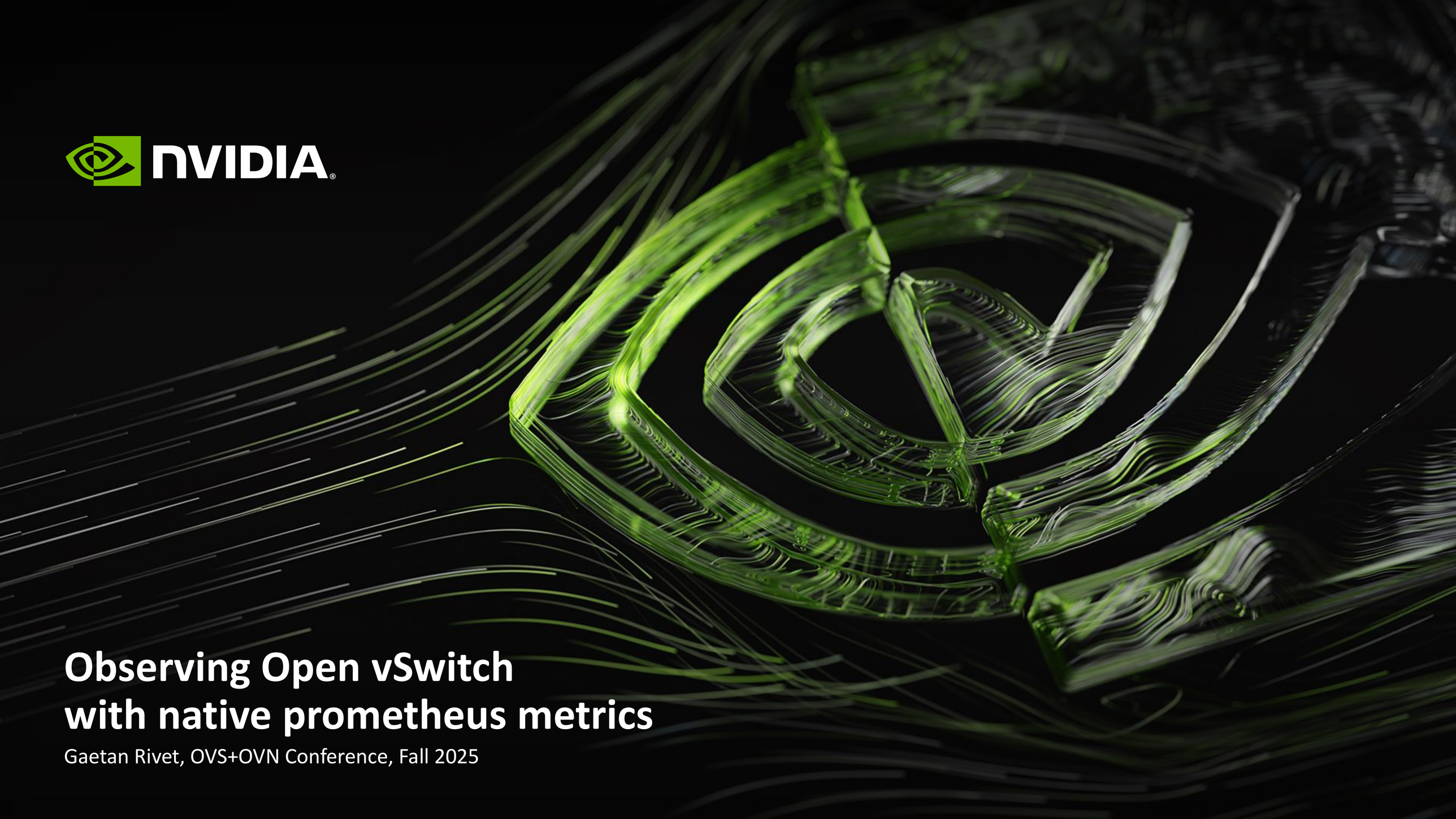
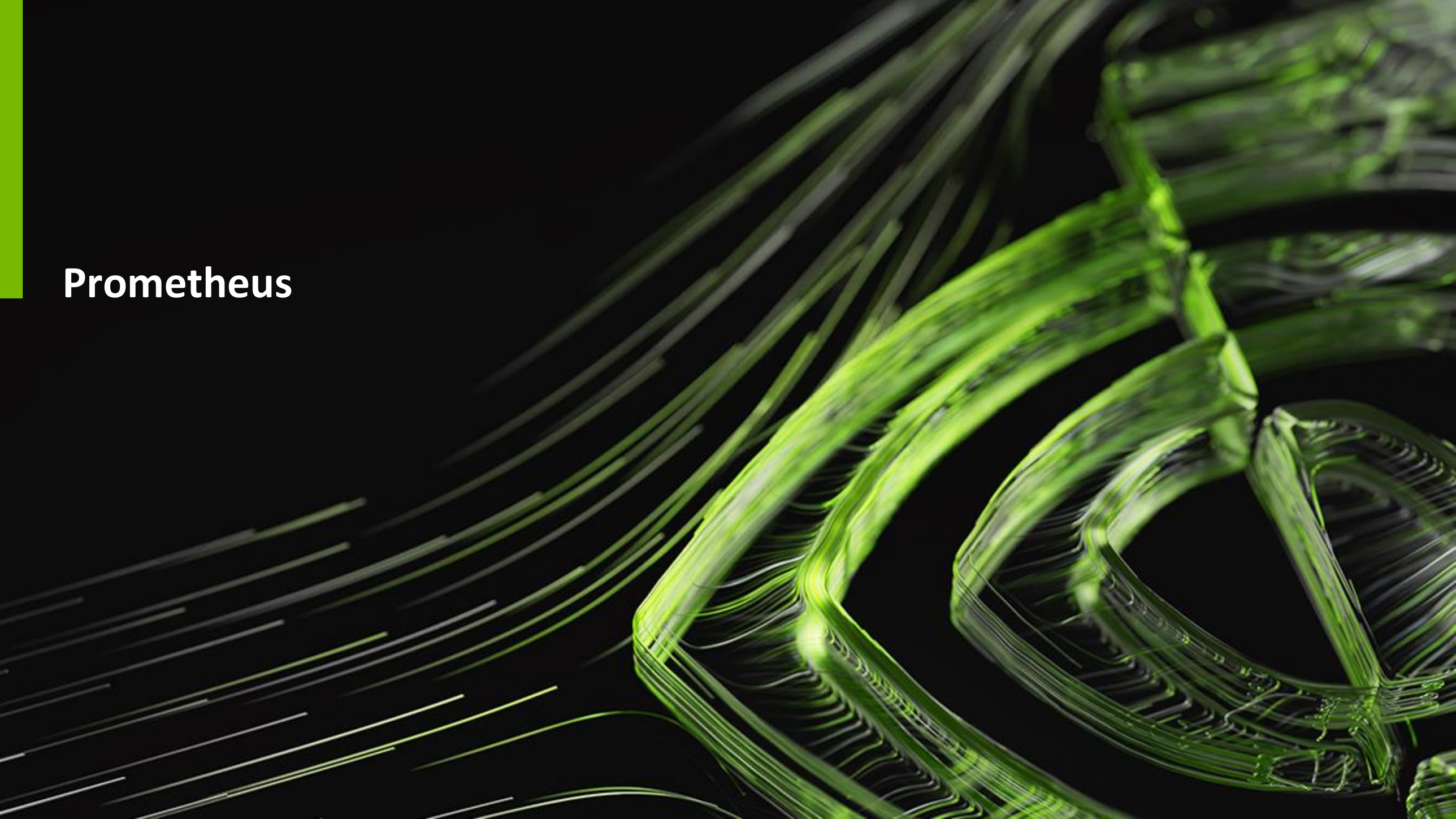


