Flow and Packet Marking Technical Specification

This document outlines detailed technical specifications for the implementation of the flow and packet marking mechanisms. It's based on the reference document on packet marking written by the Research Networking Technical Working Group. It also includes protocol extension proposals written by the XRoot and dCache teams.

Changes	
22/03/2022 Marian	App. A: UDP Firefly Schema - Updated JSON schema with new fields following WG meeting decisions (added usage and netlink as optional attributes; changed current-timestamp to optional; added examples).
12/11/2022 Marian	Updated packet marking specification (corrected the bit positions for different fields).
05/06/2023 Marian	Updated HTTP-TPC specification (changed to numeric codes)
04/07/2023 Marian	Updated HTTP-TPC specification (new simplified version)
25/09/2024 Marian	Updated HTTP-TPC specification (added examples in the processing expectations)

1. Packet and Flow Marking Definitions

There are two possible ways to mark R&E traffic, packet marking which uses a particular label in each packet or flow marking, which identifies a particular network flow by using a separate channel. For the purposes of this document network flow is defined as a five tuple, i.e. source, destination, source port, destination port and flow identifier (this is to allow multiple flows btw. same src, dest, src port, dest port). The intention of this document is to describe a system that can support both flow and packet marking.

The current **flow identifier** has the following two fields:

- **Experiment/virtual organisation** (required) - indicates experiment/virtual organisation, which has initiated the network flow. The mapping between experiment name and its value is defined in advance (and should be mostly static as frequent changes are not anticipated).
- **Experiment activity** (optional) - indicates experiment activity which has initiated the network flow. In the absence of experiment activity only experiment is encoded. Activity must not contain any personal data (any information relating to an identified or identifiable natural person). The mapping between experiment activity and its value is defined in advance (and should be mostly static as frequent changes are not anticipated).

The mapping that defines which experiment/activity has which value/id is defined by a static allocation agreed by the stakeholders. The distribution of the mappings will be initially provided by a **flow registry** served from *scitags.org* domain. The content will be available in JSON format with schema conforming to the one described in Appendix B. Later on a more complex system for broader distribution is foreseen (possibly based on DNS or some other hierarchical distribution model).

1.1 Flow Marking (UDP firefly)

UDP firefly is a mechanism to identify network flow by issuing a specific UDP packet in a separate channel to the actual TCP or UDP transfer (original flow). UDP firefly packet has the following characteristics and fields:

- A specific UDP packet is sent at the beginning and at the end of each original network flow. Optionally in regular intervals in between with at least 60 seconds granularity.
- The packet has the same source and destination IPs as the original flow for both IPv4 and IPv6.
- It uses a specific destination port (10514)
- It contains the following payload:
    - It uses syslog facility header with severity: Informational: 6 and facility: Local0: 16
    - It contains flow identifier and other fields in json format that conforms to the JSON firefly schema shown in Appendix A.
- It uses UTF-8 encoding for the payload.

- UDP firefly payload must fit within a single frame (maximum size is determined by MTU, which for the purpose of this document is 1500 bytes including all headers).

1.2 Packet Marking (Flow Label)

Packet marking is using a flow label, which is a 20 bits field in the IPv6 header and therefore will only mark IPv6 traffic.

Fixed header format

Offsets	Octet	0								1								2								3							
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
0	0	*Version*				*Traffic class*								*Flow label*																			

Fig. 1 Part of the IPv6 header showing flow label and position of bits

The flow label has the following characteristics and fields:

- It contains the experiment and activity identifiers encoded in the following way:
    - Activity identifier is encoded in 6 bits in position 24-29 (c.f. Fig. 1)
    - Entropy/random bits are 5 bits in positions 12-13, 23, 30-31, they're set at random per flow
    - Experiment identifier is encoded in 9 bits 14-22 (bits are in reversed order to allow for possible future adjustments)
- Flow label is set on each packet that is issued by the given experiment and/or activity and must follow RFC6437 (random/entropy bits are set only once for the lifetime of the flow).
- In case of absence of both activity and experiment fields, flow label is set using a random number from the range 0x00000 - 0xFFFFF (per flow).

This allocation takes into account the existing technical limits on the IPv6 flow label and restricts the possible range of values for both *experiment* and *activity*. This restriction should be implemented by the flow service or any other service implementing the packet marking mechanism.

