WHITE PAPER

Retrieving SD-WAN Statistics

TABLE OF CONTENTS

Retrieving SD-WAN Statistics 1
Introduction 1
Prerequisites 2
Overview of FILE Retrieval Methodology 2
Getting Statistics Time Range 3
Polling for a Specific Minute 3
Statistics Zip File 3
tunnel_v2.txt 4
interface_v2.txt 4
flow_v2.txt 5
dscp_v2.txt 5
tclass_v2.txt 5
shaper_v2.txt 6
zone_pair_v2.txt 6
boost_v2.txt 6
ftype_v2.txt 7
interface_overlay_v2.txt 7
Getting Memory Statistics 7
Getting CPU Statistics 8

INTRODUCTION

The Aruba EdgeConnect Enterprise SD-WAN solution generates volumes of statistics every minute. These statistics help us debug SD-WAN networks. Other network devices generate orders of magnitude less data per device. This disparity is partly due to the meshing capability of EdgeConnect and the collection of data on every tunnel. The other factor is the frequency at which data is collected – Aruba EdgeConnect Enterprise products produce statistics every minute while the industry standard is generally every five minutes.

While the Aruba Orchestrator is the main consumer of this data and uses it extensively in reporting and troubleshooting, there are many situations where customers will need to access these statistics. The use cases for statistics ranges substantially, and can span anything from integration with diagnostic tools, to external reporting tools, to the development of customized health dashboards. Orchestrator offers a REST API to provide periodic access to statistics (including real-time, time-series, and aggregated statistics), but for use cases that require large volumes of statistics or frequent polling of statistics (such as access

to minute-granularity statistics), Orchestrator is not designed for these use cases and thus, not the right integration point.

The REST API is not suited for high-frequency polling of granular statistics or "data replication" into an external data store for two reasons: 1) the volume of data that is requested scales linearly with the number of EdgeConnect devices and the granularity of the requested statistics, and 2) Orchestrator does not have the resources for the load that frequent REST API calls would impose. Orchestrator serves multiple purposes and has limited resources tuned to do many functions, including:

- Supporting 5-50 concurrent users accessing the UI
- Orchestrating policies across the SD-WAN fabric with 2,000 to 3,000 devices
- Collecting and reporting statistics from these devices for Orchestrator users
- Servicing third-party API requests from external systems

Collecting and reporting statistics consumes large amounts of CPU and memory resources. Since Orchestrator is not designed to provide end applications with a copy of its statistics, Aruba provides REST APIs that allow customers to obtain this data *directly* from the appliances for use by third-party applications. This approach is required when the use case calls for access to fine-grained statistics and/or frequent access to the statistics and avoids overloading Orchestrator by distributing the load across the SD-WAN.

PREREQUISITES

Before collecting statistics using the EdgeConnect REST APIs, the following prerequisites must be met:

1. **Appliance access**: To access the EdgeConnect REST API directly, customers must have an up-to-date list of appliances and a method to access their REST API endpoints. The list of appliances and their configuration information can be retrieved from the Orchestrator REST API as part of inventory discovery.
2. **Resource name or ID**: Depending on which resource the customer is interested in (for example, tunnels, interfaces, flows, and so on), the identity of that resource needs to be known. This can be obtained from the Orchestrator REST API as part of inventory discovery. The resource's ID or name is used to identify and associate the statistics in the files.
3. **Login access to EdgeConnect gateways**: File retrieval from EdgeConnect requires login access to EdgeConnect to ensure authorized access. Aruba recommends that customers create or orchestrate an "api-user" login on the EdgeConnect appliances for use by the third-party application that is retrieving the files.

OVERVIEW OF FILE RETRIEVAL METHODOLOGY

To use EdgeConnect statistics in an external application, one can poll each appliance directly using REST APIs. The following steps outline how to do this.

1. Invoke the EdgeConnect REST API to determine which statistics files are available. EdgeConnect stores a fixed number of statistics files. The REST API call provides information about the time range that is currently available.
2. Request the stats file for the minute(s) of interest. The stats files are zipped. Information about which stats are contained in each file is described below.
3. Store the files of interest and parse or ingest them as needed. The file formats for various statistics files are described below.

