Scientific Network Tags: Packet and Flow Marking

scitags.org

Marian Babik (CERN), Shawn McKee (Univ. of Michigan)
net-wg@cern.ch | www.scitags.org
On behalf of the Research Networking Technical Working Group
GridPP Technical Meeting
History

- **HEPiX Network Functions Virtualisation Working Group**
  - Working Group Report was published at the end of 2019 with three chapters
    - Could Native DC Networking
    - Programmable Wide Area Networks
    - Proposed Areas of Future Work

- **LHCOPN/LHCONE workshop** (spring 2020)
  - Requirements on networks from the WLCG experiments

- **Research Networking Technical Working Group**
  - Formed after the workshop in response to the requirements discussion
  - 95 members from ~ 50 organisations have joined
  - Three main areas of work:
    - Packet and Flow Marking - viewed as the appropriate first step; regular meetings every ~2 months since summer 2020
      - Packet Marking Document
        - Outlines available technologies, standards and stakeholders perspectives
        - This has led to Scientific Network Tags (scitags) initiative, which is presented today
      - Traffic Shaping
      - Network Orchestration - followed up by GNA-G, SENSE and FABRIC
Motivation

- Networks are becoming more programmable and capable with technologies such as P4, SDN, virtualisation, eBPF, etc.
- But with less and less context about the traffic they carry
  - Cloud deployments, Kubernetes, encryption, tunneling, privacy, etc.
- Understanding scientific traffic flows in detail is critical for understanding how our complex systems are actually using the network.
  - Current monitoring/logging tell us where data flows start and end, but is unable to understand the data in flight.
  - Dedicated L3VPNs can be created to track high throughput science domains, but with more domains requiring high throughput this will become expensive, it won’t scale, won’t work at big sites having to support multiple domains at the same time
- In general the monitoring we have is experiment specific and very difficult to correlate with what is happening in the network. We suggest this is a general problem for users of the Research and Education Networks (RENs)
Scientific Network Tags (scitags) is an initiative promoting identification of the science domains and their high-level activities at the network level.
How scitags work

1. Request 3rd party transfer (src, dst, auth token, etc.)

2. File transfers (src, dst)

Data Management System

Storage (src)  ---  Storage (dst)
How scitags work

1. Request 3rd party transfer (src, dst, auth token, etc.)
   experiment, activity

2. File transfers (src, dst) + experiment, activity

3. UDP firefly is sent for each network flow with payload including
   (src, dst, ports of the file transfer + experiment, activity)
How scitags work

1. Request 3rd party transfer (src, dst, auth token, etc.)
   experiment, activity

2. File transfers (src, dst) + experiment, activity

3. UDP firefly is sent for each network flow with payload including
   (src, dst, ports of the file transfer + experiment, activity)

4. R&E collects file transfer headers and/or UDP fireflies

5. R&E correlates this data with the existing netflow information

6. R&E provides feedback to the DMS

Data Management System

R&E Storage & Analytics

Storage (src)

Storage (dst)
Scitags Architecture

FTS

Rucio

DIRAC

Alice

Storage & Analytics (ELK)

Extension of the FTS API

TCP IPv6 headers

UDP firefly

dCache, Xrootd

R&E collector

R&E collector

Registry (experiments, activities catalogue)
Concepts

- Marking is based on two different approaches
  - **Flow marking** using UDP fireflies (works for both IPv4 and IPv6)
  - **Packet marking** using IPv6 flow label and/or header extensions
- Both carry flow identifier, which at present is an encoded representation of experiment/science domain and activity
  - For UDP fireflies flow id can be extended with other fields in the future
  - For packet marking the space is restricted due to number of bits available in the headers
- Experiments and activities need to be registered prior to their usage
  - Registry serves this purpose and ensures RENs and DDMs have consistent view
- Designed to work with proxies, cached proxies and private networks
- Generators, collectors, storage and analytics can evolve independently
Recent Updates