Observing Open vSwitch with native prometheus metrics

Gaetan Rivet, OVS+OVN Conference, Fall 2025

- Prometheus
- Metrics in Open vSwitch
- Implementation
- What comes next

Prometheus



Prometheus / OpenMetrics

- Prometheus: Monitoring system (alerting) & time-series database.
- Defines a data model for applications metrics.
- De-facto standard, failed IETF spec but active and widely deployed project.
- “Pull” model: Prometheus sends a request and the application responds.
- No pre-defined schema, ”free-form” metrics.

Application	→	Prometheus (DB)	→	Grafana (or others)
Generates		Store+Monitor		Display

Spec: https://github.com/prometheus/OpenMetrics/tree/main/specification

Many alternatives: OpenTelemetry, statsD, vendor-specific (now: DOCA-Telemetry) etc…
Prometheus is widely used, has a strong ecosystem.
“Reasonable” implementation (”pull” better suited for reliable analysis than ”push”).

→ Was chosen as starting point for OVS metrics.

Prometheus data Model

Prometheus defines 4 metrics types:

- **Counter**
  Value only goes up. Suffix “_total”.
- **Gauge**
  Value goes up and down.
- **Histogram**
  Value distribution
  over pre-defined bins.
- **Summary**
  Value distribution over
  quantiles.

Ignore empty lines.

Comments starting with ‘#’.

Optional HELP, TYPE, UNIT lines to describe a metric.

Metric EBNF syntax:

```
metric_name [
  "{" label_name "=" `"` label_value `"` { ","
    label_name "=" `"` label_value `"` } [ "," ]
  "}"
] value [ timestamp ]
```

```
# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter http_requests_total{method="post",code="200"} 1027 1395066363000
http_requests_total{method="post",code="400"} 3 1395066363000

# Escaping in label values: msdos_file_access_time_seconds{path="C:\\DIR\\FILE.TXT",error="Cannot find
file:\n\"FILE.TXT\""} 1.458255915e9

# Minimalistic line:
metric_without_timestamp_and_labels 12.47

# A weird metric from before the epoch:
something_weird{problem="division by zero"} +Inf -3982045
```

Prometheus data Model

Prometheus defines 4 metrics types:

- **Counter**
  Value only goes up. Suffix “_total”.
- **Gauge**
  Value goes up and down.
- **Histogram**
  Value distribution
  over pre-defined bins.
- **Summary**
  Value distribution over
  quantiles.

Ignore empty lines.

Comments starting with ‘#’.

Optional HELP, TYPE, UNIT lines to describe a metric.

Metric EBNF syntax:

```
metric_name [
  "{" label_name "=" `"` label_value `"` { ","
    label_name "=" `"` label_value `"` } [ "," ]
  "}"
] value [ timestamp ]
```

```
# A histogram, which has a pretty complex representation in the text format:
# HELP http_request_duration_seconds A histogram of the request duration.
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{le="0.05"} 24054
http_request_duration_seconds_bucket{le="0.1"} 33444
http_request_duration_seconds_bucket{le="0.2"} 100392
http_request_duration_seconds_bucket{le="0.5"} 129389
http_request_duration_seconds_bucket{le="1"} 133988
http_request_duration_seconds_bucket{le="+Inf"} 144320
http_request_duration_seconds_sum 53423
http_request_duration_seconds_count 144320
```

- Each bin as its own line suffixed with “_bucket”. **Buckets are cumulative.**
- There MUST be at least a “+Inf” bin, a “_sum” (sum of samples) and a “_count” (number of samples) lines. Mean of all samples is “sum / count”.
- Each bin line having an “le” label (less-than-or-equal).

Prometheus data Model

Prometheus defines 4 metrics types:

- **Counter**
  Value only goes up. Suffix “_total”.
- **Gauge**
  Value goes up and down.
- **Histogram**
  Value distribution
  over pre-defined bins.
- **Summary**
  Value distribution over
  quantiles.

