sFlow - What's New?

Internet2

Neil McKee 4/13/2021

sFlow = streamed counters + random packet samples

sFlow = streamed counters (standard data-plane model across all vendors)
+ random packet samples (plus forwarding and performance details)
+ packet drop headers (with standard reason codes)

From all routers, switches and servers.
Tolerant of packet loss.

Standard counters

Network I/F

- Generic (19)
  - status
  - speed
  - frames
  - bytes
  - discards
  - errors
  - ...
- Ethernet (13)
  - symbol_errors
  - FCS_errors
  - ...
- LAG(13)
  - actorMAC
  - ...
- Optical (14)
  - tx/rx power
  - ...

Host

- CPU (18)
- GPU (10)
- memory (8)
- disk I/O (9)
- Network I/O (8)
- IP (19)
- TCP (15)
- UDP (7)
- ICMP (25)
- VM/Container (28)
- Java VM (22)

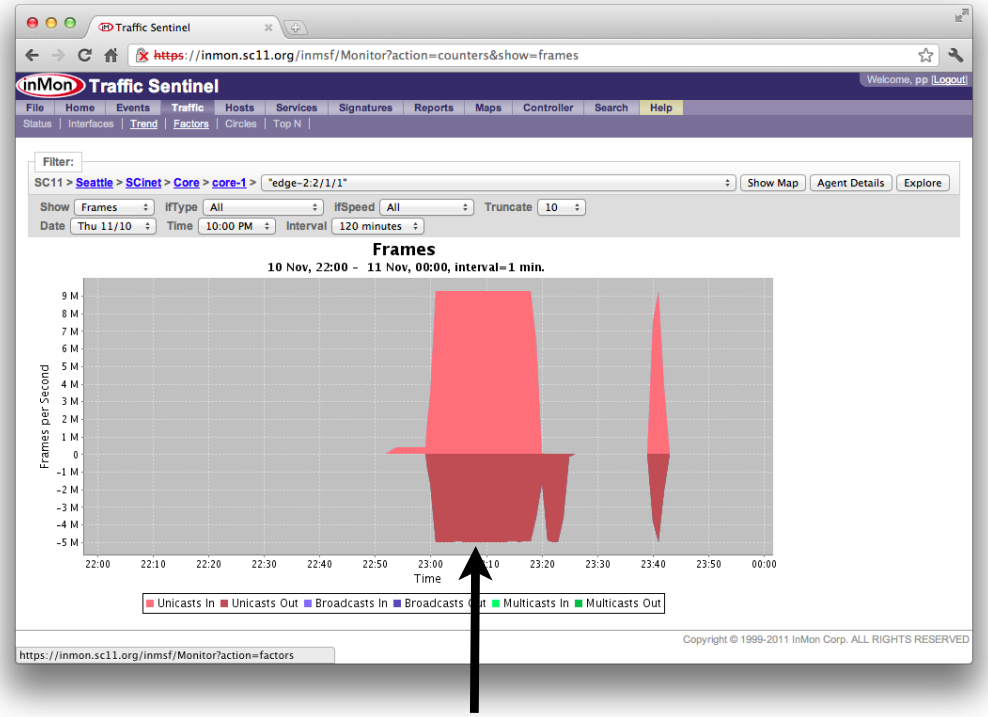
Application

- Generic (25)
- HTTP (15)
- Memcache (31)

Standard counters from every server delivered every N seconds. Plus Standard counters for every VM + Container + JVM.

Standard counters from every physical interface of every switch delivered every N seconds (e.g. N==20). Desynchronized. 200K+ interfaces sending to one collector is fine.

Counters are not enough

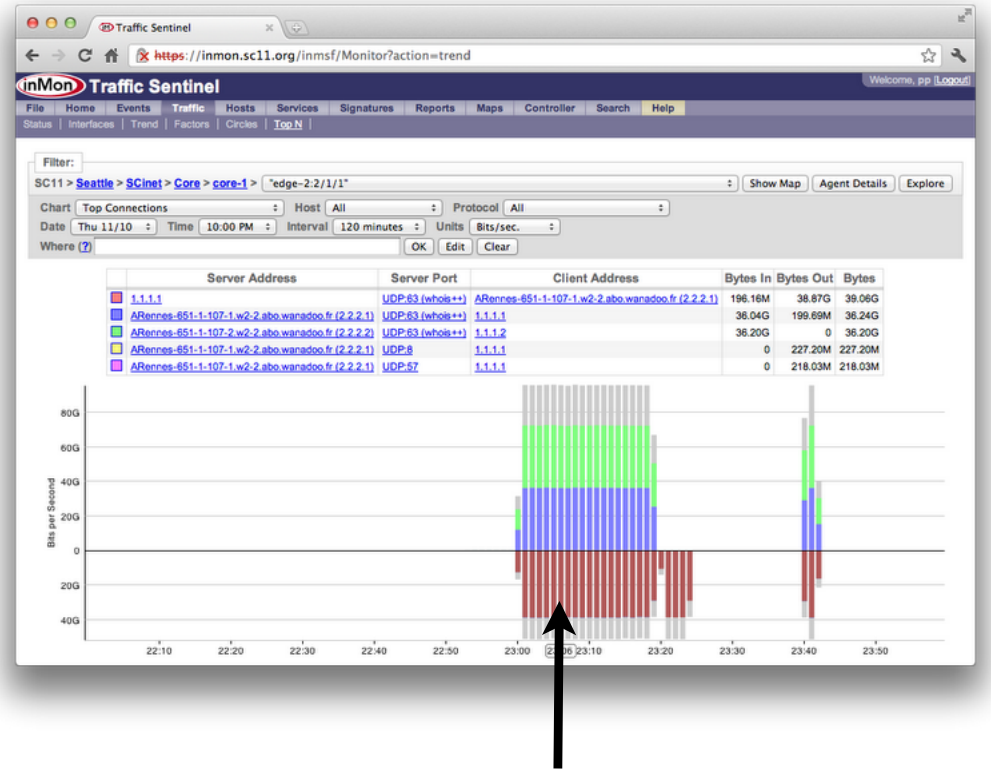


Why the spike in traffic?

