple to reuse that no one feels the need to rewrite it. This will help to ensure interoperability.

sFlow Packet Sampling Algorithm



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

Sampling flows: 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. On average 1 in N packets is sampled. The sFlow Agent also tracks total number of packets switched. This total count is used during sample analysis to scale the results. This basic sampling algorithm is simple enough to implement in hardware.

The sFlow Agent combines the source and destination interfaces for the flow, the sampled packet header, original packet length, the total number of packets, the forwarding decision associated with the sampled packet, and other key information from the packet into an sFlow datagram. The sFlow datagram is then sent over the network to a remote machine, the sFlow Collector/Analyzer, for analysis. Handing analysis off to the sFlow Collector/Analyzer minimizes the impact on the CPU and keeps traffic monitoring costs low.

Sampling of interface counters: The objective of the interface counter sampling is to provide an efficient and scalable way to report counters to a central collector. 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 interface counter sampling are designed to be 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 Collector/Analyzer, counter samples may be taken opportunistically in order to fill these datagrams.

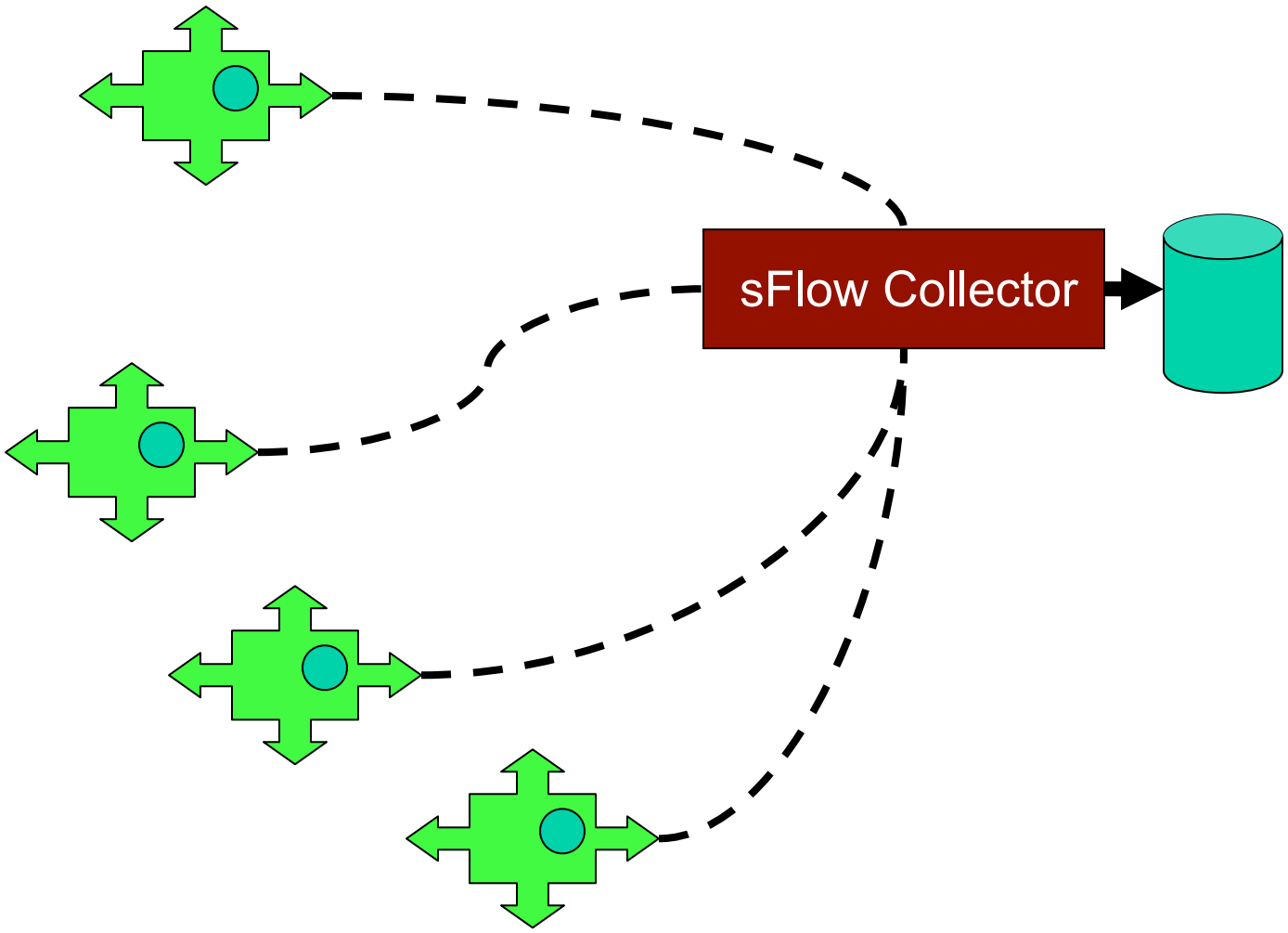
sFlow Datagram

- Interface statistics samples
- Flow sample
  - Packet header (MAC,IP,IPX,AppleTalk,HTTP,FTP,DNS…)
  - Sample process parameters (rate, pool etc.)
  - Switch
    - Input/output ports
    - Priority
    - VLAN
  - Router
    - Source/destination prefix
    - Next hop address
  - Gateway
    - Source AS, Source Peer AS
    - Destination AS Path
    - Communities, local preference
  - User
    - User IDs (TACACS/RADIUS) for source/destination
  - URL
    - URL associated with source/destination
  - NAT
    - Network address translation mapping
  - MPLS
    - Label stack information
  - Vendor Specific

- Raw data export
  - Decode at collector
- Rich detail
  - Traffic
  - Forwarding
  - I/F counters
- Extensible (sFlow v5)
  - Vendor-specific records

The sFlow datagram format specifies the standard data and format of the sampled data. Samples are sent as UDP packets to the host and port specified in the SFLOW MIB (or configured via the CLI).