Ignore empty lines.

Comments starting with ‘#’.

Optional HELP, TYPE, UNIT lines to describe a metric.

Metric EBNF syntax:

```
metric_name [
  "{" label_name "=" `"` label_value `"` { ","
    label_name "=" `"` label_value `"` } [ "," ]
  "}"
] value [ timestamp ]
```

```
# Finally, a summary, which has a complex representation, too:
# HELP rpc_duration_seconds A summary of the RPC duration in seconds.
# TYPE rpc_duration_seconds summary
rpc_duration_seconds{quantile="0.01"} 3102
rpc_duration_seconds{quantile="0.05"} 3272
rpc_duration_seconds{quantile="0.5"} 4773
rpc_duration_seconds{quantile="0.9"} 9001
rpc_duration_seconds{quantile="0.99"} 76656
rpc_duration_seconds_sum 1.7560473e+07
rpc_duration_seconds_count 2693
```

- Same format as a “histogram”, except bin label is named “quantile”.
- Each quantile must be between 0 and 1.

Metrics in Open vSwitch

Metrics in Open vSwitch

A new exporter

- OVS natively uses metrics as:
  - Counters -- e.g. NIC statistics, flow counters.
  - Gauge – e.g. resource usage: number of flows, connections, offloads, etc.
  - Histogram – e.g. PMD performance reported as histograms.
- Prometheus exporters already exist for OVS.
  - They usually assume OVS-kernel.
  - They are implemented many times: not controlled by OVS developers, competing implementations.
  - They are not well-integrated to OVS: each scrape requires many CLI calls, reads from kernel sources, etc.
    They are not written to prioritize OVS as source of truth: adds friction for userspace OVS.

OVS prometheus exporter was implemented with a different approach to solve / streamline those issues:

- Simple data format re-using internal OVS types. No summary at the time, has not yet been needed.
- Direct access to OVS: no need to “bubble-up” a new metric along the whole vertical (adding a new feature, then a CLI command, then a call to it in each exporter).
- Incentivizes using OVS as its first source of truth, for seamless transition from kernel to userspace.

Open vSwitch exporter

Metrics access

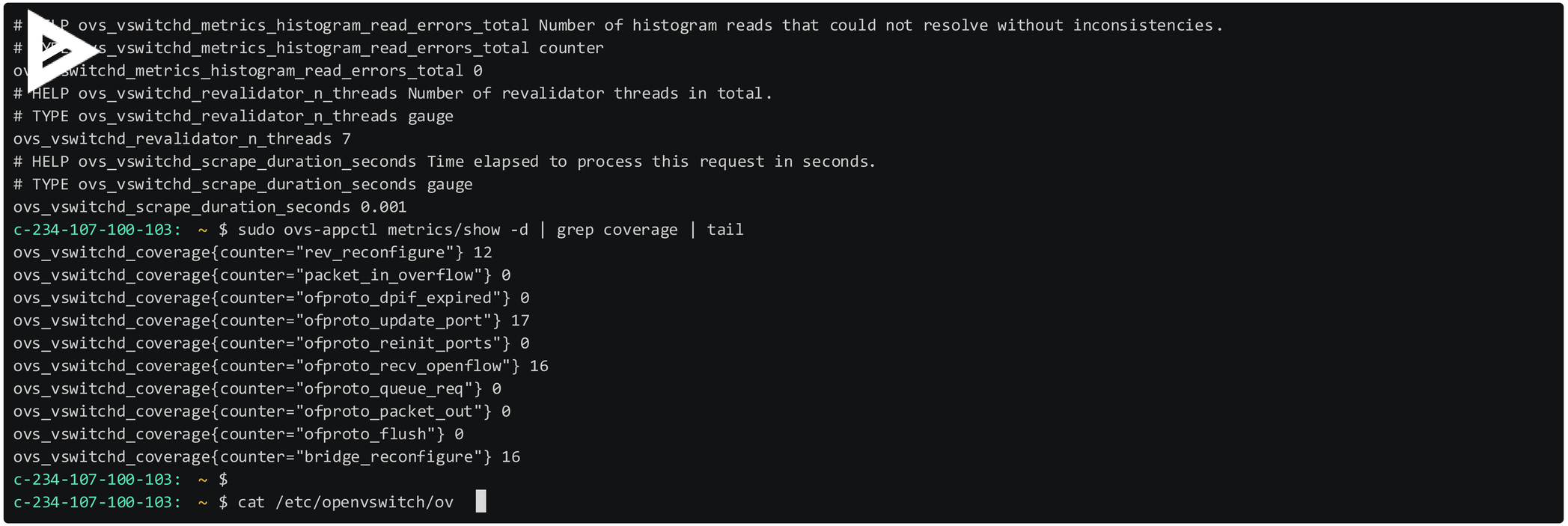
- A configurable systemd service: ovs-exporter (HTTP server on port 6103 by default)
- A CLI command: ovs-appctl metrics/show [-x] [-d]
- Python bindings: python/ovs/metrics.py
- 3 types of metrics generated by OVS: **counter**, **gauge**, **histogram**.
- Adding a new metric for a new features does not require adding new CLI command: they are all automatically added to the **exporter** output and **metrics/show**.