(100Gbit link carrying 14,000,000 packets/second)

- Counters tell you there is a problem, but not why.
- Counters summarize performance by dropping high cardinality attributes:
  - IP addresses
  - URLs
  - Memcache keys
- Need to be able to efficiently disaggregate counter by attributes in order to understand root cause of performance problems.
- How do you get this data when there are millions of transactions per second?

sFlow also exports random samples

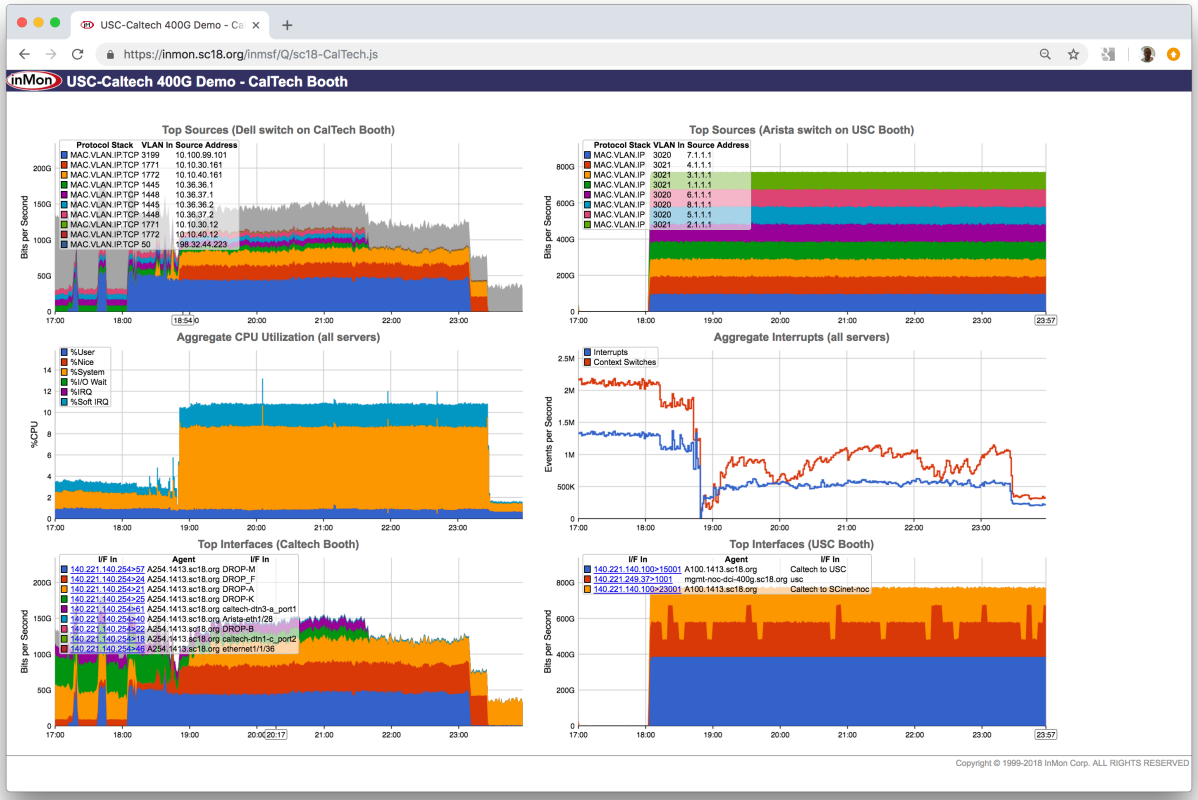


- Random sampling is lightweight
- Critical path roughly cost of maintaining one counter:
  `if(--skip == 0) sample();`
- Sampling is easy to distribute among modules, threads, processes without any synchronization
- Minimal resources required to capture attributes of sampled transactions
- Easily identify top sources, connections, clients, servers, URLs etc.
- Unbiased results with known accuracy

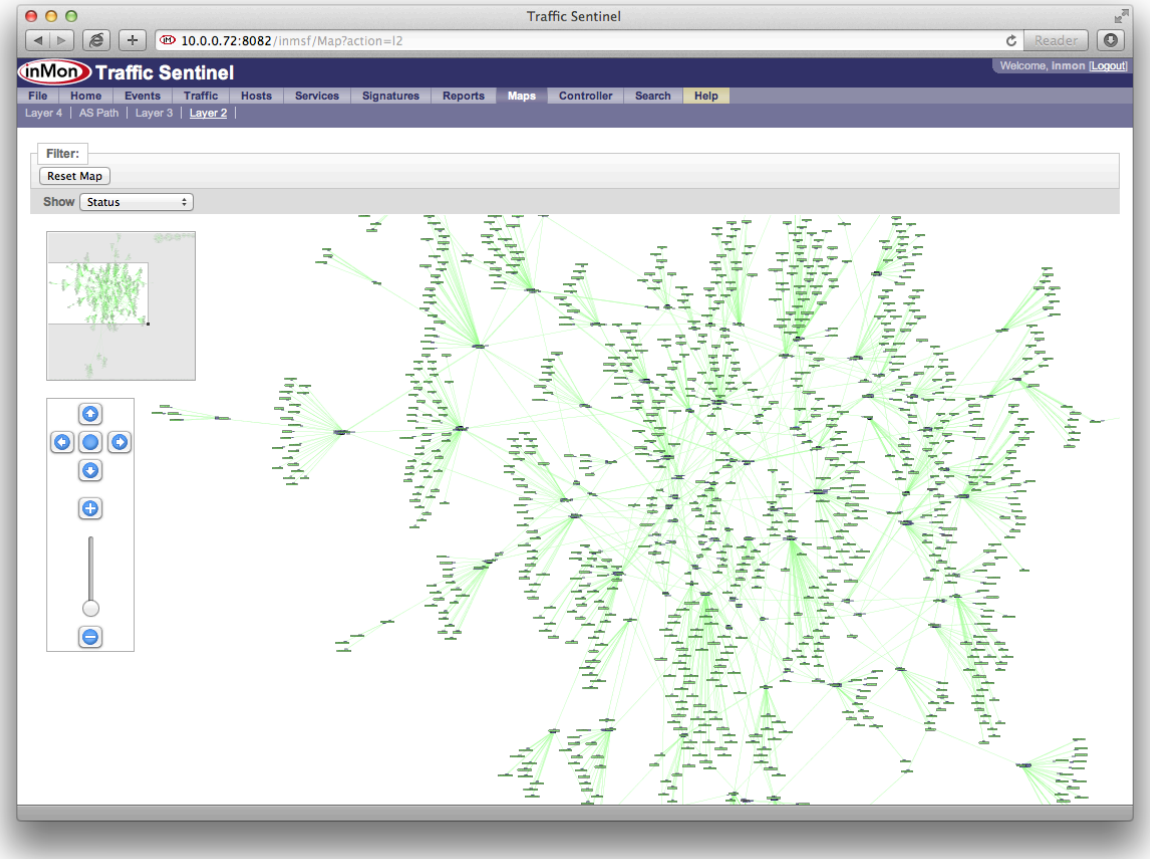
Break out traffic by client, server and port