Customers can use this procedure to obtain detailed statistics, including the popular requests for Tunnel stats (loss, latency, jitter, and so on), Interface stats, and Flow stats.

Note: The process for retrieving CPU and memory stats for each EdgeConnect follows a different process and is not file-based.

Getting Statistics Time Range

Appliances generate statistics every minute. Each minute, appliances generate statistics of different types in different CSV files. These CSV files are zipped into a single file for convenience. Appliances are also configured to keep only a certain number of these files. Appliances also retain statistics for a specified time period only. The polling systems (Orchestrator or third-party systems) must get their data before these files age out. Most appliances are configured to keep this data for at least a few hours. You can configure the length of time appliances keep data in the Orchestrator.

A poller calls the following REST API to get a range of timestamps for which data is present on the appliance:

```
GET /rest/json/stats/minuteRange
```

This will return the following output:

```
{ "newest": "1649778540", "oldest": "1649691900" }
```

These two numbers are minute boundaries expressed in standard epoch seconds. This example indicates Tuesday, April 12, 2022 3:49:00 PM to Monday, April 11, 2022 3:45:00 PM.

Polling for a Specific Minute

Once you obtain the time range, you can iterate over every minute (or start from the last minute not yet retrieved). For example:

```
for (int i=1649691900; i<=1649778540; i+60) {
   retrieve zip file
   unzip
   insert stats into your database
}
```

The API for retrieving a zip file for a specific minute looks like this:

```
GET /rest/json/stats/minuteStats/st2-1649691900.tgz
```

All file names start with `st2-` and are followed by a minute timestamp in epoch seconds. The file extension is `.tgz`

Statistics Zip File

The zip file contains the following files:

1. tunnel.csv, tunnel_peak.csv, tunnel_v2.txt
2. interface.csv, interface_peak.csv, interface_v2.txt
3. flow.csv, flow_peak.csv, flow_v2.txt
4. dscp.csv, dscp_peak.csv, dscp_v2.txt
5. tclass.csv, tclass_peak.csv, tclass_v2.txt
6. shaper.csv, shaper_v2.txt
7. jitter.csv, mos.csv
8. zone_pair.csv, zone_pair_v2.txt
9. boost.csv, boost_v2.txt
10. drops.csv, drops_v2.txt
11. drc.csv
12. ftype.csv, ftype_peak.csv, ftype_v2.txt
13. interface_overlay.csv, interface_overlay_v2.txt

Note: `_v2.txt` files are a new format optimized for the new Distributable Stats Collector. Aruba recommends that customers use this file format. The following sections describe the format, metrics, and units of v2 files.

tunnel_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
tunnel_54, // tunnel id
to_falcon_MPLS1-MPLS1, // tunnel alias (name)
0, // overlay id (0 for underlay, 1-7 for overlays)
0, // is sd-wan tunnel (0 for false, 1 for true)
unknown, // orchestrator tid -- ignore
0,0,0,0, // wan tx, wan rx, lan tx, lan rx bytes
0,0,0,0, // wan tx, wan rx, lan tx, lan rx packets
0, // average latency in milliseconds*100
0, // min latency during the minute in milliseconds*100
0,0,0, // TCP flow count, TCP Accelerated flow count, non-TCP flow count
0,0, // Flows Created, Flows Deleted
0,0, // Lost packets pre-FEC, Lost packets post-FEC
0,0, // Loss percentage*100 pre-FEC, Loss percentage*100 post-FEC
0,0, // Out of order packets pre-POC, Out of order packets post-POC
0,12, // Out of order percentage*100 pre-POC, Out of order percentage*100 post-POC
0,1248, // overhead wan rx packets, overhead wan tx packets
0,0, // overhead wan rx bytes, overhead wan tx bytes
0,0, // overhead header bytes wan rx, overhead header bytes wan tx
0, // bandwidth utilization percentage*100 (ignore - not measured properly)
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, // ignore peak values (29
columns)
440,440, // postfec and prefec MOS (MOS score*100)
4786200503987994624,4786200503987994624, // ignore
0, // jitter in milliseconds
0, // ignore
1649778540 // timestamp
```

interface_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
bvi0, // interface name
1, // interface label id
3, // traffic type 1-optimized traffic, 2-pass-through-shaped, 3-pass-through-unshaped &
4-all traffic
1272,1272, // bytes tx, bytes rx
6,6, // tx packets, rx packets
0,0, // overhead bytes tx, overhead bytes rx
```

```
0,0, // overhead tx packets, overhead rx packets
0,0, // firewall bytes tx, firewall bytes rx
0,0, // firewall tx packets, firewall rx packets
0,0, // max bandwidth tx, max bandwidth rx
4851502700234658625,4851502700234658625,4575657223058215745,4575657223058215745,0,0,0,0,
0,0,0,0,// ignore - these are peak values with peak timestamp combined
1649791860 // timestamp
```