The intent is to modify/update only the flow label 20 bits in the first 32 bit header word on IPv6 packets. How this works in practice will need testing. Also, it should be noted that IPv6 headers don't have a checksum which removes that task when updating the flow label.

2. Flow Service

The management of the flow and packet marking mechanism is performed by a standalone service/daemon that runs on the storage hosts and/or clients and is responsible for the mechanics of marking/signaling the flows. In particular it will send UDP firefly packets and optionally set flow labels (if src/dst is IPv6 and flow label is supported). It exposes the following API to the storage systems and clients:

- flow-start
    - inputs: (protocol, src, src_port, dst, dst_port, experiment, activity)
- flow-end

    - inputs: (protocol, src, src_port, dst, dst_port, experiment, activity)
- flow-update (optional)
    - inputs: (protocol, src, src_port, dst, dst_port, experiment, activity)

The possible application facing interface is TBD (there are many possible options: socket, grpc, json-rpc, http/rest interface, rmq/amq).
In both packet and flow marking it is assumed that public source IPs and ports are known. For certain deployments that use private networks, such as Kubernetes, this will require a separate mechanism to discover the public source IPs (TBD, possible options are STUN/TURN packets).

3. Flow Identifier Lifecycle

This section outlines the lifecycle of the **flow identifier**, i.e. how the flow identifier passes through the existing data management chain to reach R&E networks and their analytics.

There are two potential ways for introducing the flow identifier in the system:

1. *Full marking* (default) in which both identifier fields (*experiment/activity*) are registered with the experiment's data management system and passed on to the lower level services until reaching storage, which will mark the actual traffic that the analytical systems can consume at any point along the network path.
2. *Partial marking* (simplified) in which the *experiment field* in the flow identifier is extracted from a token (or by some other heuristics, e.g. directory path) by the storage systems and passed to the traffic without specifying the activity field.

3.1 Experiment Frameworks

The primary role of the experiment’s frameworks and their data management systems is to introduce the flow identifier into the system. For this they will need to have a representation of the flow identifier fields and be able to pass it to the data management services. The interface to pass this information is specific to the data management system used.

Rucio

Rucio currently has both experiment and activity representations and has been already passing this information to the other systems as part of ATLAS Data Carousel project. Extending the existing interface with FTS might be needed and potentially integrating flow service for cases that result in significant transfer activities and are not performed via FTS. Details TBA.

DIRAC

TBA

ALICE O^2

TBA

3.2 File Transfer Services

If there is a middleware service between experiment's data management and storages, such as FTS, its role will be to pass the flow identifier received from the experiment's data management systems to the storages and caches by means of existing protocols (and their extensions as detailed below). An alternative way that can be used in the early prototyping stage can be to add the flow identifier as part of the file metadata structure, which can then be retrieved via REST API.

3.3 Storage Systems and Caches

Storage systems and caches will be responsible for introducing the flow identifier in the network traffic. This will require either interfacing with the flow service or directly implementing UDP firefly and flow label mechanisms. This impacts all the existing storage systems that transfer significant amounts of scientific data, i.e. dCache, XrootD/XCache, EOS, StoRM, CTA and Echo.

3.4 Clients

Storage clients and client libraries (e.g. gfal2) if used to directly initiate transfers will be also responsible for introducing the flow identifier in the network traffic by the same means as storage systems and caches (more details TBA).

3.5 Protocols Extensions

3.5.1 XRoot

In the Xroot (and HTTP) URL, applications that initiate connections (the clients) also include the flow identifier in their connection URL, as CGI strings. For example, the CGI can be in the form of "scitag.flow=<experimentID><<6|<activityID>", where experimentID and activityID conform to the flow identifiers provided by the flow registry (see Appendix B). The CGI can be used:

1. By the client to set its own socket option for flow/packet marking, and pass to the server.
2. By the server to set its socket option.
3. The client also has an option to obtain the flow identifiers from an external source or from its running environment, and add that to the CGI if the CGI isn't presented by the application's high layers.
4. Assuming(3) is possible, ideally, the client would have a way to include packet/flow marking in the login information that is also passed as a CGI string.

Since Xroot protocol allows reusing a connection, it is possible that the second use of the connection may have a different flow identifier. In this unlikely case, the CGI mechanism should still work but may introduce inconsistencies that need to be explored.

