Who we are

Alexander Gall
Network engineer at SWITCH, Snabb contributor since 2014
alexander.gall@switch.ch

Max Rottenkolber
Works on Snabb since 2014
maximilian@igalia.com
Snabbflow at SWITCH

Motivation, function, deployment
Netflow at SWITCH

The concept of a “flow” is the primary mechanism used to analyze network traffic

- 5-tuple <src address, dst address, IP protocol, src port, dst port>
- Aggregates bytes/packets, additional custom fields (TCP flags, AS numbers…)
- Evolved from Cisco-proprietary to IETF standard IPFIX
- Unsampled (process every packet) or sampled (process 1 in n packets)

In use at SWITCH since mid 1990s. Until a few years ago

- Provided in Hardware by the routers
- Unsampled

Modern routers moved to sampling to cope with high-volume traffic
Sampled vs Unsampled

Sampling approximates real values well for volume-based metrics. Why use unsampled Netflow?

- Fine-grained analysis of security incidents
- Debugging of network problems for single flows, e.g.
  - TCP handshake
  - DNS transaction

Requires

- Move from router to external appliance for Netflow generation
- Find a scalable and cost-effective solution: Snabbflow
SWITCH Network

- Peak traffic values (aggregate external traffic, ingress + egress)
  - ~180Gbps
  - ~20Mpps
  - ~350k flows per second (>500kfps with aggressive port-scans)
- Aggregate IPFIX export data rate 200-300Mbps
- Average flow rate 200k/s, 1.5TiB flow data per day (~100 bytes/flow)
- Interface types: optical 10G, 100G soon 400G
- Until 2015 Netflow export on (Cisco) routers
- 2015-2020 commercial Netflow exporter using hardware acceleration
- Since 2020 Snabbflow
Per-PoP Exporter Architecture

- Optical taps on external interfaces to copy packets
- “Packet-Broker” to aggregate traffic to 2x100 Gbps links to Snabbflow exporter
  - Use VLAN tags to identify original router ports
  - “Whitebox” switch
    - EdgeCore Wedge100BF-32x/AS9516-32D
    - Tofino/Tofino2 ASIC
    - P4-programmable
    - Separate project: https://github.com/alexandergall/packet-broker
- Snabbflow on commodity 1RU server
  - AMD Epyc or Intel Xeon, 12-24 cores, ~128GiB RAM for large flow tables
  - 2x100G Mellanox ConnectX-5 NICs
Switch border router

8-port splitter

Packet Broker
adds vlan for each “color” so we know where packets came from

Snabbflow

Vlan 151
Vlan 152
Vlan 153
Vlan 154
Vlan 155
Vlan 156
Vlan 157
Vlan 158
Vlan 159
Vlan 160
Vlan 161
Vlan 162
Vlan 163
Vlan 164
Vlan 165
Vlan 166

Foreign BR1
Foreign BR2
Foreign BR3
Foreign BR8
Foreign BRx
Features of Snabbflow

snabb ipfix probe

Scaling, configuration, monitoring and their implementation
- A toolkit for building fast packet processing applications using a high-level programming language

- Written in Lua (using the amazing LuaJIT compiler)!

- Packet I/O without going through the kernel (kernel-bypass / userspace networking)

- Open source and independent (not sponsored by any $vendor)
● Simple > Complex
● Small > Large
● Commodity > Proprietary
function FlowSet:record_flows(timestamp)
    local entry = self.scratch_entry

    for i=1,link.nreadable(self.incoming) do
        local pkt = link.receive(self.incoming)
        self.template:extract(pkt, timestamp, entry)

        local lookup_result = self.table:lookup_ptr(entry.key)
        if lookup_result == nil then
            self.table:add(entry.key, entry.value)
        else
            self.template:accumulate(lookup_result, entry, pkt)
        end
        packet.free(pkt)
    end
end
flushing ipfix records

-- Walk through flow set to see if flow records need to be expired.
-- Collect expired records and export them to the collector.

function FlowSet:expire_records(out, now)
  local cursor = self.expiry_cursor

  ...
  for i = 1, self.table_tb:take_burst() do
    local entry
    cursor, entry = self.table:next_entry(cursor, cursor + 1)
    ...
    if entry then
      ...
      self:add_data_record(entry.key, out)
    end
  end
  if self.flush_timer() then self:flush_data_records(out) end
end
High-level overview

- 100G NIC (Driver written in Lua)
- Snabb ipfix probe
- tun/tap (Linux kernel network stack)
- ipfix collector
Scaling via hardware RSS

100G NIC (Driver written in Lua)

Snabb ipfix probe

RSS forwards distinct sets of flows to distinct Snabbflow processes

Horizontal scaling!

Circle = CPU core
Scaling via software RSS

Software RSS forwards distinct sets of flows to distinct exporter processes extracting different sets of metadata.

Isolate workloads! (Complex packet inspection does not bog down basic metadata export)

Circle = CPU core
“Apps” and multi-processing

Snabb programs are organized in “apps” (independent packet processing components)

Communicate with each other via “links”:

```python
p = link.receive(input)
link.transmit(output, p)
```
“Apps” and multi-processing (lib.interlink)

Packets can be shared with low overhead across CPU core boundaries using “interlinks”.

**Link** interface remains orthogonal:

```python
p = link.receive(input)
link.transmit(output, p)
```
lib.ptree

- Can query and update data-plane configuration
- Knows about data-plane state
- No particular latency requirements
- Manages multiple data-plane workers (on dedicated CPU cores)

- Soft real-time! No messing around!
- Receives configuration updates from manager
- Writes state counters to shared memory
lib.yang

Application configuration and state are described in a **YANG schema**.

```
$ snabb config set my-process / < ipfix.conf

$ snabb config get-state my-process \
   /snabbflow-state/exporter[name=ip]

packets-dropped 0;
packets-ignored 129326;
packets-received 499996;
template {
   id 1512;
   flow-export-packets 115;
   flows-exported 1318;
   packets-processed 12034;
...```
snabb-snabbflow-v1.yang

module snabb-snabbflow-v1 {
  
  container snabbflow-config {
    description
    "Configuration for the Snabbflow IPFIX exporter."
  }

  list interface {
    key device;
    unique "name vlan-tag";

    description
    "Interfaces serving as IPFIX Observation Points.";

    leaf device {
      type pci-address;
      description
      "PCI address of the network device.";
    }
  }

  ...
Flight recorder

- Minimal overhead: always on! (if you want it)
- Stores useful data
  - JIT trace info
  - Trace profiles (sampled)
  - High-frequency event log (sampled)
- Can be analyzed while running or post mortem
  - tar cf blackbox.tar /var/run/snabb; scp blackbox.tar ...
Where does my program spend its time?

Does the JIT have issues generating efficient code?

Includes full IR / assembly dump for each compiled trace!
Latency histograms derived from event log

Here: ipfix app takes ~35us to process a batch of packets.

**Useful for debugging tail latencies.**

Can add arbitrary application-specific, user-defined events.
If you write a Snabb program today

You can reuse all of these components and more!
Thanks for your attention!

Questions?

GitHub: snabbco/snabb
Snabbflow:
alexander.gall@switch.ch

SWITCH

Commercial support for Snabb:
maximilian@igalia.com

igalia