Flow Exporter in Antrea Agent

1. Objective

Network visibility into the kubernetes cluster provides benefits such as network management, troubleshooting, compliance of network policies etc. It can help answer the questions such as What are the networking policies employed between two services?, What L4 port has the highest traffic on a node?, What nodes are communicating the most (Tx/Rx bandwidth)?, How many connections have only

TCP_SYN packet sent (TCP_SYN flood) etc. Network flows in each node of the kubernetes cluster are one of the main building blocks to achieve network visibility.

This proposal discusses how to gather network flows in each node and export using IPFIX protocol. In addition, gather flow metrics and export as prometheus metrics. Exported flows can be processed at a collector that supports IPFIX. We recommend Elastiflow as the collector, which provides Kibana based UI to visualize flows: https://github.com/robcowart/elastiflow

2. User Stories

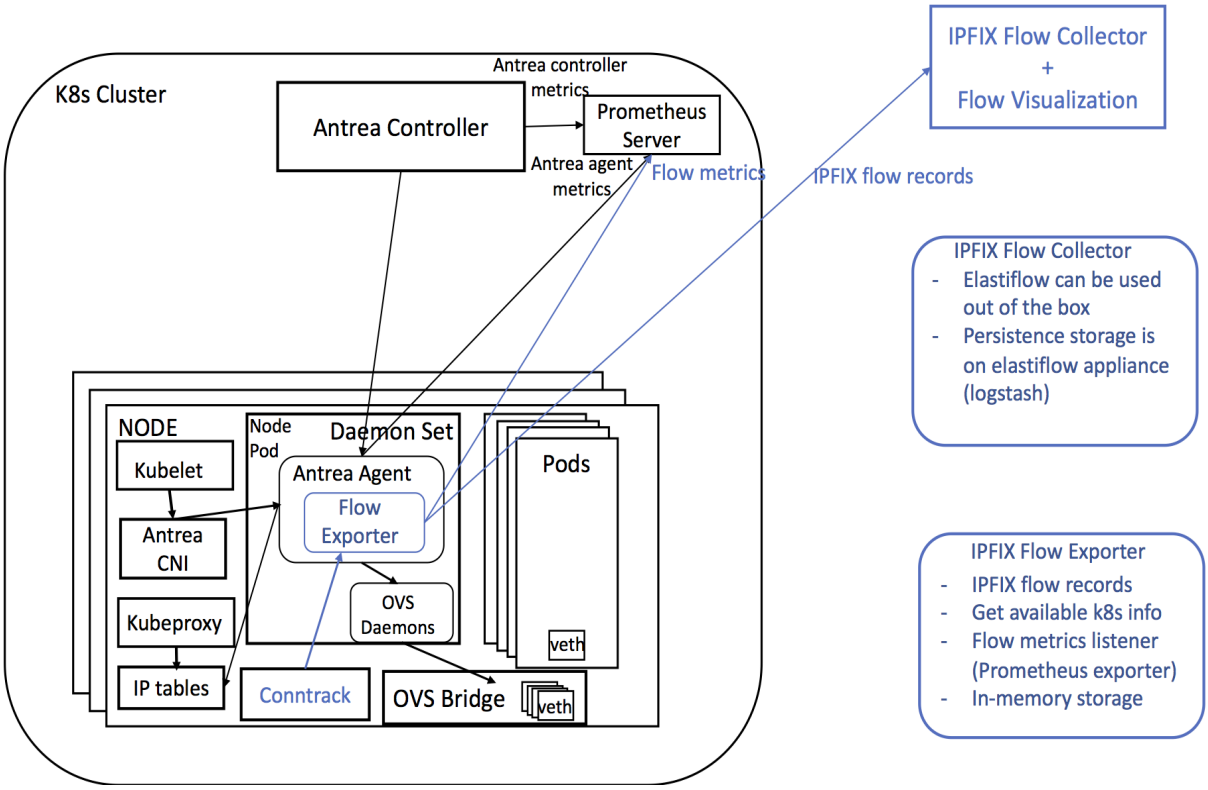
2.1. Flow Visibility UI (Elastiflow)