- This should also cover cases where additional ports/sockets are opened for data transfer. This requires corresponding logic in the storage software stack.

Use Case Specific Issues

For 3rd party copy (TPC), the CGI mechanism allows the request initiator (FTS/gfal2/xrdcp/curl) to pass the flow identifier to the server side process/thread that actually triggers TPC.

For XRootD proxy (including forwarding proxy), we need to understand whether the packet marking CGI can be propagated to the data source. In the case of an Xcache proxy, due to its shared nature, the packet mark info cannot be architecturally passed to the source. Even if it were possible, it would not accurately reflect the true purposes of all accesses to the same data file. We think this scenario is a secondary, non-dominate case. In fact, Xcache would mark packets as a general caching service that may or may not be tied to a particular experiment. The packet-marking applications table has had a “Cache” application type added in common for all Astro/HEP science domains.

3.5.2 HTTP-TPC

This section provides a mechanism that allows the controlling agent to request a specific flow identifier be used for the data-bearing transfer when making an HTTP-TPC request. Conforming active-party servers would ensure that the TCP connection over which data flows has the requested flow identifier.

The COPY request

In order for the controlling agent to affect the data bearing transfer, it must tell the active party what is the desired flow identifier. This is achieved by specifying an additional HTTP request header in the COPY request. The following guidelines apply for the processing of the HTTP/TPC headers:

The client MAY specify the `SciTag` request header. The content of the SciTag header is a positive integer with an encoded value using the formula: <expID> <<6 | <actID>, where <<6 | are bitwise operators and expID and actID are the numeric codes that map experiments and activities to numbers as found in the SciTags registry API (see Appendix B).

The following is a COPY request that request data is pulled from example.org with the flow identifier atlas (expID:2), rebalancing (actID:16).

```
COPY /path/to/destination HTTP/1.1
Host: destination.example.org
```

```
Source: https://source.example.org/path/to/source
TransferHeaderAuthorization: Bearer ABCD…
SciTag: 144
```

Processing expectation

The active endpoint should record the supplied value along with other details of the desired transfer. When processing the request, the server SHOULD ensure the flow identifier is used to mark the traffic, as received. The following processing rules apply:

- If the active party receives an HTTP-TPC COPY request with no `SciTag` request header then the server SHOULD NOT mark the corresponding network traffic.
- If the active party receives an HTTP-TPC COPY request with a `SciTag` request header then the active party SHOULD include a `SciTag` request header (with the same value as in the COPY request) in all related HTTP requests.
- If the active party receives an HTTP-TPC COPY request with a `SciTag` request header with a valid value then the server SHOULD mark the resulting network traffic with the experiment ID and activity ID encoded in the value.
    - Valid value is a single positive integer > 64 and <65536 (16bit). Any other value is considered invalid.
- If the active party receives an HTTP-TPC COPY request with a `SciTag` request header with an invalid value then the server SHOULD mark the resulting network traffic with the 0 as the experiment ID and the activity ID.
- If the active party receives an HTTP-TPC COPY request with both a `SciTag` request header and a `TransferHeaderSciTag` request header then it SHOULD ignore the `TransferHeaderSciTag` and continue to process the request.

In particular the following processing steps apply when handling particular requests:
Assuming C is a client, A and B are servers (storages).

Pull copy

- C -> A : COPY B to A
    - A sends GET file to B with `SciTag` header
    - B should send the firefly (with firefly content listing src:B/dst:A); C sends scitags header (`TransferHeaderSciTag`) to A in the COPY request; A should pass (`SciTag`) header to B in the GET request
    - If C sends `SciTag` header (`SciTag` instead of `TransferHeaderSciTag`) then A should send the firefly without passing it further to B (but in the firefly it should set src:B/dst:A to correctly flag the source and destination). This part is optional (good to have), but not essential (we won't need it once all major storages support scitags).