Initial implementation covering:

- Resource consumption: CPU, memory, datapath limits (flows, connections)
- Interfaces: packet and bytes counters, configuration, link state.
- Ofproto execution (upcalls, revalidation)
- Coverage counters

Open vSwitch exporter

Demo



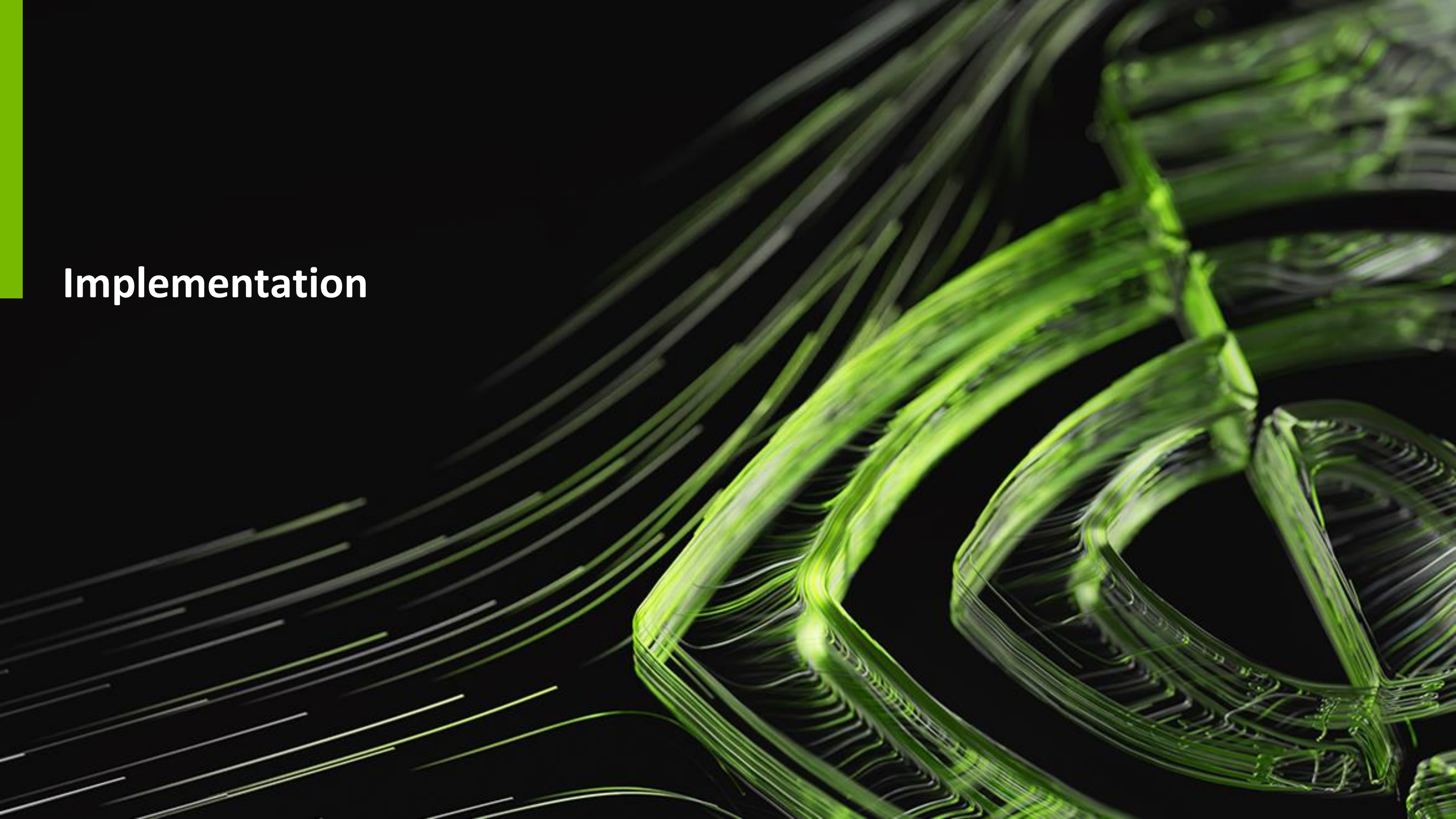
Open vSwitch exporter

Usage

- Start:
  - systemctl start ovs-exporter
  - /usr/share/openvswitch/scripts/ovs-ctl start-ovs-exporter
- Stop:
  - systemctl stop ovs-exporter
  - /usr/share/openvswitch/scripts/ovs-ctl stop-ovs-exporter
- Status:
  - systemctl status ovs-exporter
  - tail -f /var/log/openvswitch/ovs-exporter.log
- Documentation:
  - man ovs-exporter
  - ovs-exporter –h
- Configuration:

```
# /etc/openvswitch/ovs-exporter.conf:
[exporter]
port = 6103
extended = no
debug = no
```

Implementation



Metrics tree

- OVS metrics are organized as a hierarchy, represented using a tree.
- Each OVS process (e.g: **ovs-vswitchd**) is a root.
- Each metric is a leaf.
- The internal nodes of the tree reflect the structure of OVS modules.

```
ovs_vswitchd
 ├── datapath
|    ├── cache
|    ├── conntrack
|    └── offloads
 ├── interface
 ├── memory
 └── ofproto
```

Metrics tree

- OVS metrics are organized as a hierarchy, represented using a tree.
- Each OVS process (e.g: **ovs-vswitchd**) is a root.
- Each metric is a leaf.
- The internal nodes of the tree reflect the structure of OVS modules.

```
ovs_vswitchd
 ├── datapath
|    ├── cache
|    ├── conntrack
|    └── offloads
 ├── interface
 ├── memory
 └── ofproto
```

- Some nodes should be disabled, sometimes:
  - offloads, if `hw-offload=**false**`
  - dpif-netdev, if `dpif_is_netdev() returns **false**`
- Some nodes describe many objects:
  - datapath: netdev, netlink
  - Interfaces: p0, vf0, vf1
  - …

The tree must be able to describe **conditionals** (IF) and **collections** (FOREACH)

Metrics tree

- Generating a metrics output means visiting the tree.
- Each node is interpreted: foreach, if, read-value.
- Once interpreted, transformed into data-model.
  Today: Prometheus
  Tomorrow: Others? (output-agnostic)

To create a new metric, the **metrics** framework must be used.
API documented in **ovs/lib/metrics.h**.