System Diagram - Network



The sFlow Collector/Analyzer can receive sFlow from dozens, even hundreds of switches. It is then in a position to track interface counters, top talkers and traffic matrices for every interface in the network, all synchronized to the same clock. This allows interface traffic to be compared over the same time interval (e.g. over the same minute). It also allows the data to be combined into a site-wide view, or consolidated into end-to-end traffic matrices.

Key Points:

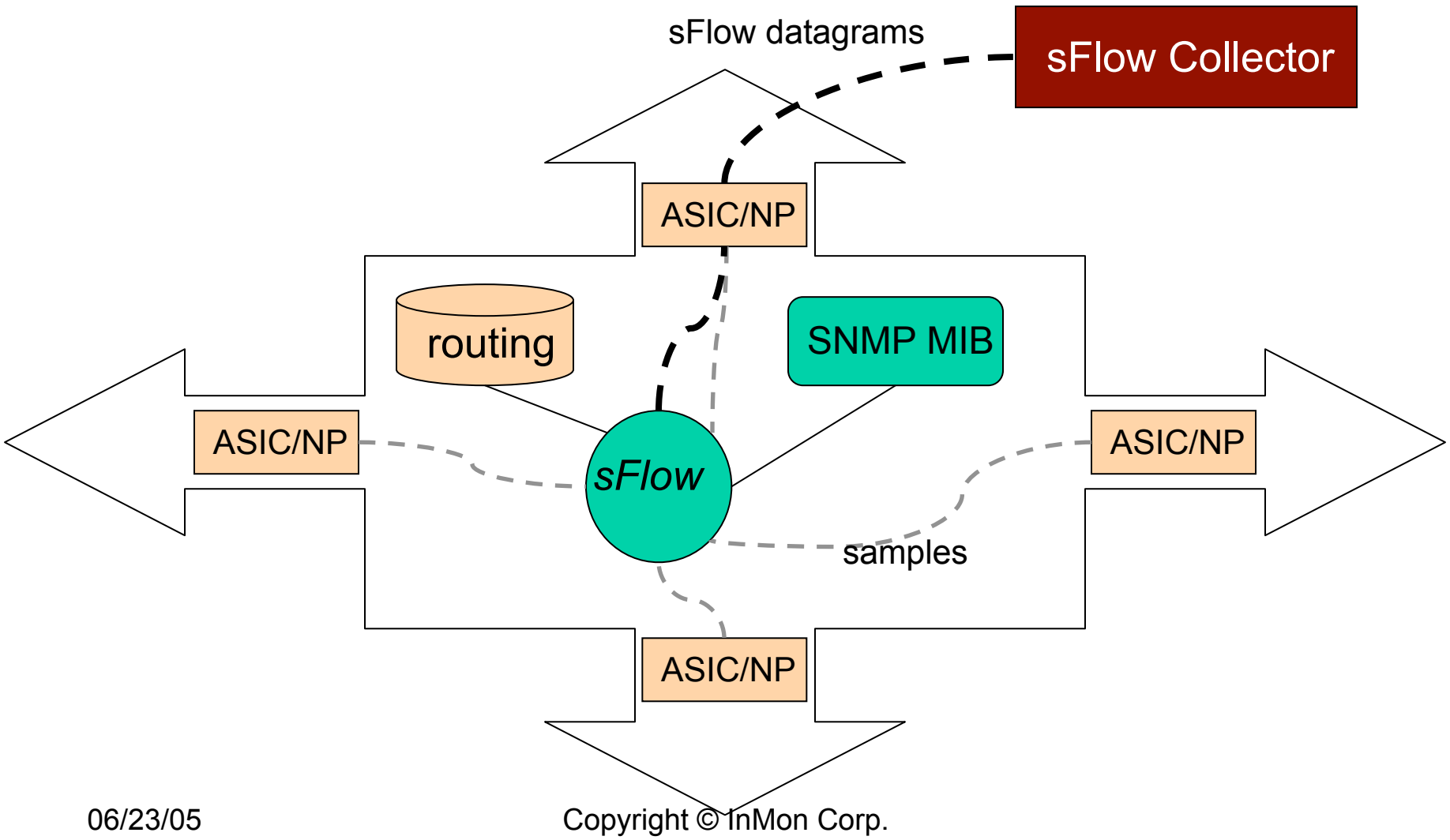
Raw data – processing/further data reduction delayed until analysis time

The sampled data processing (eg aggregation) is a (logically and physically) separate process.

Delaying data processing, means that there is sufficient detail available for a variety of different applications, rather than processing the data for a specific use before export. New types of analysis can be performed without upgrading the agent. Application changes are much easier to make and deploy. Network managers can perform ad-hoc analysis in minutes using server side scripting - for example making the packet headers available means that pattern matching can be performed on the header, such as Code Red detection in the http header.

Richness of data available – including forwarding information from the switch/router that can be used to determine network forwarding paths without tracing specific packets.

System Diagram - Switch



This slide illustrates what happens inside the switch. Each interface or module has an ASIC or Network Processor which performs the packet sampling function. The packet samples are forwarded to the central CPU where the sFlow agent is running. The sFlow agent encodes those samples in UDP datagrams, along with routing information such as subnets, next-hop and as-paths, and sends them over the network to the sFlow collector.