Push copy

- C -> A : COPY A to B
    - A sends PUT file to B without `SciTag` header
    - A should send firefly (with content src:A/dst:B); C sends scitags header (`SciTag`) to A in the COPY request, no passing of `SciTag` header to PUT, so B does nothing

Direct get

- C -> A: GET file
    - C sends GET to A with (`SciTag`) header
    - A sends firefly (with src:A/dst:C) as it got GET with scitags header (`SciTag`)

Direct put

- C -> A: PUT file
    - C sends PUT to A with (`SciTag`) header
    - C should send the firefly as C is sending the file, but A has to do it on behalf of C and will do so based on PUT with the scitags header (`SciTag`), so A sends firefly (with src:C/dst:A)
    - This is specific/exceptional behavior, but required, as it's not realistic to have all clients supporting scitags in the near future

3.6 Network Analytics

Network analytics system will ensure that the flow identifier can be consumed and processed in an efficient way in order to generate statistical records and pass them on to the 3rd parties (e.g. back to the experiment's frameworks and data management systems).

Details TBA by R&Es

4. Prototype Implementation Plan

The following can be considered for an early prototype implementation:

- Experiments (CMS & ATLAS) update their use of Rucio to supply flow identifiers when adding a rule.
- Rucio is updated so that a flow identifier may be supplied when adding a rule (in essence, a "why" field).
- FTS REST API is updated to allow the submitter to give an optional flow identifier on the PUT/POST request.
- Flow service implementation with basic UDP firefly mechanism and interface. Alternatively also including a basic prototype of the eBPF/flow label implementation.
- Partial marking by one (or more) storage systems that detects the experiment field from token or by other heuristic and calls the flow service.

- Initial prototype of the network analytics system (capturing UDP fireflies and testing correlations with the TCP traffic)

5. References

Appendix A: UDP Firefly JSON Schema

Firefly packet protocol stack

- IPv4 or IPv6
  - ip-src: IP address of the host launching the firefly packet
  - ip-dst: IP address of the destination of the original flow
  - ip-dst: IP address of the host scitag.es.net (for testing purposes)
  - NOTE: the IP version of the outer IP header is entirely independent from the flow record contained within the JSON-Firefly payload.
- UDP
  - src-port: ephemeral
  - dst-port: 10514
- Syslog
  - Severity: Informational: 6
  - Facility: Local0: 16
- JSON-Firefly schema: see below

The firefly packet should be encoded in compliance with syslog RFC5424 which requires a specific header which contains the following fields:
PRIORITY VERSION TIMESTAMP HOSTNAME APP-NAME PROCID MSGID STRUCTURED-DATA MSG

In our case the following values should be set:
PRIORITY = <134>
VERSION = 1
TIMESTAMP = see RFC5424 for options (e.g. 2021-09-22T11:12:27.808092+00:00) (optional)
HOSTNAME = hostname of the host which originated the packet (optional)
APP-NAME = application name of the application that originated the packet (optional)
PROCID = -
MSGID = firefly-json
STRUCTURED-DATA = -
MSG = For our firefly case the MSG part (see RFC5424) is the JSON described below.

Based on this the following is an example of the entire payload (syslog part is in bold, firefly json is in italics):

<134>1 2021-09-22T11:12:27.808092+00:00 26799cfec63a flowd - firefly-json -
{"flow-id": {"protocol": "tcp", "afi": "ipv4", "dst-ip": "147.213.1.1", "src-port":

54204, "src-ip": "194.12.1.1", "dst-port": 443}, "version": 1, "flow-lifecycle": {"state": "end", "end-time": "2021-09-22T11:12:27.801666+00:00", "start-time": "2021-09-22T11:12:17.776691+00:00", "current-time": "2021-09-22T11:12:27.808210+00:00"}, "context": {"experiment-id": 16, "application": "flowd v0.1.0", "activity-id": 1}}

The PRIORITY field is constructed as 8*FACILITY+SEVERITY. The result is bracketed with '<' and '>' at the front of the packet.

Current schema is at:

https://scitags.github.io/schemas/v1.0.0/firefly.schema.json

Examples of sample UDP firefly JSONs:

```
{
  "version": 1,
  "flow-lifecycle": {
    "state": "start",
    "current-time": "2021-09-07T22:22:21.998202+00:00",
    "start-time": "2021-09-07T22:22:21.998225+00:00"
  },
  "flow-id": {
    "afi": "ipv4",
    "src-ip": "10.99.0.1",
    "dst-ip": "10.100.0.2",
    "protocol": "tcp",
    "src-port": 1234,
    "dst-port": 56789
  },
  "context": {
    "experiment-id": 3333,
    "activity-id": 12,
    "application": "grid-ftp v99.99"
  }
}
```

```
{
  "version": 1,
  "flow-lifecycle": {
    "state": "start",
    "current-time": "2021-09-07T22:22:21.999925+00:00",
    "start-time": "2021-09-07T22:22:21.999933+00:00"
  },
  "flow-id": {
    "afi": "ipv6",
```

```
    "src-ip": "fe80::1",
    "dst-ip": "fe80::2",
    "protocol": "udp",
    "src-port": 1234,
    "dst-port": 56789
  },
  "context": {
    "experiment-id": 3333,
    "activity-id": 12,
    "application": "grid-ftp v99.99"
  }
}

_
{
  "version": 1,
  "flow-lifecycle": {
    "state": "end",
    "current-time": "2021-09-07T22:22:22.000708+00:00",
    "start-time": "2021-09-07T22:22:21.999933+00:00",
    "end-time": "2021-09-07T22:22:22.000690+00:00"
  },
  "flow-id": {
    "afi": "ipv6",
    "src-ip": "fe80::1",
    "dst-ip": "fe80::2",
    "protocol": "udp",
    "src-port": 1234,
    "dst-port": 56789
  },
  "context": {
    "experiment-id": 3333,
    "activity-id": 12,
    "application": "grid-ftp v99.99"
  }
}
```

```
{
  "flow-id": {
    "protocol": "tcp",
    "afi": "ipv6",
    "dst-ip": "2001:67c:1bec:236::9",
    "src-port": 30175,
    "src-ip": "2001:48a8:68f7:1:192:41:231:128",
    "dst-port": 50152
```

```
  },
  "version": 1,
  "flow-lifecycle": {
    "state": "end",
    "start-time": "2021-09-21T17:46:43.406370+00:00",
    "end-time": "2021-09-21T17:46:43.406370+00:00",
    "current-time": "2021-09-21T17:46:43.406352+00:00"
  },
  "usage":{
    "received": 4234,
    "sent": 34343443,
  },
  "netlink":{
    "rtt": 33.4,
  },
  "context": {
    "experiment-id": 16,
    "application": "flowd v0.0.1",
    "activity-id": 1
  },
}
```

Appendix B: Registry JSON Schema

Flow Registry JSON Schema
https://scitags.github.io/schemas/v1.0.0/registry.schema.json

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://scitags.org/schemas/v1.0.0/registry.schema.json",
  "title": "Flow registry schema",
  "description": "Mapping of the flow identifiers to the experiments and activities",
  "type": "object",

  "properties": {
    "version": {
      "description": "The version number of the schema",
      "type": "integer",
      "minimum": 1,
      "maximum": 1,
    },
    "experiments": {
      "type": "array",
      "items": {"$ref": "#/$defs/exps"},
      "minItems": 1,
      "uniqueItems": true
    },
    "modified": {
      "description": "The UTC date/time when the content of registry was last
modified",
      "type": "string",
      "format": "date-time",
    },
  },
  "$defs": {
    "exps": {
      "type": "object",
      "required": [ "expName", "expId", "activities" ],
      "properties": {
        "expName": {
          "type": "string",
          "description": "Experiment name",
        },
        "expID": {
          "type": "integer",
          "description": "Experiment id",
        },
        "activities": {
          "type": "array",
```

```
          "description": "Experiment's activity ids",
          "items": {"$ref": "#/$defs/acts"},
          "minItems": 1,
          "uniqueItems": true
        },
      }
    },
    "acts": {
      "type": "object",
      "required": ["activityName", "activityId"],
      "properties": {
        "activityName": {
          "type": "string",
          "description": "Activity name"
        },
        "activityId": {
          "type": "integer",
          "description": "Activity id"
        }
      }
    }
  },
  "required": [ "version", "experiments"],
}


Example:
{
  "version": 1,
  "modified": "2021-09-07T22:22:21.998225+00:00",
  "experiments": [
    {
      "expName": "atlas",
      "expId" : 16,
      "activities": [
       {
         "activityName": "production",
         "activityId": 14
       },
       {
         "activityName": "rebalancing",
         "activityId": 16
       }
       ]
    },
    {
      "expName": "cms",
```

```
      "expId": 23,
      "activities": [
       {
         "activityName": "production",
         "activityId": 14
       },
       {
         "activityName": "rebalancing",
         "activityId": 16
       }
       ]
    }
  ]
}
```