- The Metrics tree is a set of Metrics node linked together.
- Each node is statically defined and references its parent.
- When a leaf is registered, its parent chain is itself registered, making it accessible from the root.

Let’s see a few examples.

Ofproto subsystem

Metrics hierarchy: Node type **Code in ofproto/ofproto-metrics.c**

```
ovs_vswitchd ----------------------  root       <implicit>
└── ofproto  ----------------------  subsystem   METRICS_SUBSYSTEM(ofproto)
    ├── n_bridges ------------------  entries     METRICS_ENTRIES(ofproto, n_bridges, …)
    ├── foreach_ofproto ------------  collection  METRICS_FOREACH(ofproto, foreach_ofproto, …)
    │   └── bridge_entries ---------  entries     METRICS_ENTRIES(foreach_ofproto, bridge_entries
    └── foreach_ofproto_nolabel ----  collection  METRICS_FOREACH(ofproto, foreach_ofproto_nolabel,
        └── foreach_ports ----------  collection  METRICS_FOREACH(foreach_ofproto_nolabel, foreach_ports,
            ├── port_entries -------  entries     METRICS_ENTRIES(foreach_ports, port_entries,
            └── netdev_labeled_info   labels      METRICS_LABEL(foreach_ports, netdev_labeled_info,
                └── netdev_info ----  entries     METRICS_ENTRIES(netdev_labeled_info, netdev_info,
```

Simple gauge read

Metrics hierarchy: Node type **Code in ofproto/ofproto-metrics.c**

```
ovs_vswitchd ----------------------  root
└── ofproto  ----------------------  subsystem
    ├── n_bridges ------------------  entries
    ├── foreach_ofproto ------------  collection
    │   └── bridge_entries ---------  entries
    └── foreach_ofproto_nolabel ----  collection
        └── foreach_ports ----------  collection
            ├── port_entries -------  entries
            └── netdev_labeled_info   labels
                └── netdev_info ----  entries
```

```
static void
bridge_n_read_value(double *values, void *it OVS_UNUSED)
{
    values[0] = hmap_count(&all_ofprotos);
}
METRICS_ENTRIES(ofproto, n_bridges, "bridge", bridge_n_read_value,
    METRICS_GAUGE(n_bridges,
        "Number of bridges present in the instance."),
);
```

```
# HELP ovs_vswitchd_bridge_n_bridges Number of bridges present in the instance.
# TYPE ovs_vswitchd_bridge_n_bridges gauge
ovs_vswitchd_bridge_n_bridges 2
```

Foreach node

Metrics hierarchy: Node type **Code in ofproto/ofproto-metrics.c**

```
ovs_vswitchd ----------------------  root
└── ofproto  ----------------------  subsystem
    ├── n_bridges ------------------  entries
    ├── foreach_ofproto ------------  collection
    |   └── bridge_entries ---------  entries
    └── foreach_ofproto_nolabel ----  collection
        └── foreach_ports ----------  collection
            ├── port_entries -------  entries
            └── netdev_labeled_info   labels
                └── netdev_info ----  entries
```

```
# HELP ovs_vswitchd_bridge A metric with a constant value '1' labeled by
# TYPE ovs_vswitchd_bridge gauge
ovs_vswitchd_bridge{name="br-phy",type=”netdev"} 1
ovs_vswitchd_bridge{name="br-int",type=”netdev"} 1
# HELP ovs_vswitchd_bridge_n_flows Number of flows present on the bridge.
# TYPE ovs_vswitchd_bridge_n_flows gauge
ovs_vswitchd_bridge_n_flows{name="br-phy",type=”netdev"} 5
ovs_vswitchd_bridge_n_flows{name="br-int",type=”netdev"} 10
# HELP ovs_vswitchd_bridge_n_ports Number of ports present on the bridge.
# TYPE ovs_vswitchd_bridge_n_ports gauge
ovs_vswitchd_bridge_n_ports{name="br-phy",type=”netdev"} 1
ovs_vswitchd_bridge_n_ports{name="br-int",type=”netdev"} 2
```

```c
METRICS_SUBSYSTEM(ofproto);

static void
do_foreach_ofproto(metrics_visitor_cb callback,
                   struct metrics_visitor *visitor,
                   struct metrics_node *node,
                   struct metrics_label *labels,
                   size_t n OVS_UNUSED)
{
    struct ofproto *ofproto;

    HMAP_FOR_EACH (ofproto, hmap_node, &all_ofprotos) {
        visitor->it = ofproto;
        if (labels[0].key) {
            labels[0].value = ofproto->name;
            labels[1].value = ofproto->type;
        }
        callback(visitor, node);
    }
}

METRICS_FOREACH(ofproto, foreach_ofproto,
                do_foreach_ofproto, "name", "type");

METRICS_FOREACH(ofproto, foreach_ofproto_nolabel,
                do_foreach_ofproto, NULL);
```

