Why?

- SDN is complex
- Highly dynamic
- Lack of open source tooling for troubleshooting
Goals

- SDN agnostic
- Real-time / post-mortem network analysis framework
- Lightweight, easy to deploy
Overview

- Distributed
- Single binary
- Agents
  - Capture topology and flows
  - Forwards to the analyzers
- Analyzers
  - Aggregate and store topology and flows
  - Serve API
• OVS objects
  - bridge, port, interfaces
  - using OVSDB

• Network objects
  - Interfaces, bridges, bonds, VLAN, ...
  - Properties (MAC, IP, MTU, Statistics, ...)
  - Network namespaces
  - Static objects
Topology probes

- External connectors
  - Docker
  - OpenStack Neutron
  - OpenContrail

- Create a graph:
  - Nodes: interfaces, network objects with metadata
  - Links: L2, ownership, membership, ...
Topology query

- Graph engine
- Event based
  - Graph listener through WebSocket (agents, Web UI, your software)
- Gremlin like query language
- Full history
Topological query example:

```bash
$ skydive client topology query -q 'G.V().Has("Type", "ovsbridge").Out().Out().Has("Name", Without("br-int")){ "Host": "localhost.localdomain", "ID": "a190409e-f76e-4c8f-55b9-985e662a37c0", "Metadata": { "Driver": "veth", "IfIndex": 168, "MAC": "3e:88:b9:65:04:7e", "MTU": 1500, "Name": "vm1-eth0", "State": "UP", "Type": "veth", "UUID": "b6e9bf79-9b58-4b65-800e-1ddf9909d9dc" }}]
```
Topology probes

- 2 VMS with the Skydive agent
- On each VM
  - 2 physical interfaces connected to a TOR
  - A network namespace
  - A pair of veth
  - Connected to an OVS bridge « br-int »
  - A GRE tunnel between the nodes
What we call a flow

- Layers:
  - Link, Network, Transport
- Metrics (packets, bytes)
- Source and destination
- ID, Tracking ID
- Encapsulation support (GRE, VXLAN, MPLS)
Flow capture

- Captures
  - OVS (sFlow)
  - AFPackets
  - libpcap
  - eBPF
  - NDPI
Flows

• Defined capture using the Skydive API
• Traffic is captured on the agent
• Stored into a local flow table
• Push metrics about live and updated flows to the analyzer
• Map endpoints to known interfaces
• Stored into database
• Still the same Gremlin language
  ... and the history

• Examples of Gremlin queries
  
  - g.Flows().Has('TrackingID', '123').Hops()
  - g.Flows().Has('Network.A', '192.168.0.1').Hops()
  - g.Context("An hour ago").V().Has('Name', 'br-int').Flows().Count()}
Use cases

• Validation
• Troubleshooting
• Detection network issues
  – Packet loss
  – Fragmentation
  – Bad performance, congestion points
• Post mortem analysis
Use cases

• Monitoring
  – Grafana plugin
  – Alarming
• Capacity planning
  – Schedule services at the best place
• Billing
Flow demo

- Same topology than the previous demo
- GRE tunnels
- Create capture points
- Generate traffic
- Follow traffic in the tunnel
- Skydive analyzer on my laptop with Elasticsearch
Flow demo

- Same topology than the previous demo
- GRE tunnels
- Create capture points
- Generate traffic
- Follow traffic in the tunnel
- Skydive analyzer on my laptop with Elasticsearch
Quickstart

• Executable
  – # skydive allinone
• Docker
  – docker-compose up
• Kubernetes
  – kubectl create -f contrib/kubernetes/skydive.yaml
• Devstack plugin
Community

- Apache License
- https://github.com/skydive-project/skydive
- Written in Go
- (Good?) Documentation
Questions?

- IRC: #skydive-project @ freenode.net
- Mailing list: skydive-dev@redhat.com