(graph based on samples from100Gbit link carrying 14,000,000 packets/second)

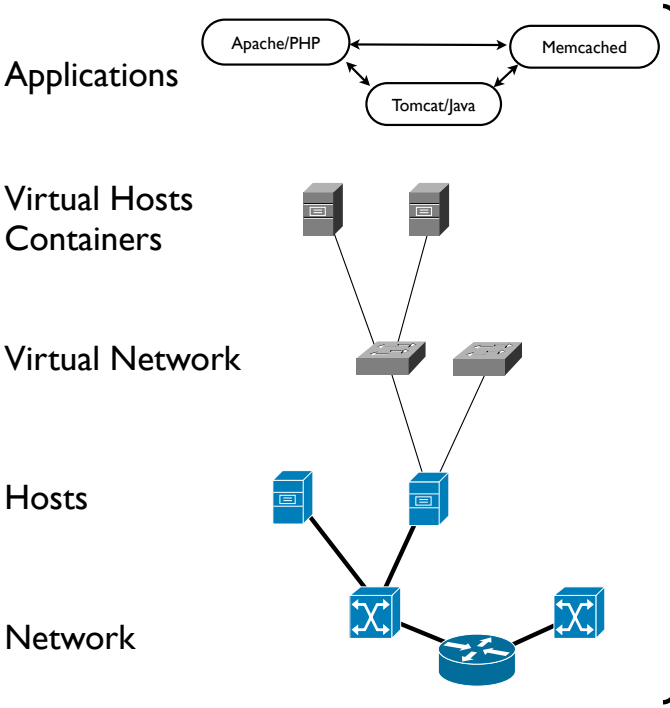
Scalability - link speed



Scalability - Network Size



Solution Architecture



sFlow
Telemetry

sFlow-RT

REST
Analytics

- Real-time
- Scaleable
- Programmable

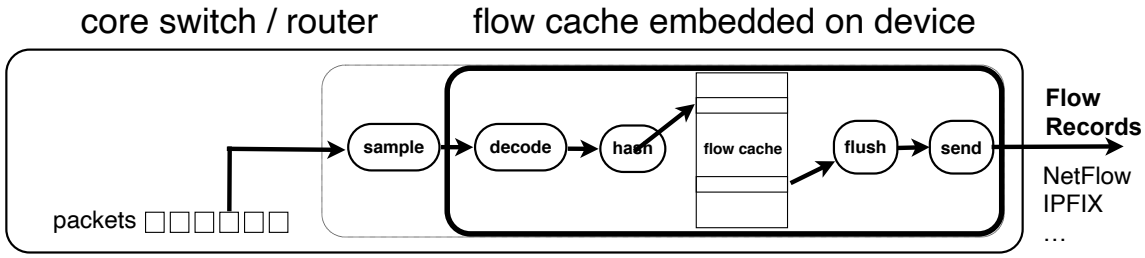
Orchestration

DevOps

SDN Controllers

Embedded monitoring of all switches, all routers, all servers, all applications, all the time

sFlow: Disaggregated Analytics

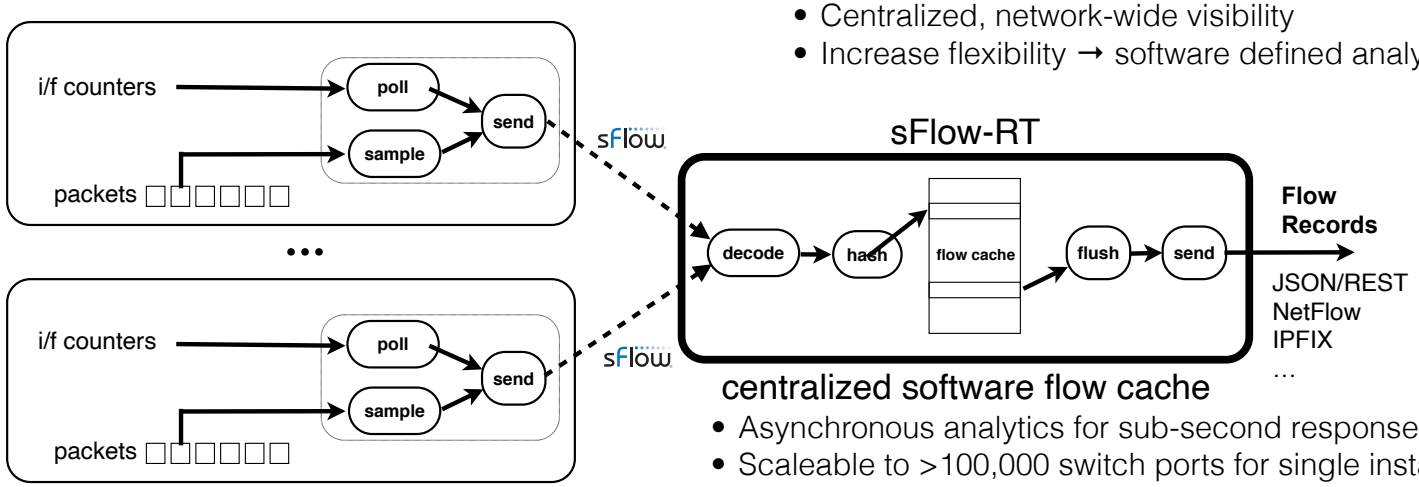


Stream counters:

- Scale-out alternative to SNMP polling

Move flow cache from ASIC to external software:

- Reduce ASIC cost / complexity
- Fast response (data not sitting on switch)
- Centralized, network-wide visibility
- Increase flexibility → software defined analytics