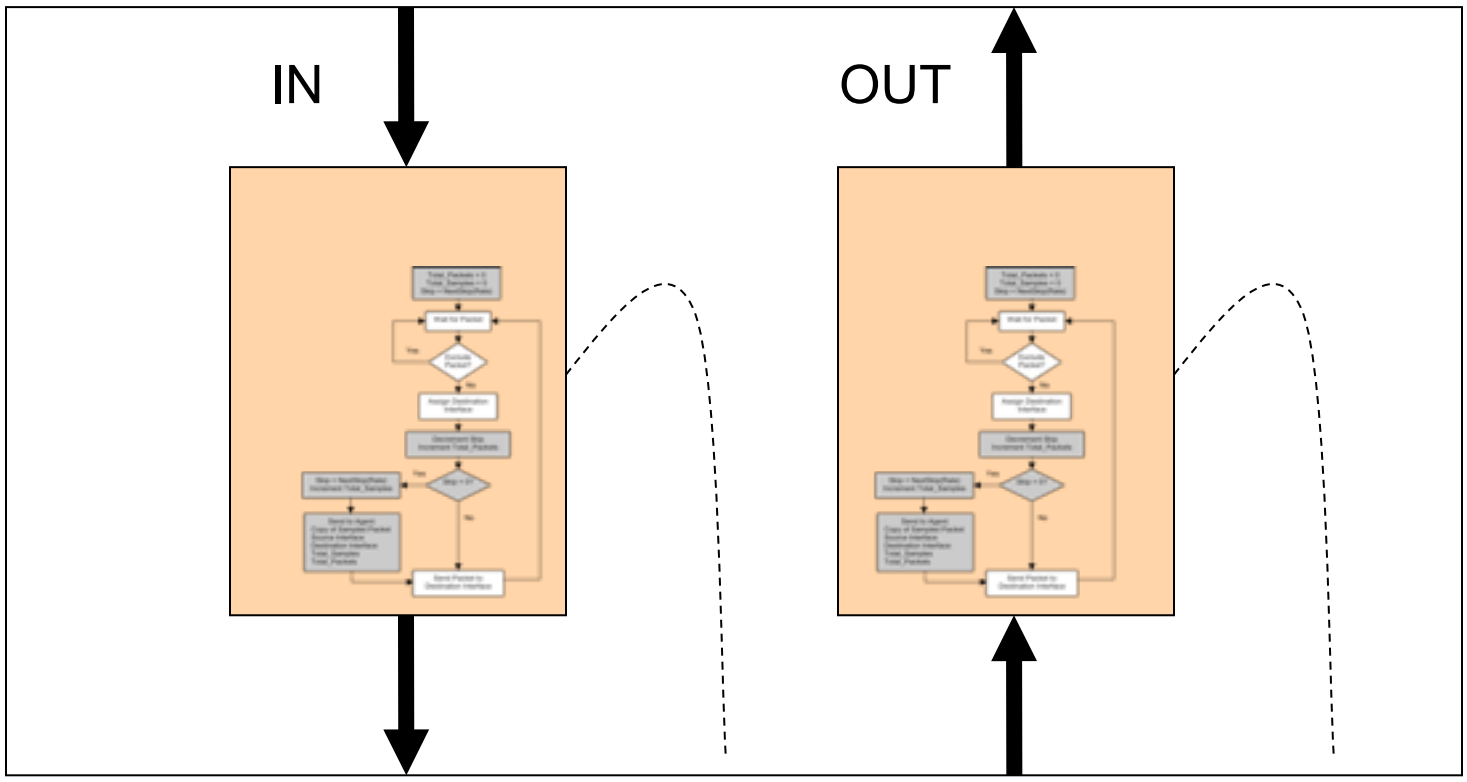
There is an SNMP MIB that controls the sampling and forwarding settings. It is also possible to control sFlow via the command-line interface for the switch.

The advantages of this kind of instrumentation are described in white papers accessible on http://www.sflow.org and at http://www.inmon.com/technology.htm. The main points are:

1. Possible to monitor all ports of the switch continuously, with no impact on the distributed switching performance.
2. Very little memory/CPU required. Samples are not aggregated into a flow-table on the switch, they are forwarded immediately over the network to the sFlow collector.
3. System is tolerant to packet loss in the network (statistical model means loss is equivalent to slight change in sampling rate).
4. sFlow collector can receive data from multiple switches, providing a real-time synchronized view of the whole network.
5. Collector can analyze traffic patterns for whatever protocols are found in the headers (e.g. TCP/IP, IPX, Ethernet, AppleTalk…) No need for layer 2 switch to decode and understand all protocols.

Choice of Sampling Points: Inbound/Outbound

ASIC/Network Processor



The decision on whether to sample outbound packets as well as inbound packets involves several trade-offs.

Advantages of inbound + outbound sampling:

- Can enable on just one link, and see all traffic in and out of that link (useful if only want to monitor WAN link, for example).
- Can see exactly what traffic is on link, including broadcasts and multicasts that came in from other ports on the switch.

Disadvantages of inbound + outbound sampling:

- When enabled on all ports, means switch is working twice as hard and generating twice as many samples as is strictly necessary.
- Broadcasts and multicasts can potentially generate many samples – sometimes even from just one input packet.