flow_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
3, // flowtype 1-TCP Accelerated, 2-TCP Not Accelerated & 3-Non TCP
3, // traffic type 1-optimized traffic, 2-pass-through-shaped, 3-pass-through-unshaped &
4-all traffic
0, // flows created
0, // flows deleted
1, // flows exist
0, 0, 4575657223058215737, // ignore - these are peak values
1104,1104,1104,1104, // wan tx, wan rx, lan tx, lan rx bytes
6,6,6,6, // wan tx, wan rx, lan tx, lan rx packets
4843621400886760257,4843621400886760257,4843621400886760257,4843621400886760257,45756572
23058215745,4575657223058215745,4575657223058215745,4575657223058215745,// ignore -
these are peak values with peak timestamp combined
1649791860 // timestamp
```

dscp_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
0, // DSCP
3, // traffic type 1-optimized traffic, 2-pass-through-shaped, 3-pass-through-unshaped &
4-all traffic
1272,1272,1272,1272, // wan tx, wan rx, lan tx, lan rx bytes
6,6,6,6, // wan tx, wan rx, lan tx, lan rx packets
4851502700234658625,4851502700234658625,4851502700234658625,4851502700234658625,45756572
23058215745,4575657223058215745,4575657223058215745,4575657223058215745,// ignore -
these are peak values with peak timestamp combined
1649791860 // timestamp
```

tclass_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
0, // Traffic class
3, // traffic type 1-optimized traffic, 2-pass-through-shaped, 3-pass-through-unshaped &
4-all traffic
1272,1272,1272,1272, // wan tx, wan rx, lan tx, lan rx bytes
```

```
6,6,6,6, // wan tx, wan rx, lan tx, lan rx packets
4851502700234658625,4851502700234658625,4851502700234658625,4851502700234658625,4575657223058215745,4575657223058215745,4575657223058215745,4575657223058215745,// ignore - these are peak values with peak timestamp combined
1649791860 // timestamp
```

shaper_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
0, // Traffic class
0, // 0-Outbound,1-Inbound
1040, // the total amount of bytes being shaped
1040, // the amount of bytes used for shaping
20, // the amount of packets used for shaping
0, // the specified amount of time (ms) Orchestrator waits until packets are dropped while shaping is in progress
20, // wait count
0, // the amount of packets that have been reported as dropped due to expiration in the Shaper queue
0, // all other drops besides the expired drop packets
1649791860 // timestamp
```

zone_pair_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
0, // from zone
0, // to zone
100, // flows allowed
104, // flows dropped
20, // policy drop packets
0, // policy drop bytes
20, // routing drop packets
0, // routing drop bytes
100, // tx packets
100, // tx bytes
100, // rx packets
100, // rx bytes
1649791860 // timestamp
```

boost_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
200, // the amount of Boost configured for the selected appliance (Kpbs)
276674473, // boost bytes
```

```
8, // seconds not boosted in a minute
1, // ignore
1649791860 // timestamp
```

ftype_v2.txt

These stats are now available in flow_v2.txt.

interface_overlay_v2.txt

Each row has the following format:

```
1.NE, // Orchestrator key for the appliance
bvi0, // interface name
1, // interface label id
0, // overlay id (0 for underlay, 1-7 for overlays)
2, // tunnel type
1272,1272, // bytes tx, bytes rx
6,6, // tx packets, rx packets
0,0, // overhead bytes tx, overhead bytes rx
0,0, // overhead tx packets, overhead rx packets
0,0, // max bandwidth tx, max bandwidth rx
4851502700234658625,4851502700234658625,4575657223058215745,4575657223058215745,0,0,0,0,
// ignore - these are peak values with peak timestamp combined
1649791860 // timestamp
```