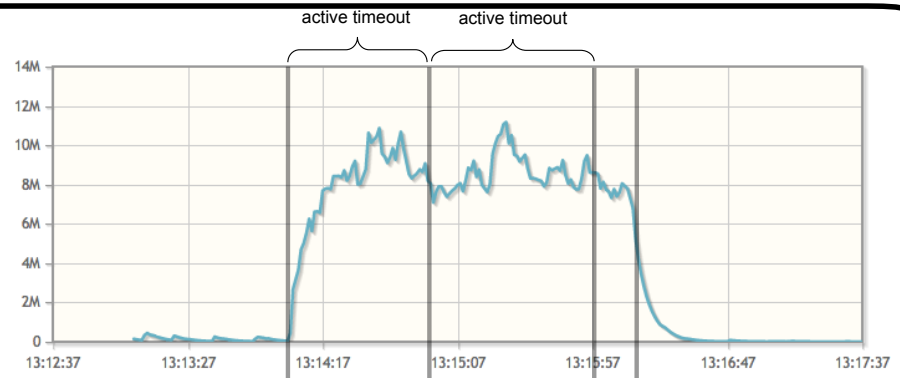


- Asynchronous analytics for sub-second response time
- Scaleable to >100,000 switch ports for single instance
- JavaScript/REST platform for analytics based apps

Continuous Tracking of Large Flows

sFlow does not use flow cache, so real-time chart accurately reports flow in progress

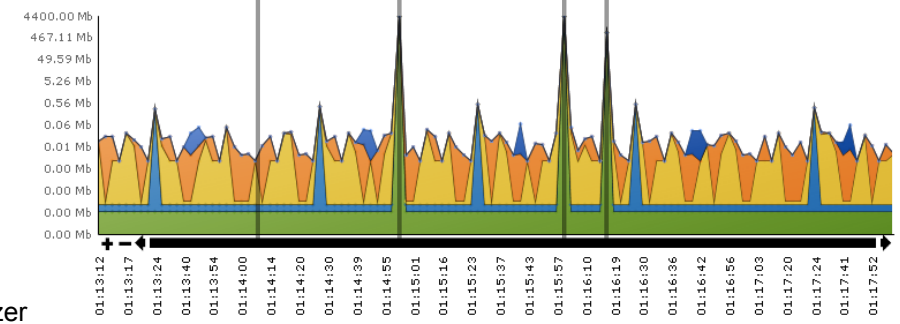
Chart: sFlow-RT real-time analyzer



NetFlow

NetFlow spikes caused by flow cache active-timeout for long running connections

Chart: SolarWinds Real-Time NetFlow Analyzer



NetFlow active timeout delays large flow detection, limits value of signal for real-time visibility and control

Example: DDoS mitigation

Define Flows to be managed

```
setFlow('udp_target',{keys:'ipdestination,udpsourceport',value:'frames'});
```

Define Threshold for action

```
setThreshold('attack',{metric:'udp_target', value:thresh, byFlow:true});
```

Act on threshold events, in this example by creating BGP Flowspec rule matching DDoS amplification attack traffic

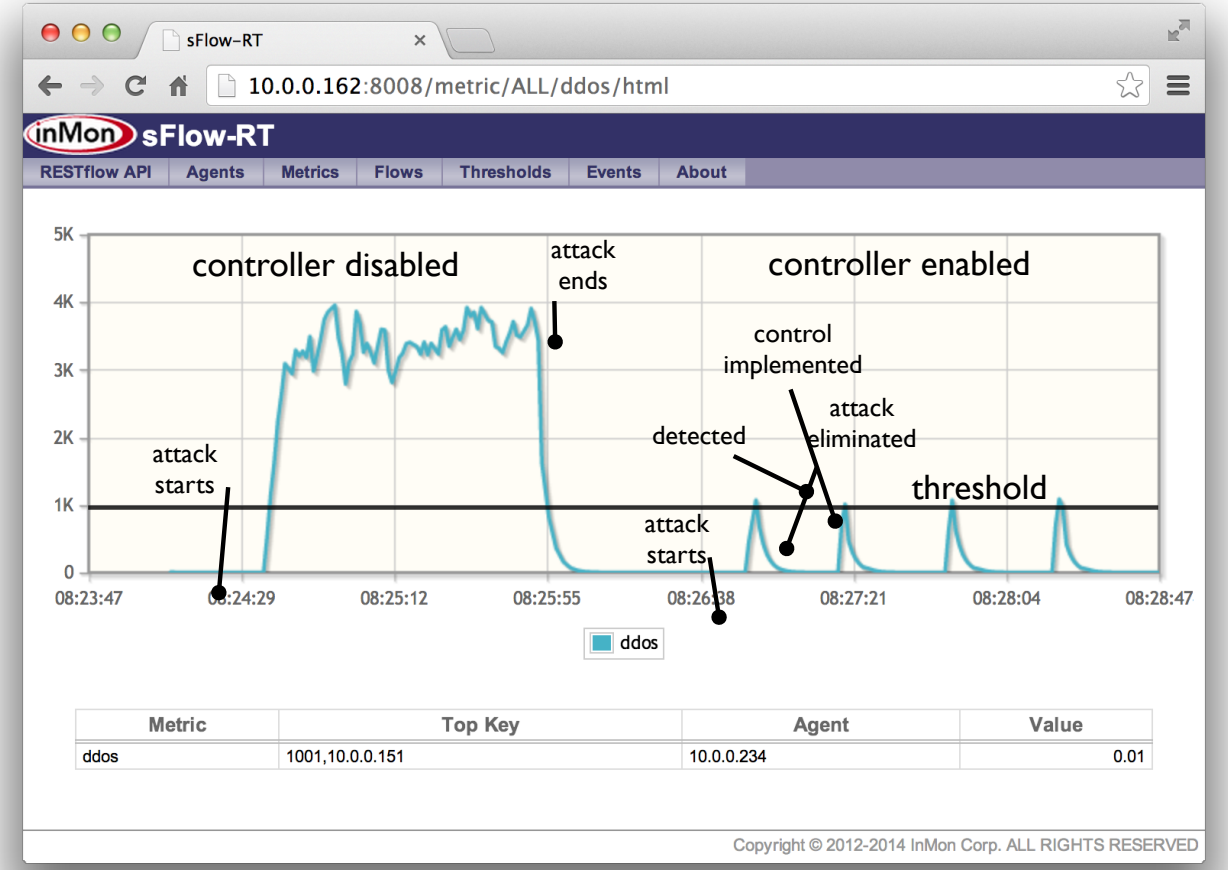
```
setEventHandler(function(evt) {
  var key = evt.flowKey;
  if(controls[key]) return;

  var [ip,port] = key.split(',');
  var flow = {
    'match':{
      'protocol':'=17',
      'source-port':'='+port,
      'destination': ip
    },
    'then': {'traffic-rate':0}
  };
  controls[key] = {time:evt.timestamp, target: ip, port: port, flow:flow};
  bgpAddFlow(router, flow);
  logInfo('block target='+ip+' port='+port);
},['attack']);
```