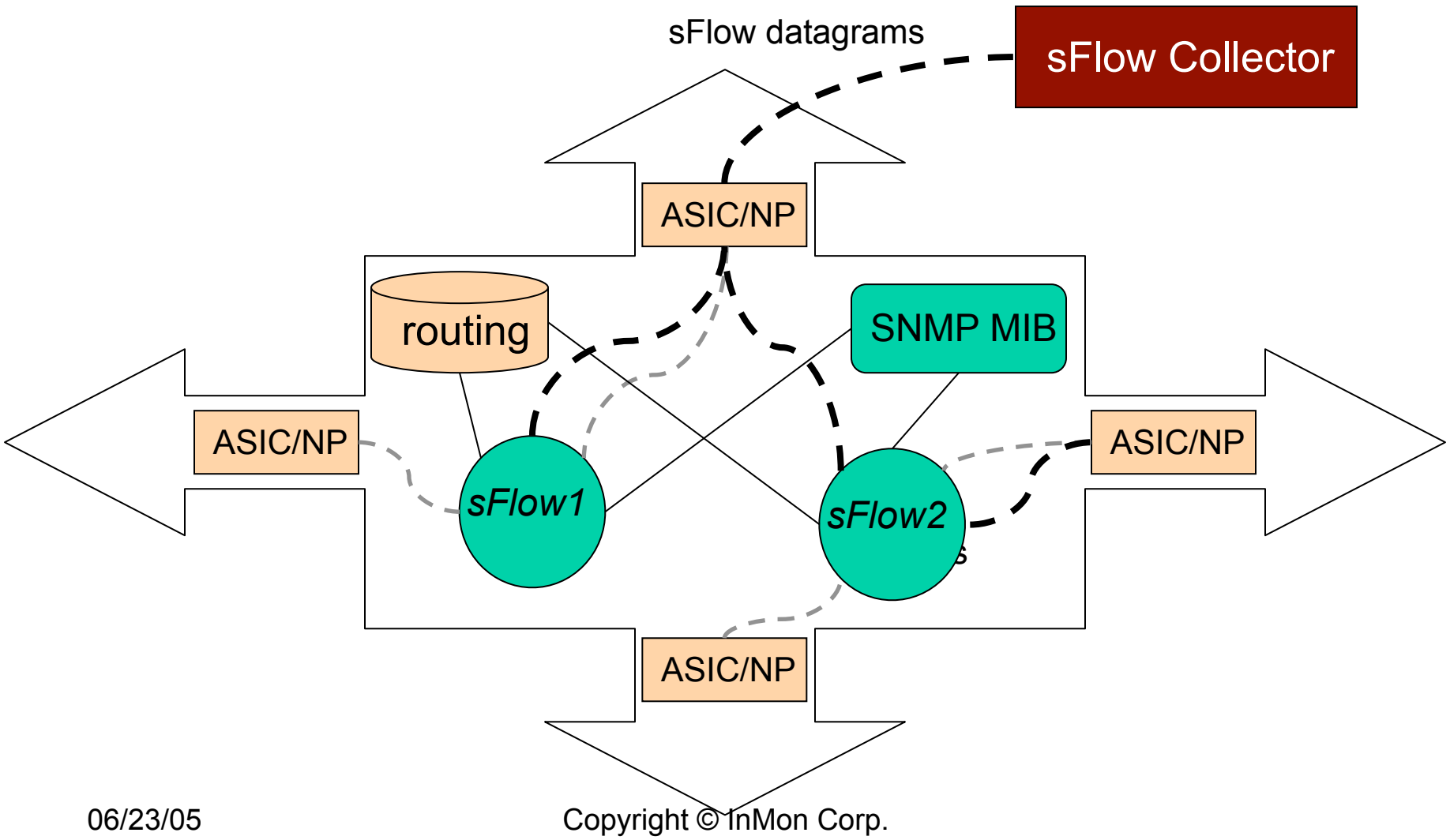
Choice of Sampling Points: Backplane or Distributed

In the simplest implementation of the sFlow agent, there is only one sampling point which represents the whole switch backplane (encoded as sFlowDataSource=ifIndex.0). This has the advantage of simplicity and performance. However, there are times when users want to sample selectively on just one interface.

Supporting different sampling rates on different interfaces is important because one switch may have interfaces with vastly different speeds.

If there is only one sampling point for a set of interfaces, then sub-sampling can be done in software to match the configured sampling rate for each interface. This implies that hardware sampling-rates should be available in powers of 2.

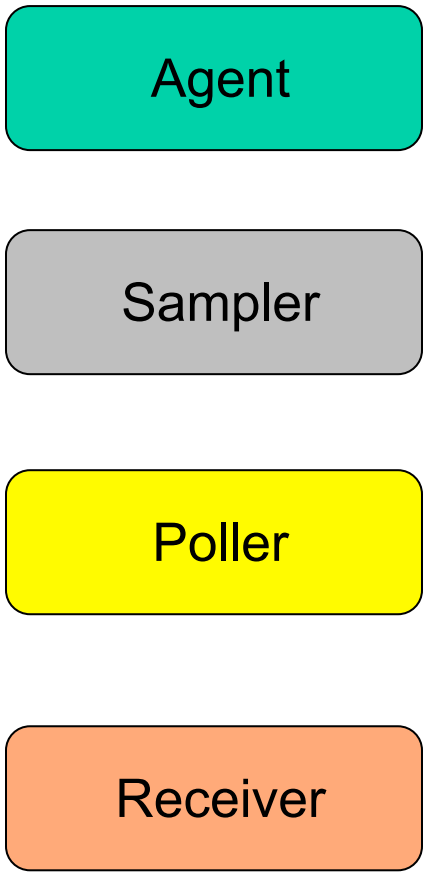
System Diagram – Sub-Agents



In sFlow version 5 there is an additional field in the datagram header: sub-agent id. This allows multiple, independent sFlow agent processes to run on a switch/router. Each sub-agent must have a unique sub-id (which is carried as a field in the datagram header). The only restriction is that the flow data from a given data-source must be reported through one and only one sub-agent.

Depending on the switch architecture, it may be advantageous to distribute the sflow data collection using this mechanism. Sub-agents could run in parallel on different processors. Each will assemble it's own datagrams to send to the collector. No communication is necessary between sub-agents, although they are all under the control of the same SNMP MIB.

Object Classes



One Agent to represent whole switch (or Part of switch if agent subId is used).

One Sampler for each interface (typically). Collects packet samples.

One Poller for each interface (typically). Collects counter samples.