- Show flows between various pods, pod namespaces etc.
    - Add filtering for L4 proto + L4 destination ports in pod flows
- Show throughputs in each node
- Hot spots in node-to-node or pod-to-pod communications through connections, packets and byte per second metrics in flows

2.2. Flow metrics (Prometheus)

- Flow metrics in a node
    a. connections per node, connections per pod
    b. Track packet, bytes in addition to connections wherever relevant
    c. L4 proto+destination ports connection distribution in a node.
- Flow metrics based on connection tracking state in a node
    a. Flows with SYN_SENT
    b. Flows with TCP RST flag–how to identify in conntrack flows?
- Flow metrics based on network policy names/UUIDs in a node
    - Accepted and denied connections per second

3. Design



3.1. Network flows from OVS dataplane

We discuss a few possible sources for network flows from OVS dataplane and our choice in the end.

3.1.1. OVS Conntrack flows

- Conntrack flows with OVS can be dumped by command "ovs-appctl dpctl/dump-conntrack" or using conntrack utility.
  Example of dump flows:

```
stati@ubuntu:~/ovs_scripts$ sudo conntrack -L -o timestamp
[sudo] password for stati:

tcp,orig=(src=192.168.86.78,dst=192.168.86.78,sport=51476,dport
=6443,packets=1774,bytes=110696),reply=(src=192.168.86.78,dst=1
92.168.86.78,sport=6443,dport=51476,packets=1694,bytes=101266),
timeout=61757,protoinfo=(state=ESTABLISHED)
tcp,orig=(src=100.10.10.1,dst=100.10.10.16,sport=54088,dport=81
```

```
81,packets=5,bytes=387),reply=(src=100.10.10.16,dst=100.10.10.1
,sport=8181,dport=54088,packets=5,bytes=405),timeout=113,protoi
nfo=(state=TIME_WAIT)
```

- Use conntrack zone id as filter to get OVS bridge "br-int" related flows.
- For Antrea, we can use existing Go libraries that provides interface to conntrack module such as: https://godoc.org/github.com/ti-mo/conntrack
  These libraries are similar to popular C library:
  https://www.netfilter.org/projects/libnetfilter_conntrack/
- Conntrack flows have TCP connection info such as connection state, and they maintain counters in reverse direction (“reply”)

3.1.2. OVS IPFIX

- OVS bridge IPFIX configuration info from ovs-vsctl man page:
  http://www.openvswitch.org/support/dist-docs/ovs-vsctl.8.txt
- One direction flow records are only exported. It can track the number of packets and bytes in each flow. As a result, IPFIX does not provide visibility into flow in reverse directions and connection state such as retransmissions etc.
- OVS IPFIX incurs high overhead with no sampling (refs are given below). With higher sampling accuracy drops.
  From Riverbed community page: https://community.riverbed.com/s/article/DOC-5783 (With higher sampling, there is a comment on accuracy–check considerations section)

3.1.3. OVS sFlow

- sFlow agent runs on each node to forward filtered packets based on sampling rate to sFlow collector. Multiple nodes can forward to sFlow collector, where packets are analyzed and flow records are created for each node at sFlow collector. There is VXLAN/Geneve/GRE tunnel support.
- sFlow collector can also keep track of OVS interface metrics as well. In our case, these can be POD interface metrics.
- With sFlow, we can parse till L7. This could be useful to get http related statistics.

3.1.4. Our choice

We pick conntrack flow as the source for network flows. Reasons are following:

- No overhead on the dataplane. No issues with sampling/accuracy of flow information compared to OVS IPFIX/OVS sFlow
- Conntrack flows provide state of reverse flow (counters like packets/bytes), TCP connection state etc.
- Conntrack flows can provide information on network policy rules
- Other communication channels could be used as alternatives to IPFIX. We prefer IPFIX communication channel as many collectors support it.