Remove actions when no longer needed

```
setIntervalHandler(function(now) {
  for(var key in controls) {
    if(now - controls[key].time < 1000 * 60 * block_minutes) continue;
    var control = controls[key];
    delete controls[key];
    bgpRemoveFlow(router,control.flow);
    logInfo('allow target='+control.target+' port='+control.port);
  }
});
```

sFlow-RT script detects and drops DDoS UDP amplification attacks within 1 second

DDoS Mitigation

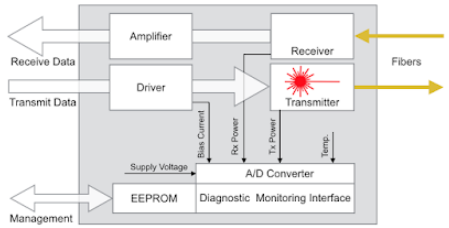


OK, OK, so what's new?

1. Optical Interface Monitoring
2. TCP Performance Monitoring
3. Discarded Packet Headers
4. Transit Delay and Queueing

Optical Interface Monitoring

```
struct lane {
  unsigned int index; /* 1-based index of lane within module, 0=unknown */
  unsigned int tx_bias_current; /* microamps */
  unsigned int tx_power;        /* microwatts */
  unsigned int tx_power_min;    /* microwatts */
  unsigned int tx_power_max;    /* microwatts */
  unsigned int tx_wavelength;   /* nanometers */
  unsigned int rx_power;        /* microwatts */
  unsigned int rx_power_min;    /* microwatts */
  unsigned int rx_power_max;    /* microwatts */
  unsigned int rx_wavelength;   /* nanometers */
}

/* Optical SFP / QSFP metrics */
/* opaque = counter_data; enterprise=0; format=10 */
struct sfp {
  unsigned int module_id;
  unsigned int module_num_lanes;      /* total number of lanes in module */
  unsigned int module_supply_voltage; /* millivolts */
  int module_temperature;             /* thousandths of a degree Celsius */
  lane<> lanes;
}
```