Represents the remote collector {destination IP address + port}
- encodes samples into UDP datagrams.

Agent

There is only one Agent object created (typically this will be instantiated during firmware initialization). The Agent represents the whole switch/router. All interaction between your code and the sFlow agent is channeled through this object. If you have a distributed architecture and multiple CPUs, then you may assign agent sub-ids so that multiple agents can run independently and still represent different parts of the same switch.

Sampler

An Agent can be told to create a number of Sampler objects. Typically you will create one for each interface, although it is also possible to create Samplers to represent a VLAN, a module, or even one to represent the whole switch backplane. Each will have a unique data-source-id which describes what it represents. Packet samples are passed in to the appropriate sampler (or samplers).

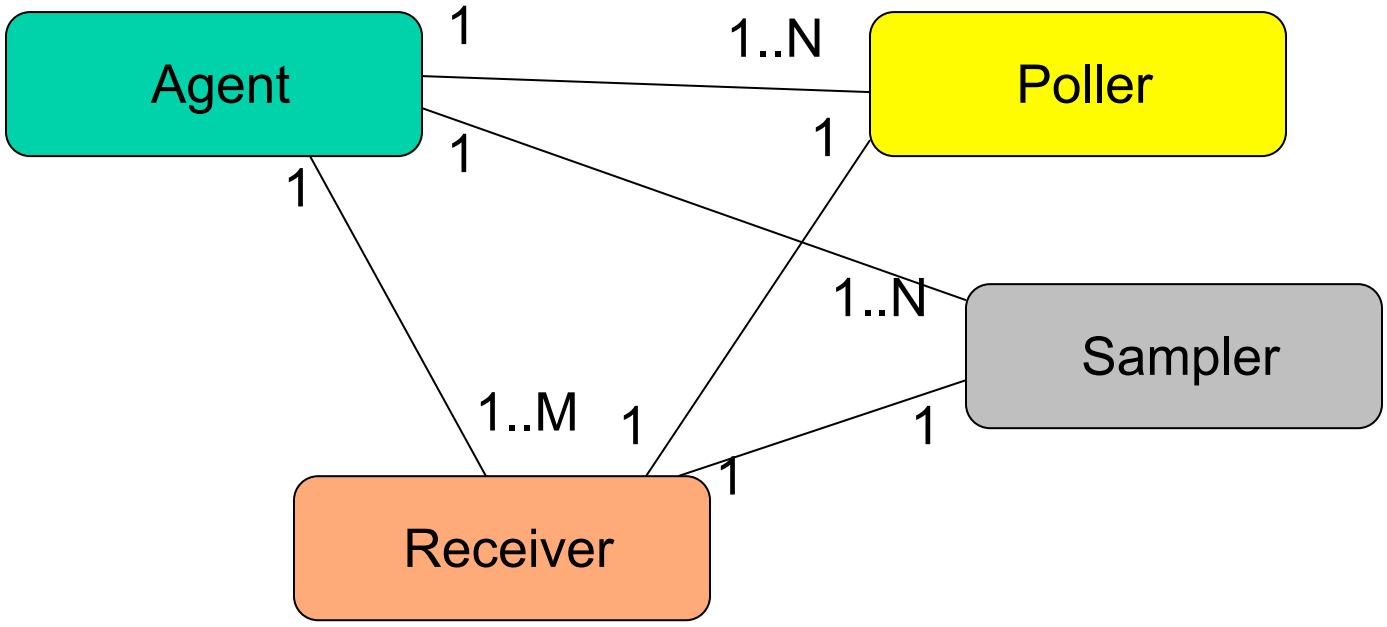
Poller

A Poller is similar to a sampler, except that it collects time-sampled counters. Typically there is a separate Poller for each interface.

Receiver

A Receiver object encodes the flow and counter samples into UDP datagrams, to be send to the destination host/port.

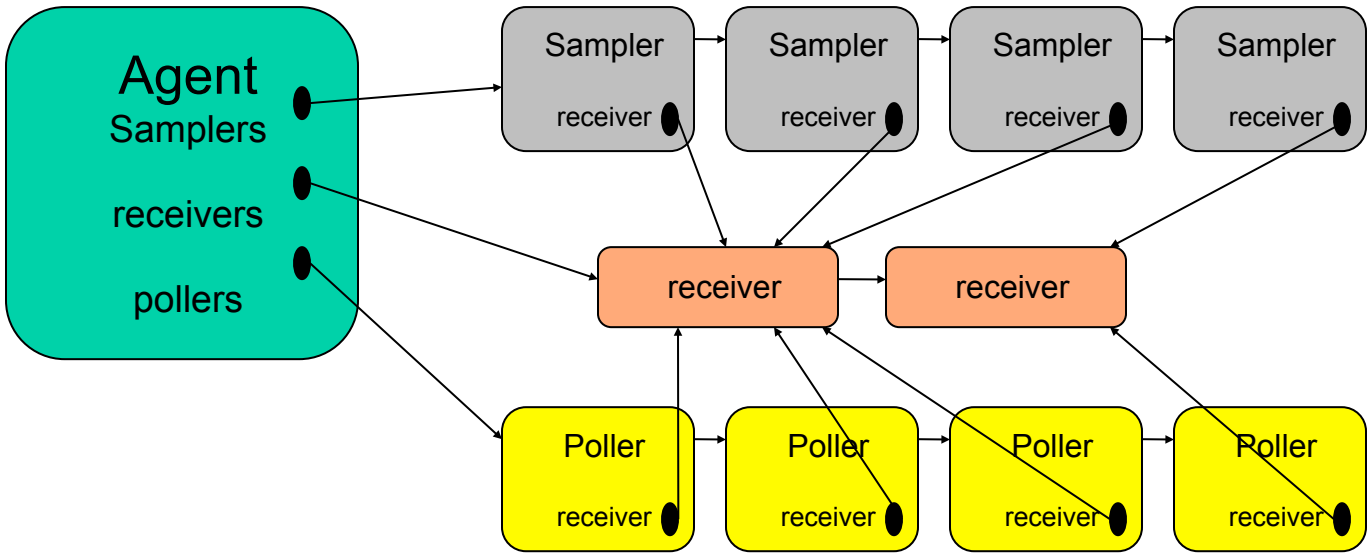
Entity-Relationship



- One Agent - represents switch/router.
- A Sampler and Poller for each Interface (typically).
- One Receiver for each unique {destination IP address + port}

This ER diagram shows the three “classes” in the agent code, and the cardinality of their relationships. One Agent can hold multiple Samplers and multiple Receivers, but each Sampler and Poller points to only one Receiver.