Foreach node

Metrics hierarchy: Node type **Code in ofproto/ofproto-metrics.c**

```
ovs_vswitchd ----------------------  root
└── ofproto  ----------------------  subsystem
    ├── n_bridges ------------------  entries
    ├── foreach_ofproto ------------  collection
    │   └── bridge_entries ---------  entries
    └── foreach_ofproto_nolabel ----  collection
        └── foreach_ports ----------  collection
            ├── port_entries -------  entries
            └── netdev_labeled_info   labels
                └── netdev_info ----  entries
```

```
# HELP ovs_vswitchd_bridge A metric with a constant value '1' labeled by
# TYPE ovs_vswitchd_bridge gauge
ovs_vswitchd_bridge{name="br-phy",type=”netdev"} 1
ovs_vswitchd_bridge{name="br-int",type=”netdev"} 1
# HELP ovs_vswitchd_bridge_n_flows Number of flows present on the bridge.
# TYPE ovs_vswitchd_bridge_n_flows gauge
ovs_vswitchd_bridge_n_flows{name="br-phy",type=”netdev"} 5
ovs_vswitchd_bridge_n_flows{name="br-int",type=”netdev"} 10
# HELP ovs_vswitchd_bridge_n_ports Number of ports present on the bridge.
# TYPE ovs_vswitchd_bridge_n_ports gauge
ovs_vswitchd_bridge_n_ports{name="br-phy",type=”netdev"} 1
ovs_vswitchd_bridge_n_ports{name="br-int",type=”netdev"} 2
```

```c
enum {
    OF_BRIDGE_NAME,
    OF_BRIDGE_N_PORTS,
    OF_BRIDGE_N_FLOWS,
};

static void
bridge_read_value(double *values, void *it)
{
    struct ofproto *ofproto = it;
    struct oftable *table;
    unsigned int n_flows;

    n_flows = 0;
    OFPROTO_FOR_EACH_TABLE (table, ofproto) {
        n_flows += table->n_flows;
    }

    values[OF_BRIDGE_NAME] = 1.0;
    values[OF_BRIDGE_N_PORTS] = hmap_count(&ofproto->ports);
    values[OF_BRIDGE_N_FLOWS] = n_flows;
}

METRICS_ENTRIES(foreach_ofproto, bridge_entries, "bridge", bridge_read_value,
    [OF_BRIDGE_NAME] = METRICS_GAUGE(,
        "A metric with a constant value '1' labeled by bridge name and type "
        "present on the instance."),
    [OF_BRIDGE_N_PORTS] = METRICS_GAUGE(n_ports,
        "Number of ports present on the bridge."),
    [OF_BRIDGE_N_FLOWS] = METRICS_GAUGE(n_flows,
        "Number of flows present on the bridge."),
);
```

Label node

Metrics hierarchy: Node type **Code in ofproto/ofproto-metrics.c**

```
ovs_vswitchd ----------------------  root
└── ofproto  ----------------------  subsystem
    ├── n_bridges ------------------  entries
    ├── foreach_ofproto ------------  collection
    │   └── bridge_entries ---------  entries
    └── foreach_ofproto_nolabel ----  collection
        └── foreach_ports ----------  collection
            ├── port_entries -------  entries
            └── netdev_labeled_info   labels
                └── netdev_info ----  entries
```

```c
static void
do_foreach_ports(metrics_visitor_cb callback,
                 struct metrics_visitor *visitor,
                 struct metrics_node *node,
                 struct metrics_label *labels,
                 size_t n OVS_UNUSED)
{
    struct ofproto *ofproto = visitor->it;
    struct ofport *port;

    HMAP_FOR_EACH (port, hmap_node, &ofproto->ports) {
        labels[0].value = port->ofproto->name;
        labels[1].value = netdev_get_name(port->netdev);
        labels[2].value = netdev_get_type(port->netdev);
        labels[3].value = port->pp.name;
        visitor->it = port;
        callback(visitor, node);
    }
}

METRICS_FOREACH(foreach_ofproto_nolabel, foreach_ports,
                do_foreach_ports, "bridge", "name", "type", "port");
```

```
ovs_vswitchd_interface_info{driver_name="tun",driver_version="1.6",bridge="br-phy",type="tap",port="br-phy"} 1
ovs_vswitchd_interface_info{bridge="br-int",name="vxlan100",type="vxlan",port="vxlan100"} 1
ovs_vswitchd_interface_info{driver_name="tun",driver_version="1.6",bridge="br-int",type="tap",port="br-int"} 1
```

Label node

Metrics hierarchy: Node type **Code in ofproto/ofproto-metrics.c**

ovs_vswitchd ---------- root
└── ofproto ---------- subsystem
  ├── n_bridges ---------- entries
  ├── foreach_ofproto ---------- collection
  │ └── bridge_entries --------- entries
  └── foreach_ofproto_nolabel ---- collection
    └── foreach_ports ---------- collection
      ├── port_entries ------- entries
      └── netdev_labeled_info labels
        └── netdev_info ---- entries

```
ovs_vswitchd_interface_info{driver_name="tun",driver_version="1.6",bridge="br-phy",type="tap",port="br-phy"} 1
ovs_vswitchd_interface_info{bridge="br-int",name="vxlan100",type="vxlan",port="vxlan100"} 1
ovs_vswitchd_interface_info{driver_name="tun",driver_version="1.6",bridge="br-int",type="tap",port="br-int"} 1
```

```c
static void
netdev_info_label(struct metrics_label *labels, size_t n, void *it)
{
    static char values[3][100];
    struct smap netdev_status;
    struct netdev *netdev;
    struct ofport *port;
    size_t i;

    port = it;
    netdev = port->netdev;

    labels[0].value = values[0];
    labels[1].value = values[1];
    labels[2].value = values[2];

    /* By default, reset the labels values to an empty string. */
    for (i = 0; i < n; i++) {
        values[i][0] = '\0';
    }

    smap_init(&netdev_status);
    if (netdev_get_status(netdev, &netdev_status) == 0) {
        for (i = 0; i < n; i++) {
            const char *value = smap_get(&netdev_status, labels[i].key);

            if (value != NULL) {
                snprintf(values[i], sizeof values[i], "%s", value);
            } else {
                values[i][0] = '\0';
            }
        }
    }
    smap_destroy(&netdev_status);
}

METRICS_LABEL(foreach_ports,
    netdev_labeled_info, netdev_info_label,
    "driver_name", "driver_version", "firmware_version");

METRICS_ENTRIES(netdev_labeled_info, netdev_info,
    "interface", metrics_array_read_one,
    METRICS_GAUGE(info,
        "A metric with a constant value '1' labeled with the driver name, "
        "version and firmware version of the interface."),
);
```