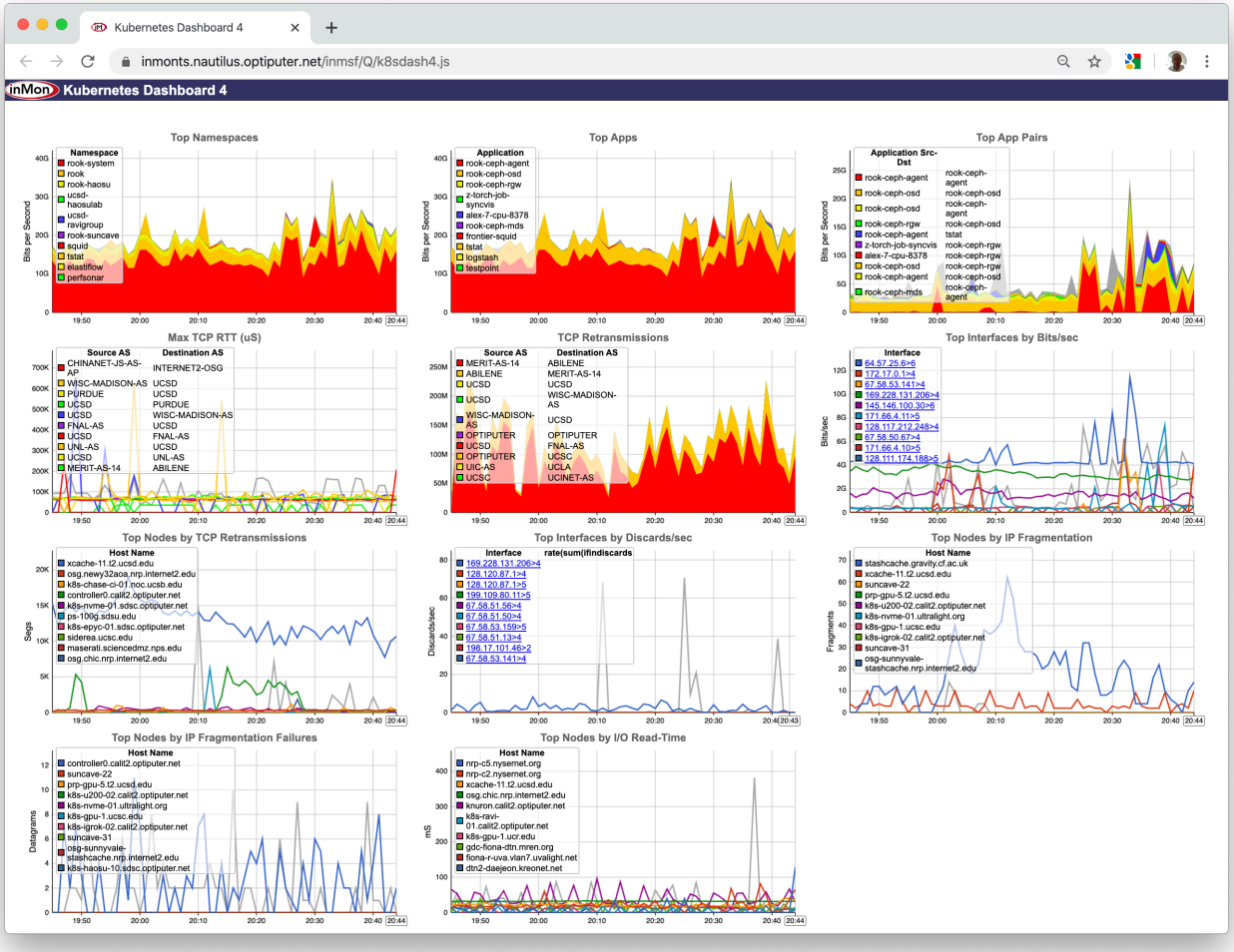


TCP Performance Monitoring

```
startSample ----------------------
sampleType_tag 0:1
sampleType FLOWSAMPLE
sampleSequenceNo 153026
sourceId 0:2
meanSkipCount 10
samplePool 1530260
dropEvents 0
inputPort 1073741823
outputPort 2
flowBlock_tag 0:2209
tcpinfo_direction sent
tcpinfo_send_mss 1448
tcpinfo_receive_mss 536
tcpinfo_unacked_pkts 0
tcpinfo_lost_pkts 0
tcpinfo_retrans_pkts 0
tcpinfo_path_mtu 1500
tcpinfo_rtt_uS 773
tcpinfo_rtt_uS_var 137
tcpinfo_send_congestion_win 10
tcpinfo_reordering 3
tcpinfo_rtt_uS_min 0
flowBlock_tag 0:1
flowSampleType HEADER
headerProtocol 1
sampledPacketSize 84
strippedBytes 4
headerLen 66
headerBytes 08-00-27-09-5C-F7-08-00-27-B8-32-6D-08-00-45-C0-00-34-60-79-40-00-01-06-03-7E-0A-00-00-88-0A-00-00-86-84-47-00-B3-50-6C-E7-E7-D8-49-29-17
ED-15-34-00-00-01-01-08-0A-18-09-85-3A-23-8C-C6-61
dstMAC 080027095cf7
srcMAC 080027b8326d
IPSize 66
ip.tot_len 52
srcIP 10.0.0.136
dstIP 10.0.0.134
…
```

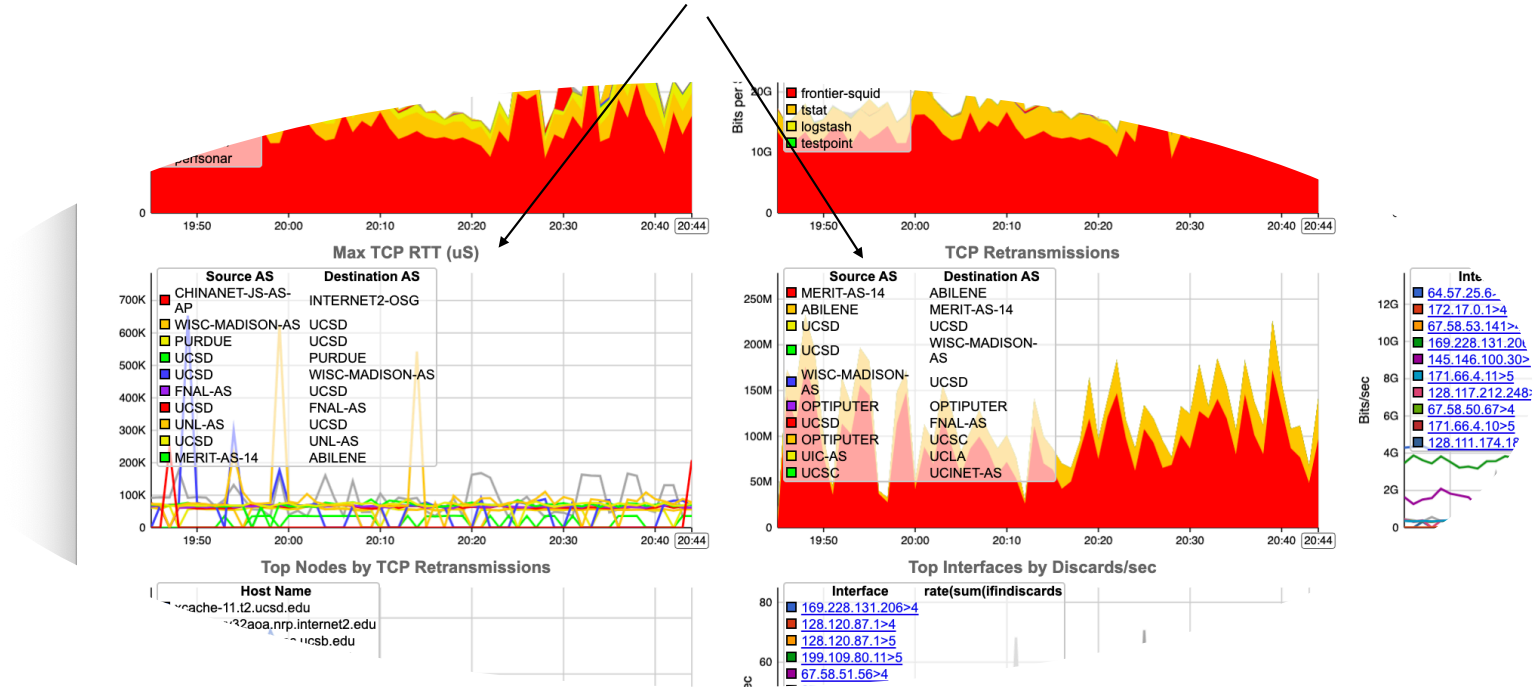
TCP info goes out with packet header sample