Data Structures



Agent maintains linked lists of Samplers, Receivers and Pollers. Each Sampler or Poller is given a pointer to his Receiver.

This diagram illustrates how the object instances of Agent, Sampler, Poller and Receiver are linked together in memory when the agent is running. The Agent maintains linked-lists of Samplers, Pollers and Receivers.

Not shown here is the Agent's "jump-table" that allows a Sampler to be located efficiently by its ifIndex (implemented as an array lookup). The jump table is automatically created the first time it is used. If there is any change to the list of Samplers, then the jump table is removed (so that it must be rebuilt again next time). The jump table is important because a switch with, say, 600 interfaces would otherwise waste a lot of CPU cycles in searching along the linked list to find the Sampler for a given flow-sample (or when processing an SNMP GET request).

SNMP MIB

sFlowAgent
- sFlowVersion
- sFlowAgentAddressType
- sFlowAgentAddress

sFlowFsTable
- sFlowFsEntry
  - sFlowFsDataSource
  - sFlowFsInstance
  - sFlowFsReceiver
  - sFlowFsPacketSamplingRate
  - sFlowFsMaximumHeaderSize

sFlowRcvrTable
- sFlowRcvrEntry
  - sFlowRcvrIndex
  - sFlowRcvrOwner
  - sFlowRcvrTimeout
  - sFlowRcvrMaximumDatagramSize
  - sFlowRcvrAddressType
  - sFlowRcvrAddress
  - sFlowRcvrPort
  - sFlowRcvrDatagramVersion

sFlowCpTable
- sFlowCpEntry
  - sFlowCpDataSource
  - sFlowCpInstance
  - sFlowCpReceiver
  - sFlowCpInterval

This slide summarizes the structure of the SNMP MIB.

There are three tables in the MIB: sFlowRcvrTable, sFlowFsTable and sFlowCpTable. Each Receiver object is represented by one row in the sFlowRcvrTable. Each Sampler object is represented by one row in the SFlowFsTable and each Poller object is represented by one row in the sFlowCpTable.

The sFlowDataSource identifiers are encoded specially in the C implementation. One 64-bit integer is used to represent ds_class ds_index and ds_instance. In the MIB representation, the ds_class is actually an OBJECT-IDENTIFIER. For example, ds_class=0 is represented in the MIB as the OBJECT-IDENTIFIER ifIndex, which is 1.3.6.1.2.1.2.2.1.1. Thus interface 55 with have the sFlowDataSource = 1.3.6.1.2.1.2.2.1.1.55, but in the Sampler object this will be encoded with the integers ds_class=0, ds_index=55.

Flow Samples

Packet sample
from switch fabric

- Call sfl_agent_getSampler() or sfl_agent_getSamplerByIfIndex() to get appropriate Sampler object for this sample.
- Fill in “form” (SFLFlow_Sample structure) with Sampled header + information from forwarding tables and submit to sampler with sfl_sampler_writeFlowSample().
- Sampler accumulates samplePool and sends sample to receiver to be encoded into next output packet.

When the sampling-rate is set, either through an SNMP SET operation or through the switch/router command line interface, then that sampling rate mean should be passed to the switch hardware. As packets pass through the switch, random samples will be taken (with mean skipcount=sampling-rate) and these will appear for treatment in the software (perhaps marked with a flag to indicate that they were samples. To process each sample that is taken, simply fill in an SFLFlowSample “form” and submit it to the appropriate Sampler object. The SFLFlowSample structure includes a place for the raw sampled header, as well as optional fields where forwarding-decision information can be filled in (such as source and destination ports, VLANs and subnets).

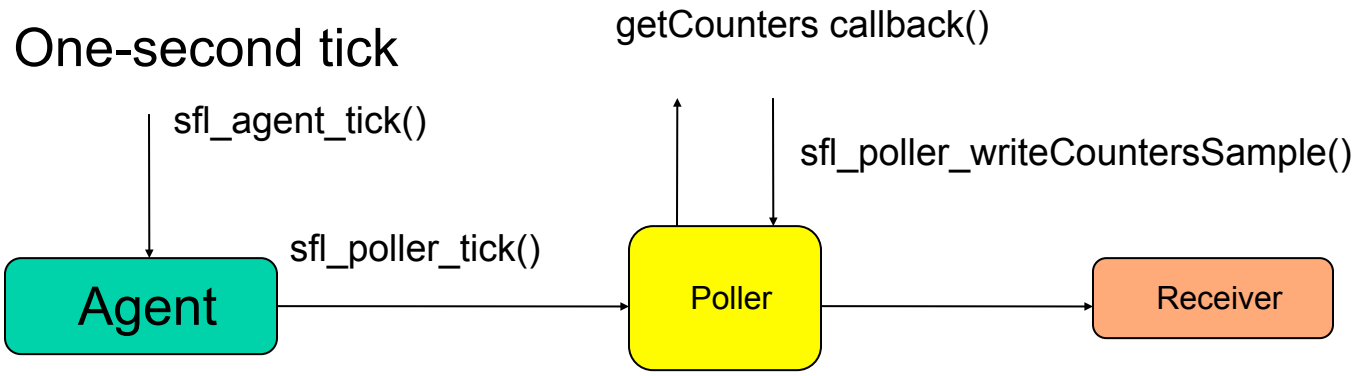
When this form is given to the Sampler by calling sfl_sampler_writeFlowSample() the Sampler object will update its internal counters and then pass the SFLFlowSample to it’s receiver to be encoded into the next datagram that is going out.