3.2. Flow Exporter Functionality/Features

Flow exporter runs as part of Antrea agent container with a config option in config map to enable or disable the exporter. Flow exporter polls the conntrack flows periodically (say 5s to 10s--configurable). Flow exporter exports IPFIX flow records periodically (say 5-10mins--configurable) to an IPFIX flow collector. IPFIX flow collector configuration (IP+port) is provided through config map.

If there are any resource consumption issues because flow exporter is in the same container as Antrea Agent, then the flow exporter can be moved to a different container that could be part of antrea agent pod.

3.2.1. Exporting IPFIX flow records

- Flow exporter converts these flows into IPFIX flow records.
  Example of a IPFIX flow record template with IPFIX information elements.

IPFIX Information Element	Enterprise ID	Field ID	Type
octetDeltaCount	0(IANA)	1	uint64
packetDeltaCount	0(IANA)	2	uint64
flowStartSeconds	0(IANA)	150	uint32
flowEndSeconds	0(IANA)	151	uint32
sourceIPv4Address	0(IANA)	8	uint32

destinationIPv4Address	0(IANA)	12	uint32
sourceTransportPort	0(IANA)	7	uint16
destinationTransportPort	0(IANA)	11	uint16
protocolIdentifier	0(IANA)	4	uint8
packetTotalCount	0(IANA)	86	uint64
octetTotalCount	0(IANA)	85	uint64
sourcePodNamespace	12399(antrea)	176	string
sourcePodName	12399(antrea)	177	string
destinationPodNamespace	12399(antrea)	178	string
destinationPodName	12399(antrea)	179	string
reversePacketDeltaCount	29305(Reverse ent ID)	180	uint64
reverseOctetDeltaCount	29305(Reverse ent ID)	181	uint64
reversePacketTotalCount	29305(Reverse ent ID)	182	uint64
reverseOctetTotalCount	29305(Reverse ent ID)	183	uint64
sourceNodeName	12399(antrea)	184	string

destinationNodeName	12399(antrea)	185	string

- There is one public library to encode flows into IPFIX flow records: https://github.com/CN-TU/go-ipfix. However, this library is old and not maintained well. In addition, it does not have the capability to intercept IPFIX records and enhance them. We may need this feature in future work. We are writing our own library to export IPFIX flow records in Go.
- Flow exporter aggregates the flows for a given time period, converts to IPFIX flow record and exports to a IPFIX collector. This can be in the range of 5mins to 10mins (configurable)
  Flow exporter can also send to a cluster flow aggregator where all the k8s context info could be added. This cluster flow aggregator can send to IPFIX flow collector
- Possible optimization: For efficiency, flow exporter gets only dropped (invalid state) connections, closed connections and other than established connections every poll. Right before exporting to the Collector, flow exporter gets active flows with cumulative stats from kernel. This way we can save cost on flow aggregation in the exporter.

3.2.2. Add available k8s resource info at Agent

- Add available Kubernetes resource info from the interface store of agentQuerier. This will be either src IP or dest IP, which can be mapped to a local pod name and namespace.
- Possibility: Get network policy UUID from conntrack flow. An idea is to embed the UUID in OVS flow using ct_label when installing network policy rule. Another idea is to use conjunct ID/rule ID in ct_label, and add another cache map in agentQuerier to retrieve rule name/network policy UUID/ network policy name.
- Future work: K8s cluster IP is available in service to service flows. This is same as the destination IP of flow in the original direction originating from a pod. This will be used to do service mapping for flows