Conditional node

```
METRICS_FOREACH(dpif, foreach_dpif, do_foreach_dpif, "datapath");
METRICS_FOREACH(dpif, foreach_dpif_nolabel, do_foreach_dpif, NULL);

static bool
ct_stats_supported(void *dpif)
{
    uint32_t u32;

    return ct_dpif_get_nconns(dpif, &u32) == 0;
}

METRICS_IF(foreach_dpif, if_ct_stats_supported, ct_stats_supported);
```

```
enum {
    CT_DPIF_METRICS_N_CONNECTIONS,
    CT_DPIF_METRICS_CONNECTION_LIMIT,
    CT_DPIF_METRICS_TCP_SEQ_CHK,
};

static void
ct_dpif_read_value(double *values, void *_dpif)
{
    struct dpif *dpif = _dpif;
    bool tcp_seq_chk;
    uint32_t u32;

    ct_dpif_get_nconns(dpif, &u32);
    values[CT_DPIF_METRICS_N_CONNECTIONS] = u32;

    ct_dpif_get_maxconns(dpif, &u32);
    values[CT_DPIF_METRICS_CONNECTION_LIMIT] = u32;

    ct_dpif_get_tcp_seq_chk(dpif, &tcp_seq_chk);
    values[CT_DPIF_METRICS_TCP_SEQ_CHK] = tcp_seq_chk ? 1 : 0;
}

METRICS_ENTRIES(if_ct_stats_supported, ct_dpif_entries,
        "datapath", ct_dpif_read_value,
    [CT_DPIF_METRICS_N_CONNECTIONS] = METRICS_GAUGE(n_connections,
        "Number of tracked connections."),
    [CT_DPIF_METRICS_CONNECTION_LIMIT] = METRICS_GAUGE(connection_limit,
        "Maximum number of connections allowed."),
    [CT_DPIF_METRICS_TCP_SEQ_CHK] = METRICS_GAUGE(tcp_seq_chk,
        "The TCP sequence checking mode: disabled(0) or enabled(1)."),
);
```