So the typical sequence (for input-only sampling by interface) is:

1. Receive a sample from the hardware.
2. Look at the input interface, and use that number to look up the Sampler with sfl_agent_getSamplerByIfIndex().
3. Fill in an instance of SFLFlowSample with the packet header and the forwarding information.
4. Submit it to the Sampler with sfl_sampler_writeFlowSample().

Note that in some configurations the same packet sample may be given to more than one Sampler. This is OK. The sFlow collector/analyzer should be able to differentiate by data-source-id and avoid double-counting.

Counter Samples



Agent dispatches tick to each Sampler.

Sampler counts down to next counter poll. When reaches zero, calls the callback fn to get the counters.

Fill in SFLCountersSample “form” and submit via sfl_poller_writeCountersSample().

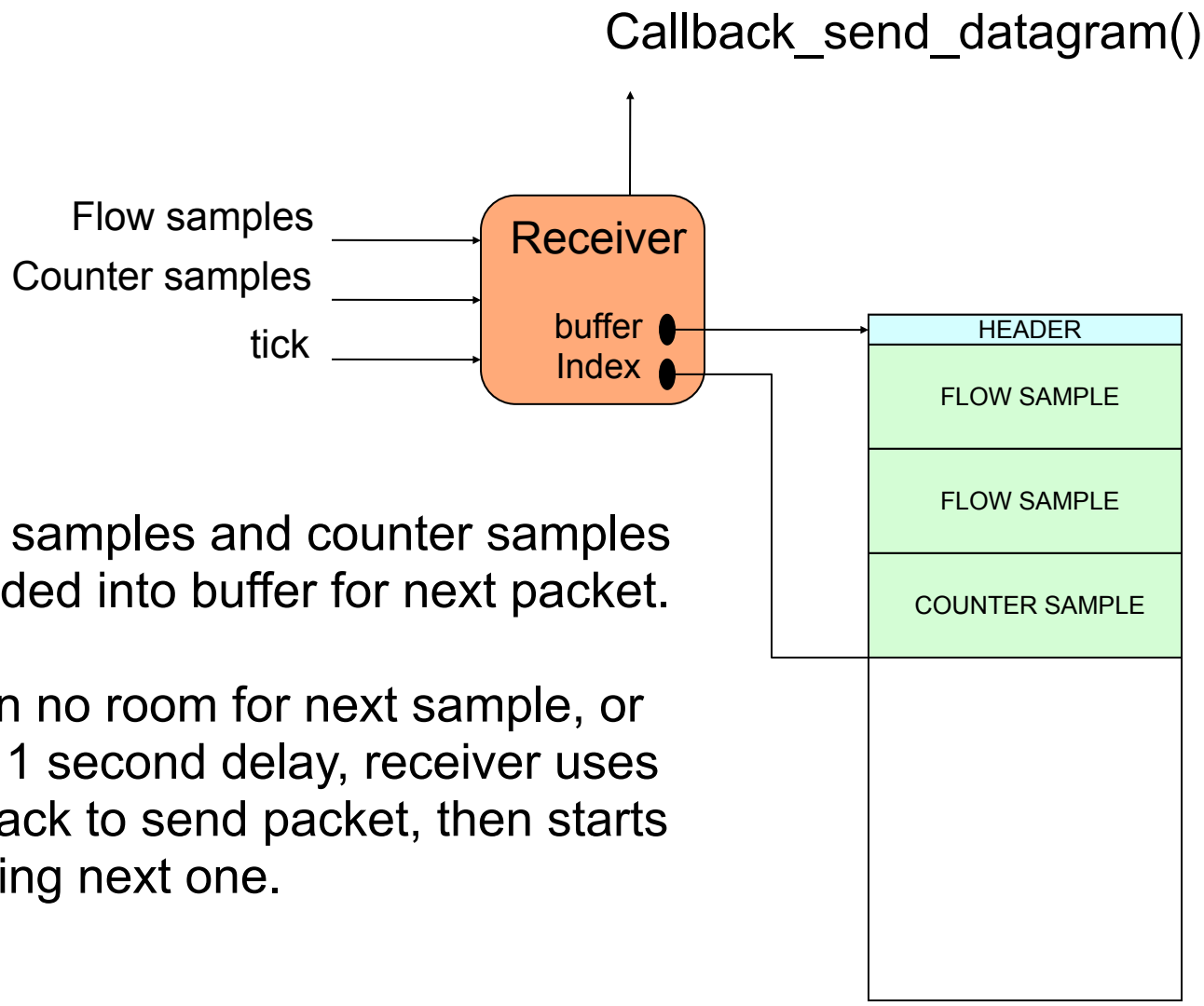
Counter data is sent to the receiver to be encoded into the next datagram.

The sfl_agent_tick() function should be called approximately every second. This should not be on a timer-interrupt, but should be at the same priority as the thread that processes samples from the hardware.

When a Sampler is given a new counter polling-interval, the first thing it does is use a random number to start the countdown to the next counter sample. This ensures that even if all the interfaces are to be polled every 30 seconds, this won’t all happen in a burst of activity on one tick.