- New domain and web site ([www.scitags.org](http://www.scitags.org))
- New github organisation ([https://github.com/scitags](https://github.com/scitags))
  - Serves [www.scitags.org](http://www.scitags.org) via github pages
- Flow and Packet Marking [Technical Specification](#)
- Implementation
  - Flow service (flowd, [https://github.com/scitags/flowd](https://github.com/scitags/flowd))
  - Initial implementation in [Xrootd](#)
- Participation in the Data Challenge
Technical Specification Updates

- **Content**
  - Packet and Flow Marking Definitions
  - Flow Service
  - Flow Identifier Lifecycle
    - Provides overview of the expected functionality from each storage/transfer component
    - Proposes extension to Xroot and HTTP TPC protocols
  - Prototype Implementation Plan

- **Protocols updates**
  - Xroot protocol extension with `<scitag.flow>` attribute to pass flow identifier as part of the URL
  - HTTP TPC protocol extension (passing flow identifier as part of the HTTP headers)

- **UDP firefly packet specification**
  - Payload is a syslog message that conforms to RFC5424
    - Last part of the syslog message is a structured data specification (in JSON)
    - JSON schema for the structured data is also available

- **Flow registry specification**
  - Maps experiments and activities to IDs
  - Draft JSON schema, which is already used in the API
  - [https://www.scitaqs.org/api.json](https://www.scitaqs.org/api.json)
Implementation

- **Flow service (flowd)** - developed to help test and validate the approach
  - Provides reference implementation of the technical specification
  - Storage systems can either provide their own implementation or use flowd
  - Written in python, runs as Linux service (integrates with systemd/journal, supports CC8/C8/docker)

- Provides **pluggable** system to test different flow/packet marking strategies.
  - Currently supports flow marking (UDP fireflies) via sampling plugin (netstat) or storage API
  - Sampling plugin using netlink instead of netstat is also in development
    - Can provide additional information per connection (TCP cong. algo, RTT/RTO, CWND, bytes sent/rcvd)
  - Possibility to combine storage API to mark start/end flow and sampling plugin to add additional information
    - This might be needed for storages that don’t have access to the underlying socket interface

- **XRoot 5.4.0 release**
  - Full implementation of the UDP firefly spec (marks start and end of each flow)
  - UDPs fireflies are sent to a dedicated endpoint
  - Supports different options to detect flow identifiers (both experiments and activities)
  - Connects to flow registry API

- Initial implementation of packet marking in XRoot also exists but requires further testing
WLCG Data Challenge

● Aim was to test and validate our approach in gradual steps, our initial goals:
  ○ Test flow service deployment directly on the site’s storages (done)
  ○ Generate UDP fireflies based on real traffic (done)
  ○ Capture UDP packets (initially using a dedicated endpoint) (done)
  ○ Understand how UDP firefly information can be correlated with R&E netflow data (on-going)

● Flow service (flowd) deployment
  ○ Currently deployed at AGLT2, BNL, KIT, UNL and Caltech
  ○ Runs directly on the storage nodes, uses netstat plugin
  ○ Generates UDP fireflies based on real traffic

● ESnet has setup a dedicated collector to capture the UDP fireflies
  ○ Will attempt to correlate them with their netflow data

● Results
  ○ Deployment, packet generation and collection worked fine
  ○ On-going - summary/results on the correlation with netflow
Plans

● Near-term objectives
  ○ Finalise validation and get feedback from ESnet correlation exercise
  ○ Extend testing to Xrootd using dedicated R&E collection endpoint(s) and partial-marking
    ■ Detect flow identifiers from storage path/url, activities from user role mapping
    ■ Test proxies, cached proxies, private networks (K8s)
  ○ Involve other storage systems (dCache, etc.); discuss possible design/implementation
  ○ Instrument Rucio/FTS to pass flow identifiers to the storages

● Continue with the validation and testing using the existing deployment
  ○ Improve existing prototypes based on the feedback from the initial DC tests

● Engage other R&Es and explore available technologies for collectors
  ○ Deploy additional collectors and perform R&D in the packet collectors
  ○ Improve existing data collection and analytics

● Test and validate ways to propagate flow identifiers
  ○ Engage experiments and data management systems
  ○ Validate, test protocol extensions and FTS integration
  ○ Explore other possibilities for flow identifier propagation, e.g. tokens

● R&D activities
  ○ Packet marking - further testing and validation is required for IPv6 flow label implementation.
  ○ Packet collectors - currently UDP fireflies are sent to a dedicated collector(s). R&D is needed to understand how to run generic collectors (that would capture UDP fireflies from real traffic).
Questions, comments?