GETTING MEMORY STATISTICS

Retrieving memory usage statistics from EdgeConnect gateways is achieved via the REST API and not file retrieval. The REST API provides an instant snapshot of current memory usage. Memory usage of an EdgeConnect gateway does not change because EdgeConnect gateways use statically allocated memory for 99% of their functions. But occasionally, memory leaks can occur. You can use the following API to monitor and detect potential leaks.

```
GET /rest/json/memory
```

This will return data in following format:

```
{
  "total": 3932860,
  "free": 885672,
  "buffers": 135056,
  "cached": 668484,
  "used": 3047188,
  "swapTotal": 3906244,
  "swapFree": 3905476,
  "swapUsed": 768
}
```

Each number is in kilobytes. The key value to monitor is the “free” value. This number must be above 256,000 for an EdgeConnect to be considered healthy. This number can go as low as 50,000 and EdgeConnect will still function normally.

However, Aruba recommends setting alarms at 256,000.

GETTING CPU STATISTICS

CPU usage can also be monitored with the EdgeConnect REST API. Monitoring CPU usage on the EdgeConnect and setting a threshold crossing alert can be complicated. This is because many CPUs work on “compression” and if the CPU is completely busy, it has less compression. There are other CPUs, however, where 100% CPU usage will result in the EdgeConnect dropping packets.

Note: The critical CPUs to monitor depends on the hardware model and software version. This document does not address these dependencies.

Customers can use the following API to get CPU statistics.

```
GET /rest/json/cpustat?time=0
```

This will return the latest CPU stats from now until the last 5 minutes, in 5 second intervals (total of 60 objects). This might return a large amount of data and should meet the needs of most customers.

The “pIdle” statistic is the statistic of interest to most customers. All numbers are expressed as a percentage.

Aruba recommends that customers set the Threshold Crossing Alert to 95% for warning and 99% for alert.


```
{
  "latestTimestamp": 1652470818069,
  "data": [
    {
      "1652470527344": [
        {
          "cpu_number": "ALL",
          "pIdle": "80.44",
          "pUser": "14.67",
          "pSys": "4.89",
          "pIRQ": "0.00",
          "pNice": "0.00"
        },
        {
          "cpu_number": 0,
          "pIdle": "76.86",
          "pUser": "16.14",
          "pSys": "7.01",
          "pIRQ": "0.00",
          "pNice": "0.00"
        },
        {
          "cpu_number": 1,
          "pIdle": "83.88",
          "pUser": "13.27",
```

```
        "pSys": "2.86",
        "pIRQ": "0.00",
        "pNice": "0.00"
      }
    ]
  },
  {
    "1652470532355": [
      {
        "cpu_number": "ALL",
        "pIdle": "79.15",
        "pUser": "16.49",
        "pSys": "4.36",
        "pIRQ": "0.00",
        "pNice": "0.00"
      },
      {
        "cpu_number": 0,
        "pIdle": "74.68",
        "pUser": "19.62",
        "pSys": "5.70",
        "pIRQ": "0.00",
        "pNice": "0.00"
      },
      {
        "cpu_number": 1,
        "pIdle": "83.47",
        "pUser": "13.47",
        "pSys": "3.06",
        "pIRQ": "0.00",
        "pNice": "0.00"
      }
    ]
  },

.......

  {
    "1652470818069": [
      {
        "cpu_number": "ALL",
```

```
            "pIdle": "80.06",
            "pUser": "16.03",
            "pSys": "3.91",
            "pIRQ": "0.00",
            "pNice": "0.00"
          },
          {
            "cpu_number": 0,
            "pIdle": "75.99",
            "pUser": "18.58",
            "pSys": "5.43",
            "pIRQ": "0.00",
            "pNice": "0.00"
          },
          {
            "cpu_number": 1,
            "pIdle": "84.01",
            "pUser": "13.56",
            "pSys": "2.43",
            "pIRQ": "0.00",
            "pNice": "0.00"
          }
        ]
      }
    ]
}
```