The sfl_agent_tick() function will distribute the tick to all the Samplers, who will use it to decrement the countdown to the next counter sample. If a Sampler reaches zero with this countdown, it will call the getCountersCallback function (which you supplied to the Agent when it was created). In this callback function you are expected to create an SFLCountersSample “form” so that you can fill it in with the requested counter snapshot and submit it with sfl_poller_writeCountersSample().

Datagram Encoding



Flow samples and counter samples encoded into buffer for next packet.

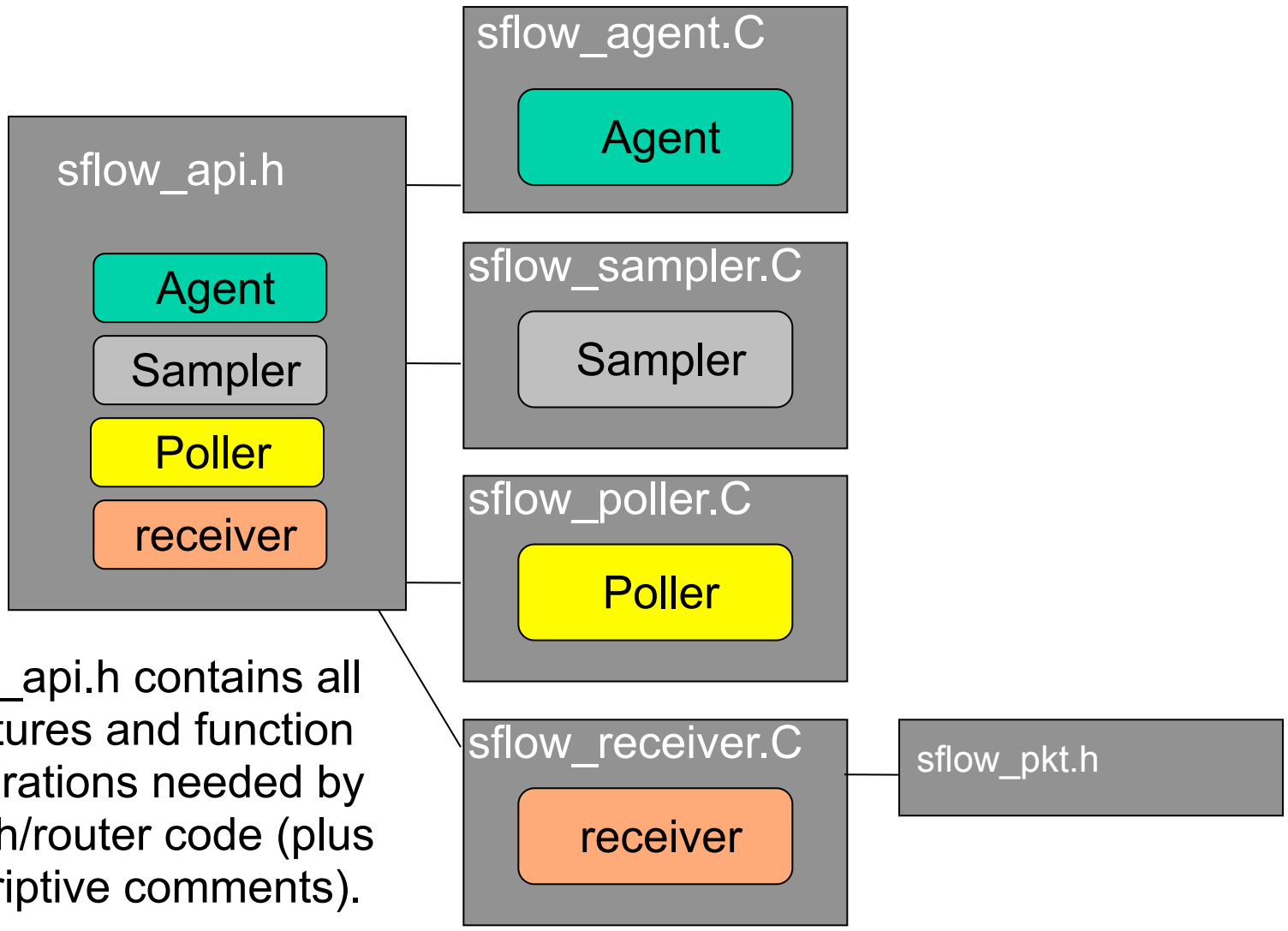
When no room for next sample, or after 1 second delay, receiver uses callback to send packet, then starts building next one.

The sFlow datagram consists of a header followed by one or more FLOW or COUNTER samples. The Receiver object understands how to do the XDR encoding, and pack the data into a contiguous buffer, so that it can be sent out in the next datagram.

If the buffer is full and there is no room for the sample to go into it, then it is sent out (using the callback that you supplied to the Agent at initialization time), and then reset. The sample then becomes the first sample in the next datagram.

If the 1-second tick comes around and there are samples in the buffer, then the datagram is sent and the buffer is reset. This is a simple way to ensure that samples are never delayed for more than 1 second in the agent.

Source Code Files



sflow_api.h contains all structures and function declarations needed by switch/router code (plus descriptive comments).

This slide illustrates how the software is arranged in the source code files.

sflow_api.h contains all the structures and functions that are needed to use the agent.
sflow_pkt.h contains the structures for the encoding of flow-samples and counter-samples.
sflow_agent.C contains the C code for the Agent functions.
sflow_sampler.C contains the C code for the Sampler functions.
sflow_poller.C contains the C code for the Poller functions.
sflow_receiver.C contains the C code for the receiver functions.

More Information

http://www.sflow.org

- Agent Source Code
- sFlow v5 specifications
- sFlow MIB (ASN.1)
- sFlow Datagram Format (XDR)
- Testing tools
- Discussion mailing list

Email: neil_mckee@inmon.com