TCP Performance Monitoring



TCP Performance Monitoring

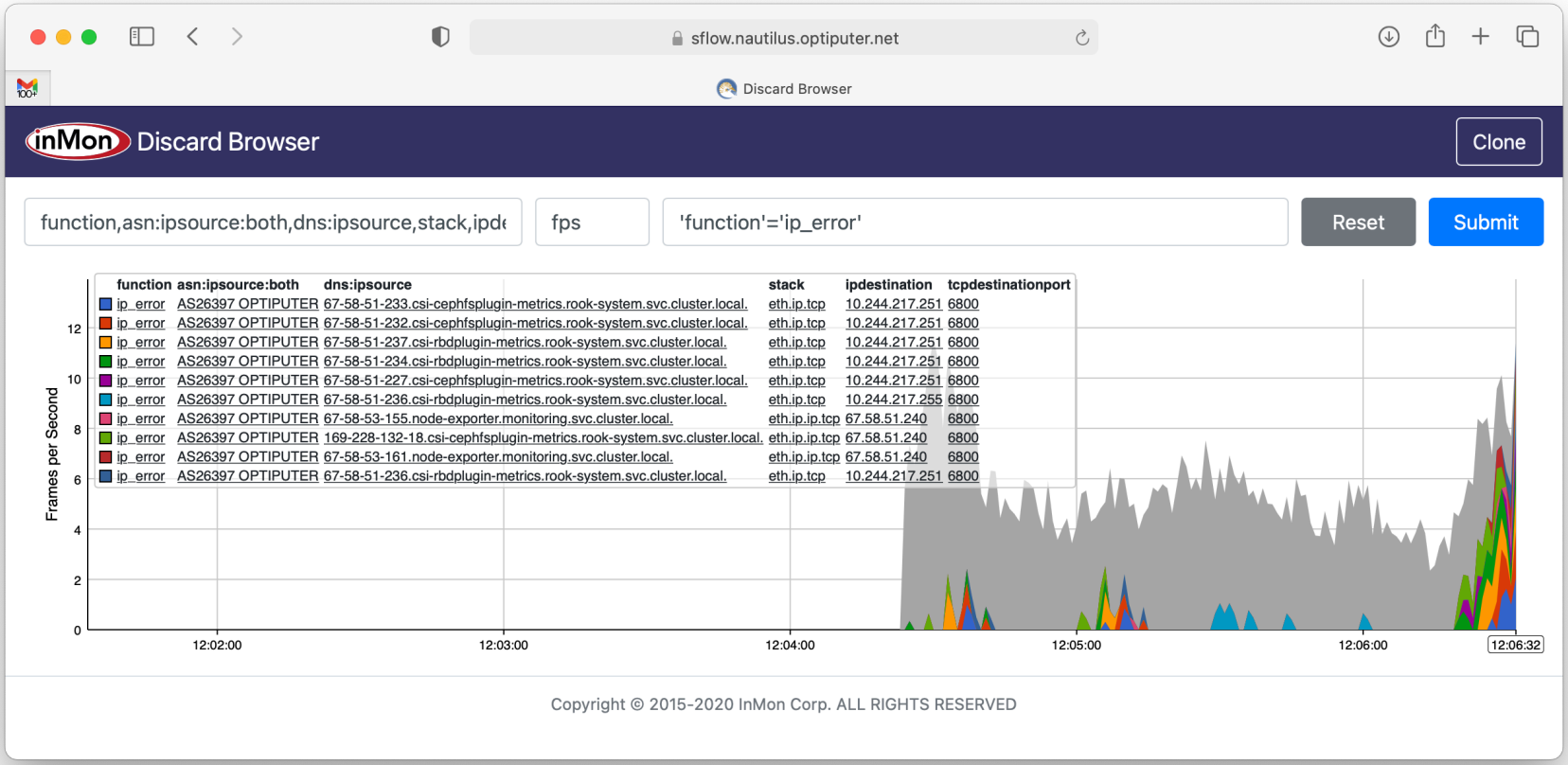
Pairwise source-destination measurements from packet samples



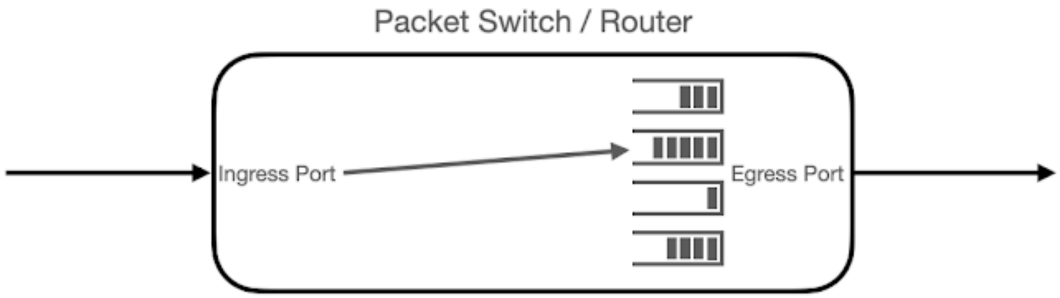
Discarded Packet Headers

```
struct discarded_packet {
   unsigned int sequence_number;  /* Incremented with each discarded packet
                                     record generated by this source_id. */
   sflow_data_source_expanded source_id; /* sFlowDataSource */
   unsigned int drops;            /* Number of times that the sFlow agent
                                     detected that a discarded packet record
                                     was dropped by the rate limit, or because
                                     of a lack of resources. The drops counter
                                     reports the total number of drops detected
                                     since the agent was last reset. Note: An
                                     agent that cannot detect drops will always
                                     report zero. */
   unsigned int inputifindex;     /* If set, ifIndex of interface packet was
                                     received on. Zero if unknown. Must identify
                                     physical port consistent with flow_sample
                                     input interface. */
   unsigned int outputifindex;    /* If set, ifIndex for egress drops. Zero
                                     otherwise. Must identify physical port
                                     consistent with flow_sample output
                                     interface. */
   drop_reason reason;            /* Reason for dropping packet. */
   flow_record discard_records<>; /* Information about the discarded packet. */
}
```

Discarded Packet Headers



Transit Delay and Queueing



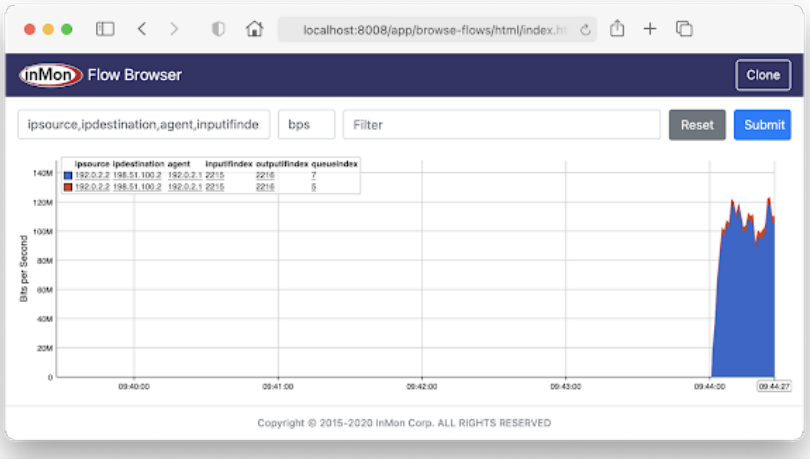
Transit Delay and Queueing

```
startSample ----------------------
sampleType_tag 0:1
sampleType FLOWSAMPLE
sampleSequenceNo 91159
sourceId 0:2216
meanSkipCount 400
samplePool 36463600
dropEvents 0
inputPort 2215
outputPort 2216
flowBlock_tag 0:1036
extendedType egress_queue
egress_queue_id 7
flowBlock_tag 0:1040
extendedType queue_depth
queue_depth_bytes 11354112
flowBlock_tag 0:1039
extendedType transit_delay
transit_delay_nS 839660224
flowBlock_tag 0:1
flowSampleType HEADER
headerProtocol 1
sampledPacketSize 1446
strippedBytes 4
headerLen 128
headerBytes 98-03-9B-8F-B5-CC-98-03-9B-94-C7-D5-08-00-45-16-05-94-12-C7-00-00-FE-11-B8-43-C0-00-02-02-C6-33-64-02-30-39-D4-31-05-80-
D7-1D-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-4
2-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42
dstMAC 98039b8fb5cc
srcMAC 98039b94c7d5
IPSize 1428
ip.tot_len 1428
srcIP 192.0.2.2
dstIP 198.51.100.2
IPProtocol 17
IPTOS 22
IPTTL 254
IPID 50962
UDPSrcPort 12345
UDPDstPort 54321
UDPBytes 1408
endSample   ----------------------
```