3.2.3. Tracking and exposing flow metrics

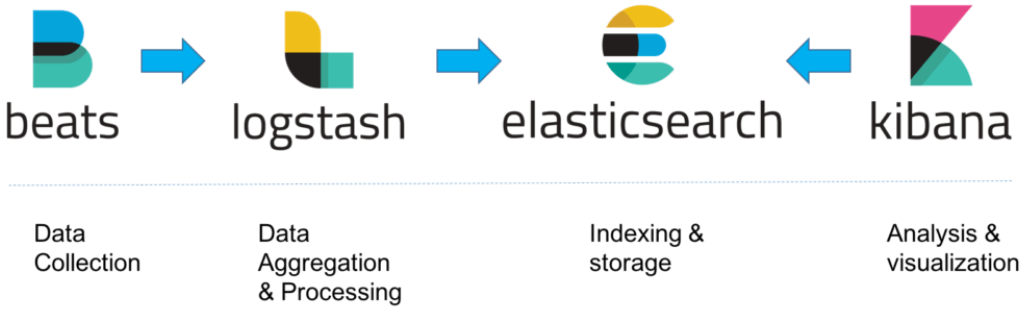
- Flow metrics such as number of connections per node or per pod, number of connections hitting network policy accept/denied, number of connections that are in SYN_SENT, terminated through RST flag etc.
- Classify flow metric types for easy configuration through yml manifest. E.g., TCP state/pod centric/network policy based etc.
- Flow metrics are processed and aggregated after every poll of flow exporter
- Flow metrics are exposed through Prometheus exporter. This can be a different listener from Antrea agent/controller metrics.

3.2.4. Connection map in a node

- In-memory storage for connection map. IPFIX flow records are created out of the connection map. Memory is flushed after every export to the IPFIX collector. No persistence.
- As an alternative, we could also flush the connections from connection map when they are not present in polled output with a timeout value say 60s.
- Limit the number of connections in connection map for memory scalability issues

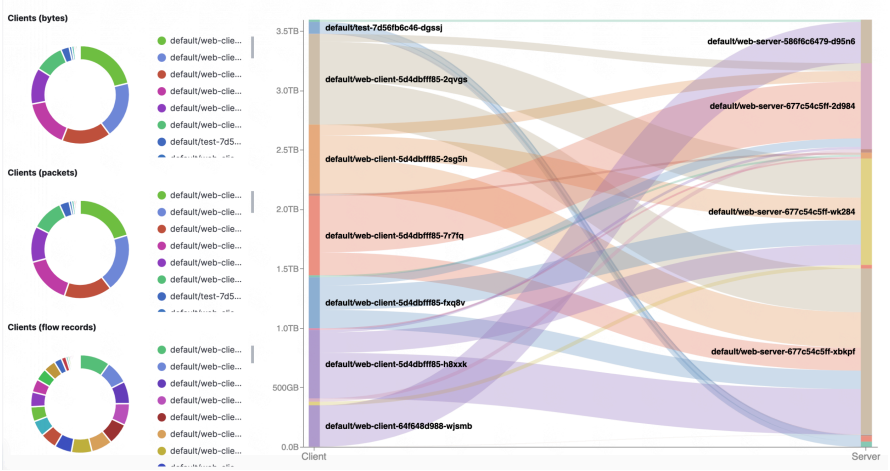
3.3. Integration with Elastiflow

Elastiflow provides network flow data collection and visualization using the Elastic Stack (Elasticsearch, Logstash, Kibana) which supports Netflow v5/v9, sFlow and IPFIX flow types.



Flow Visibility:

