sFlow based automata for finding hierarchical heavy hitter

Friday,

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.

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).

```
void
print_pkt(const opennsl_pkt_t *pkt)
{
  uint8  i;

  if (pkt == NULL) {
    return;
  }
```

```
    VLOG_DBG("[%s:%d]; # of blocks=%d, pkt_len=%d, tot_len=%d",
        __FUNCTION__, __LINE__, pkt->blk_count,
        pkt->pkt_len, pkt->tot_len);

    VLOG_DBG("[%s:%d]; vlan=%d, src_port=%d, dest_port=%d, "
        "rx_port=%d, untagged=%d, vtag0=%d, vtag1=%d, "
        "vtag2=%d, vtag3=%d", __FUNCTION__, __LINE__,
        pkt->vlan, pkt->src_port, pkt->dest_port, pkt->rx_port,
        pkt->rx_untagged, pkt->_vtag[0], pkt->_vtag[1],
        pkt->_vtag[2], pkt->_vtag[3]);

#define PKT pkt->pkt_data[i].data

    for(i=0; i<pkt->blk_count; i++) {
        VLOG_DBG("[%s:%d]; blk num=%d, blk len=%d", __FUNCTION__, __LINE__,
            i, pkt->pkt_data[i].len);

        VLOG_DBG("==============ETHERNET HEADER===============");
        VLOG_DBG("DMAC: %02X:%02X:%02X:%02X:%02X:%02X",
            pkt->pkt_data[i].data[0], pkt->pkt_data[i].data[1],
            pkt->pkt_data[i].data[2], pkt->pkt_data[i].data[3],
            pkt->pkt_data[i].data[4], pkt->pkt_data[i].data[5]);

        VLOG_DBG("SMAC: %02X:%02X:%02X:%02X:%02X:%02X",
            pkt->pkt_data[i].data[6], pkt->pkt_data[i].data[7],
            pkt->pkt_data[i].data[8], pkt->pkt_data[i].data[9],
            pkt->pkt_data[i].data[10], pkt->pkt_data[i].data[11]);

        VLOG_DBG("EtherType: %02X%02X Vlan: %02X%02X EtherType: %02X%02X",
            pkt->pkt_data[i].data[12], pkt->pkt_data[i].data[13],
            pkt->pkt_data[i].data[14], pkt->pkt_data[i].data[15],
            pkt->pkt_data[i].data[16], pkt->pkt_data[i].data[17]);

        VLOG_DBG("==============IPv4  HEADER===============");
        VLOG_DBG("Ver/IHL: %02X ToS: %02X Len: %02X%02X",
            pkt->pkt_data[i].data[18], pkt->pkt_data[i].data[19],
            pkt->pkt_data[i].data[20], pkt->pkt_data[i].data[21]);

        VLOG_DBG("Src: %02X.%02X.%02X.%02X",
            pkt->pkt_data[i].data[30], pkt->pkt_data[i].data[31],
            pkt->pkt_data[i].data[32], pkt->pkt_data[i].data[33]);

        VLOG_DBG("Dest: %02X.%02X.%02X.%02X",
            pkt->pkt_data[i].data[34], pkt->pkt_data[i].data[35],
            pkt->pkt_data[i].data[36], pkt->pkt_data[i].data[37]);
    }
}
```