Histogram node

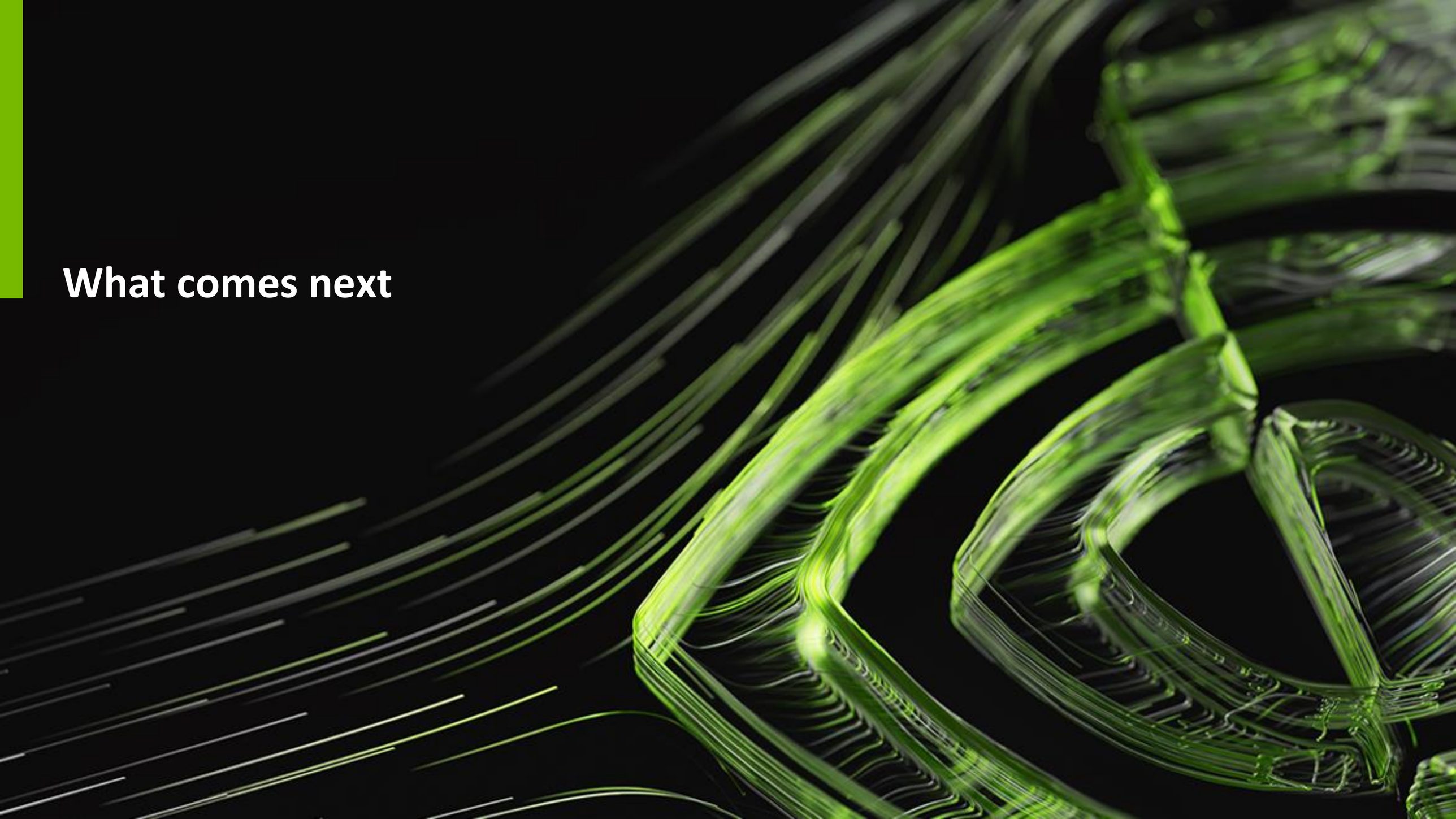
```
ovs_vswitchd_hw_offload_latency_buckets{le="1",thread_num="1",datapath="ovs-netdev"} 1
ovs_vswitchd_hw_offload_latency_buckets{le="2",thread_num="1",datapath="ovs-netdev"} 1
ovs_vswitchd_hw_offload_latency_buckets{le="3",thread_num="1",datapath="ovs-netdev"} 1
ovs_vswitchd_hw_offload_latency_buckets{le="4",thread_num="1",datapath="ovs-netdev"} 1
ovs_vswitchd_hw_offload_latency_buckets{le="5",thread_num="1",datapath="ovs-netdev"} 2
ovs_vswitchd_hw_offload_latency_buckets{le="6",thread_num="1",datapath="ovs-netdev"} 2
ovs_vswitchd_hw_offload_latency_buckets{le="7",thread_num="1",datapath="ovs-netdev"} 2
ovs_vswitchd_hw_offload_latency_buckets{le="8",thread_num="1",datapath="ovs-netdev"} 3
ovs_vswitchd_hw_offload_latency_buckets{le="9",thread_num="1",datapath="ovs-netdev"} 4
ovs_vswitchd_hw_offload_latency_buckets{le="10",thread_num="1",datapath="ovs-netdev"} 4
ovs_vswitchd_hw_offload_latency_buckets{le="12",thread_num="1",datapath="ovs-netdev"} 5
ovs_vswitchd_hw_offload_latency_buckets{le="16",thread_num="1",datapath="ovs-netdev"} 9
ovs_vswitchd_hw_offload_latency_buckets{le="20",thread_num="1",datapath="ovs-netdev"} 62
ovs_vswitchd_hw_offload_latency_buckets{le="26",thread_num="1",datapath="ovs-netdev"} 62
ovs_vswitchd_hw_offload_latency_buckets{le="34",thread_num="1",datapath="ovs-netdev"} 62
ovs_vswitchd_hw_offload_latency_buckets{le="44",thread_num="1",datapath="ovs-netdev"} 66
ovs_vswitchd_hw_offload_latency_buckets{le="57",thread_num="1",datapath="ovs-netdev"} 70
ovs_vswitchd_hw_offload_latency_buckets{le="74",thread_num="1",datapath="ovs-netdev"} 73
ovs_vswitchd_hw_offload_latency_buckets{le="95",thread_num="1",datapath="ovs-netdev"} 73
ovs_vswitchd_hw_offload_latency_buckets{le="123",thread_num="1",datapath="ovs-netdev"} 73
ovs_vswitchd_hw_offload_latency_buckets{le="158",thread_num="1",datapath="ovs-netdev"} 73
ovs_vswitchd_hw_offload_latency_buckets{le="204",thread_num="1",datapath="ovs-netdev"} 73
ovs_vswitchd_hw_offload_latency_buckets{le="263",thread_num="1",datapath="ovs-netdev"} 73
ovs_vswitchd_hw_offload_latency_buckets{le="339",thread_num="1",datapath="ovs-netdev"} 73
ovs_vswitchd_hw_offload_latency_buckets{le="437",thread_num="1",datapath="ovs-netdev"} 74
ovs_vswitchd_hw_offload_latency_buckets{le="563",thread_num="1",datapath="ovs-netdev"} 92
ovs_vswitchd_hw_offload_latency_buckets{le="725",thread_num="1",datapath="ovs-netdev"} 92
ovs_vswitchd_hw_offload_latency_buckets{le="935",thread_num="1",datapath="ovs-netdev"} 92
ovs_vswitchd_hw_offload_latency_buckets{le="1204",thread_num="1",datapath="ovs-netdev"} 92
ovs_vswitchd_hw_offload_latency_buckets{le="1552",thread_num="1",datapath="ovs-netdev"} 92
ovs_vswitchd_hw_offload_latency_buckets{le="2000",thread_num="1",datapath="ovs-netdev"} 92
ovs_vswitchd_hw_offload_latency_buckets{le="+Inf",thread_num="1",datapath="ovs-netdev"} 92
ovs_vswitchd_hw_offload_latency_sum{thread_num="1",datapath="ovs-netdev"} 11431
ovs_vswitchd_hw_offload_latency_count{thread_num="1",datapath="ovs-netdev"} 92
```

```c
static struct histogram *
hw_offload_latency_get(void *_it)
{
    struct hw_offload_it *it = _it;

    return &it->thread->latency;
}

METRICS_HISTOGRAM(foreach_hw_offload_threads_dbg, hw_offload_latency,
                  "Latency in milliseconds between an offload request and its "
                  "completion.", hw_offload_latency_get);
```

Conclusion

- Metrics is a framework describing **where** to find the data:
  - Iterating containers / collections
  - Only if enabled / in-use
  - Decorate with labels
  - Read counters or gauges
  - Provide pointer to histograms
- Each metrics nodes provide functions to be interpreted:
  - Boolean for conditionals
  - Recursion for collections
  - Reads for entries
- Once a set of metrics entries is defined and registered,
  it is automatically added to the metrics output and accessed by all consumers.
- All reads are made from main thread. Access safety is aligned on existing patterns (locking, RCU, etc).

What comes next



What comes next

Phase 1:

- Series submitted, requesting comments.
  https://mail.openvswitch.org/pipermail/ovs-dev/2025-November/427946.html
  http://patchwork.ozlabs.org/project/openvswitch/list/?series=482574
- OVS project can register a default prometheus HTTP port to ease deployment.

Phase 2:

- Additional prometheus support
  - Type “Summary” (if needed, streaming approximation might work)
  - Type “Info” (to replace labeled ‘gauges’ used to convey text info)
  - Timestamps
- Additional local dashboards
  - Currently: oriented toward HW-offloads.
  - Additionally: SW-performance (PMD caches)
  - Resources: CPU, memory, flows, CT, etc


NVIDIA