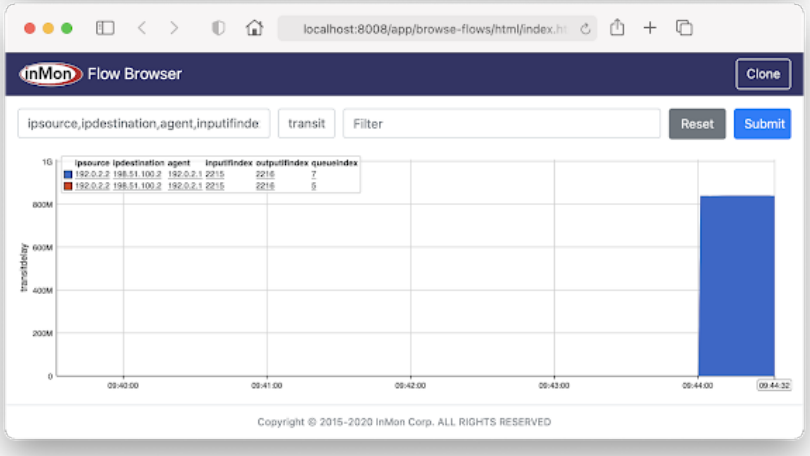
Queueing info goes out with packet header sample

Transit Delay and Queueing

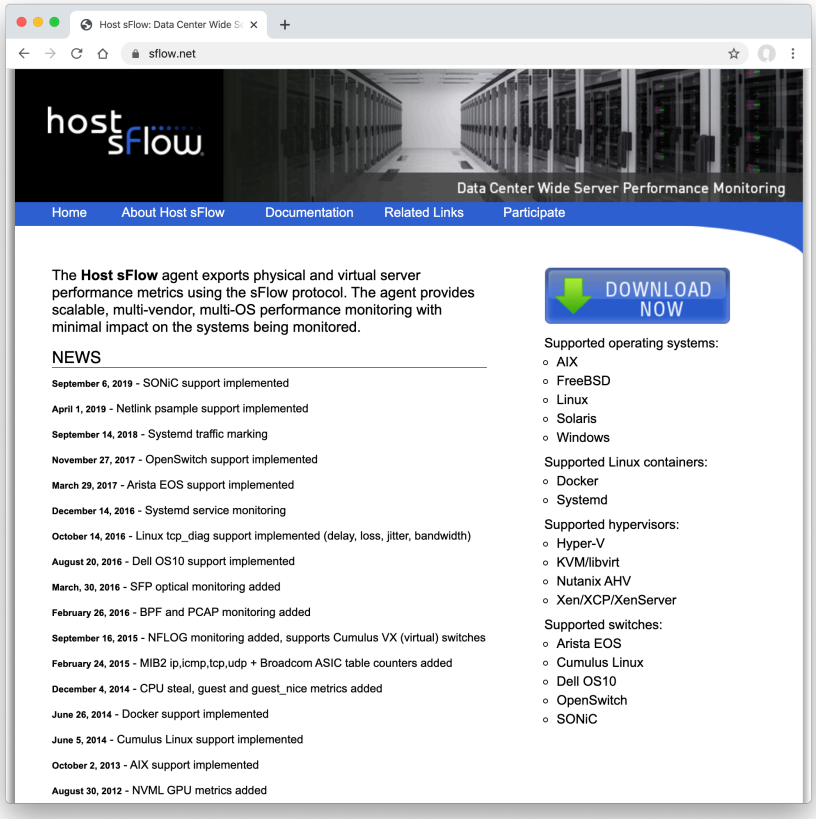
Bits/sec



Transit Delay

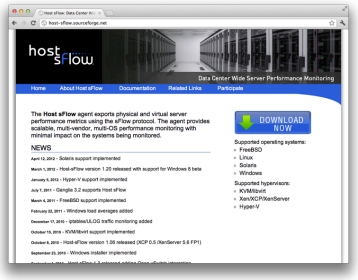


Host sFlow



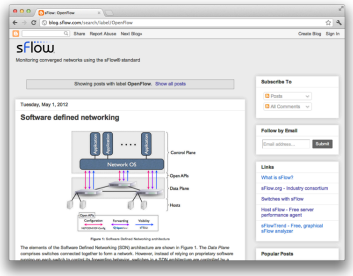
- Host sFlow (sFlow.net), free open source agent.
- Exports standard sFlow metrics for network, host.
- Includes TCP QoS metrics (latency, loss, retransmissions) and packet-drop monitoring.
- Standard sFlow data model, shared with switches, provides integrated view of network, host, and application performance
- Extend visibility into public/ private cloud hypervisors, virtual machines, and containers

More information



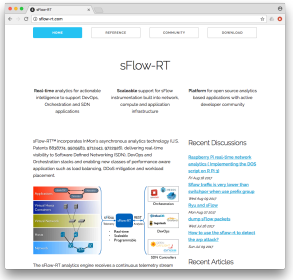
sflow.net

freeware agents



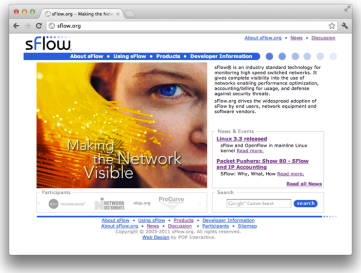
blog.sflow.com

articles



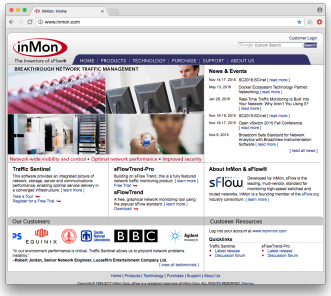
sflow-rt.com

real-time analytics
closed-loop control



sflow.org

sFlow standard



inmon.com

commercial products