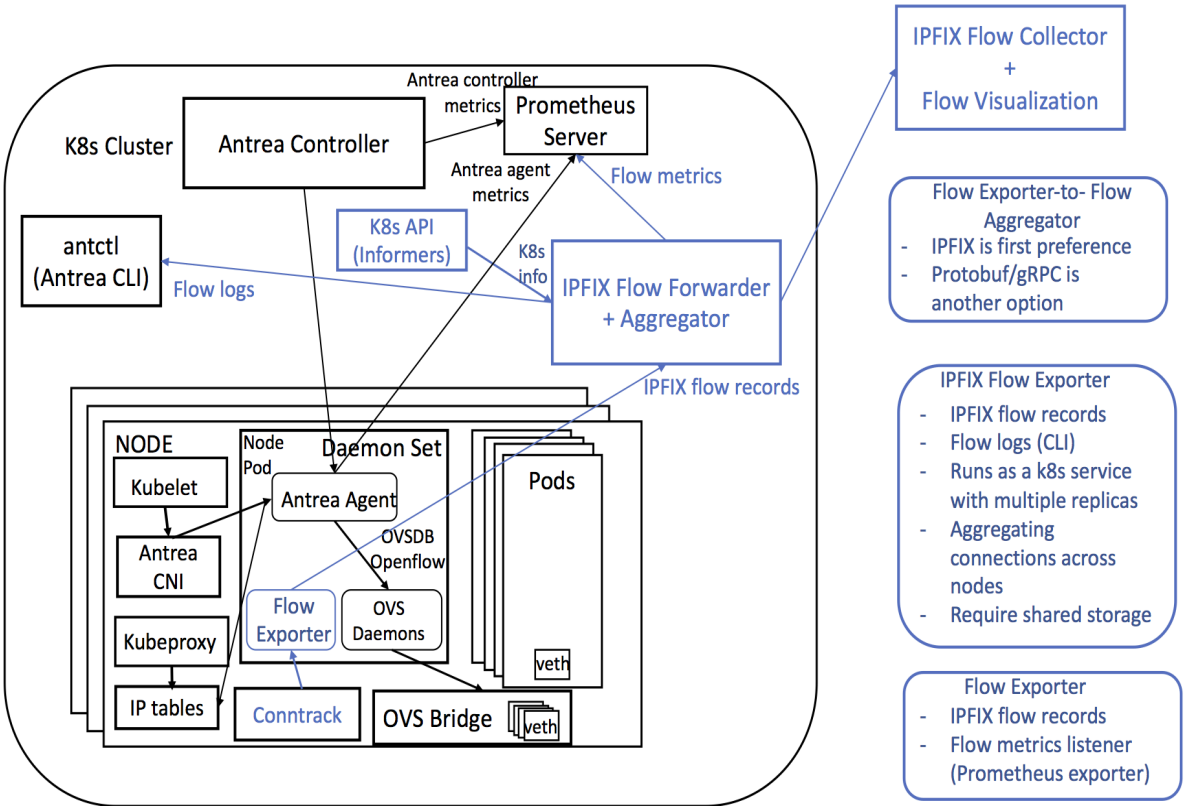
- As an analytics and visualization platform, Kibana provides versatile dashboards such as sankey diagrams to visualize the network flow. Elastiflow also has pre-built dashboard that can visualize flow exporters, traffic details https://github.com/robcowart/elastiflow
- To add Antrea customized fields in IPFIX flow records, Filebeat (lightweight shipper for forwarding and centralizing log data) will be run as DaemonSet on every node to collect data and ship to Logstash
- Sample visualization from PoC



Flow Metrics:

- Prometheus module (https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-module-prometheus.html) in Metricbeat can help with shipping metrics from Prometheus server and send to Logstash for further processing.
- Prometheus module also provides a predefined dashboard in Kibana for us to customize with

4. Future Work



5. Work items

Task	Assigned	Status
Poll connections from conntrack table		

IPFIX library in Golang		
Create IPFIX flow records and send flow records to a collector		
Add local k8s info to the flows		
Add flow metrics through Prometheus		
End-to-end test with local collector		
Configure Logstash scripts		
Implementation of Elastiflow deployment		
Design and build reusable dashboard on Kibana		
Implement and configure Metricbeat for Prometheus		
End-to-end testing with Elastiflow collector		
Documentation and Benchmarking		

Questions from proposal discussion:

1. Pod labels as one of the fields. Is this useful?
2. Flow aggregation across a destination IP/port instead of 5-tuple to aggregate multiple flows. Helpful if transmission rates are high in scale setups
3. Authorization and access levels in Elastiflow to visualize the flows; cluster role and k8s RBAC. Follow up with Cody.
4. Scalability for flows and metrics. Benchmarking will be done to see how the solution is scaling. Following are possible enhancements to manage the scale:
    - With flow aggregator, instead of scraping the metrics from Antrea agent on every node. We could expose all the metrics from the flow aggregator.

- If transmission bandwidth is high for IPFIX, we could aggregate flow records across minimal dimensions rather than 5-tuples. In addition, we could use protobuf as a communication channel between antrea agent and flow aggregator instead of IPFIX, which provides stateless flow updates, that is we can update a subset of fields for a flow rather than sending all fields multiple times.