Draft Technical Specification available; Packet Marking Overview

Prototype testing as part of the WLCG Data Challenges effort in collaboration with ESnet

Prototype code of the flow service (flowd) implementing UDP fireflies

https://www.scitags.org
Backup slides
IPv6 header

| Offsets | Octet | Bit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---------|-------|-----|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Octet   |       |     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0       | 0     | 0   | 0 |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 4       | 32    | 32  |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 8       | 64    | 64  |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 12      | 96    | 96  |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 16      | 128   | 128 |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 20      | 160   | 160 |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 24      | 192   | 192 |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 28      | 224   | 224 |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 32      | 256   | 256 |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 36      | 288   | 288 |   |   |   |   |   |   |   |   |   |   | 15 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

For more details and discussion of various trade-offs please refer to the [Packet Marking Document](#).
IPv6 Ext. headers: Dst Option

The Destination Options header is used to carry optional information that need be examined only by a packet's destination node(s)

- Allocated as one or more blocks of 8 octets; options are TLV encoded

Can be set/changed using standard socket interface (IPV6_DSTOPTS), but requires the options to be built first

- This can be done using standard ancillary data functions

Reading options is performed via socket interface (IPV6_2292PKTOPTIONS)

### Hop-by-Hop Options and Destination Options extension header format

<table>
<thead>
<tr>
<th>Octet</th>
<th>Offset</th>
<th>Bit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>32</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# IPv6 Flow Label

## RFCs (10 hits)

<table>
<thead>
<tr>
<th>RFC</th>
<th>Title</th>
<th>Year</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1809</td>
<td>Using the Flow Label Field in IPv6</td>
<td>1995-06</td>
<td>Informational RFC</td>
</tr>
<tr>
<td>3595</td>
<td>Textual Conventions for IPv6 Flow Label</td>
<td>2003-09</td>
<td>Proposed Standard RFC</td>
</tr>
<tr>
<td>6294</td>
<td>Survey of Proposed Use Cases for the IPv6 Flow Label</td>
<td>2011-06</td>
<td>Informational RFC</td>
</tr>
<tr>
<td>6436</td>
<td>Rationale for Update to the IPv6 Flow Label Specification</td>
<td>2011-11</td>
<td>Informational RFC</td>
</tr>
</tbody>
</table>

## Active Internet-Drafts (2 hits)

<table>
<thead>
<tr>
<th>Draft</th>
<th>Date</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>draft-filsfils-6man-structured-flow-label-00</td>
<td>2021-03-16</td>
<td>I-D Exists</td>
</tr>
<tr>
<td>Structured Flow Label</td>
<td>12 pages</td>
<td>New</td>
</tr>
</tbody>
</table>
Flow Label in Linux Kernel

- Ways to implement:
  - Advanced socket interface
    - Native socket interface, uses kernel network subsystem directly
    - Comes with limitations due to the complexity of the network stack
  - eBPF (XDP, TC-BPF)
    - Sandbox programs running via JIT directly in Linux Kernel
  - Netfilter
    - Kernel module using netfilter subsystem/hooks
  - DPDK, VPP - vendor-specific technologies
  - Software switches (Open vSwitch) - requires OpenFlow
  - SmartNICs (via P4, etc.)
    - Requires dedicated HW, but can be very useful for analytics
## Linux Flow Label Implementation Status

<table>
<thead>
<tr>
<th>OS/Kernel</th>
<th>Flow Label Socket Interface</th>
<th>Netfilter</th>
<th>TC-BPF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flow UDP client server</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flow TCP client</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flow TCP server</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remote flow read</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flow label change on client</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC7 (3.10)</td>
<td>client only</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td>--</td>
<td>ok</td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>C8 (4.15)</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td>5.8</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
<tr>
<td></td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
</tr>
</tbody>
</table>