Segmented Telemetry Data Filter

Administrator's manual

Eduard Tibet

28.03.2022

Table of Contents

1. Introduction 1
1.1. Scope of this document 1
1.2. Document structure 1
2. Description of the STDF 2
2.1. Brief description of the STDF 2
2.2. Overall design of STDF 2
3. Installation of the software 4
3.1. System requirements 4
3.2. User qualification 4
3.3. Installation process of components 4
4. Authoring filtering rules 7
5. Using and verifying filtered data 8
6. Troubleshooting 8
6.1. Problem: no connection from a filter node to a coordinator 8
6.2. Problem: filtering node doesn't receive filtering rules 8
6.3. Problem: filtering node doesn't receive data 9
6.4. Problem: loadbalancer doesn't receive any data 9
6.5. Problem: Filter produces incorrect results 9
A. Technology stack behind this sample document 9
B. License 10

1. Introduction

1.1. Scope of this document

This is a complete administrator's manual of the Segmented Telemetry Data Filter (STDF) software. It describes in a brief what STDF is proposed for, its overall design, what each component is indented for. Also this manual includes a full information about an installation process and usage of STDF. The theory and principles of data filtering, explanation of the Erlang language syntax (used for data filtering) are completely out of scope of this manual.

1.2. Document structure

This document includes a following parts:

- Introduction - current section.
- Description of the STDF - a description of the software's overall design, features and functionality.
- Installation of the software - the information about system requirements and installation of the software.
- Authoring filtering rules - current section describes, how to create and mastering filtering rules required to be deployed into the one of the software component.

- Using and verifying filtered data - section about customizing and fine tuning final data.
- Troubleshooting - list of possible issues and ways to resolve them.

2. Description of the STDF

2.1. Brief description of the STDF

STDF is a data handling software designed to help in capturing high speed telemetry data. The purpose of the STDF is to automatically and linearly scale processing capacity for such data. The STDF segments data into smaller chunks and sends them through a load balancer to several servers that filter received data. That way it is possible to:

- avoid using a single high-powered processing unit working with data;
- reduce power of any unit, used for processing;
- deploy the system with a great flexibility and scalability, based on various initial requirements and/or conditions.

2.2. Overall design of STDF

The system contains of several parts:

- coordinator component (node) - is used for smart management of the whole system;
- loadbalancer component (node) - is used for receiving raw data from external sources (i.e. sensors) and transfer it further based on coordinator's directives;
- filter component(s)/node(s) - are used to process data received from the loadbalancer. Processing is based on the current workload. If it exceeds the maximum, defined by a coordinator, data chunks automatically migrate to other filter nodes, which free resources are enough to manipulate the data. The number of filter components within installation varies and based on current performance needs.

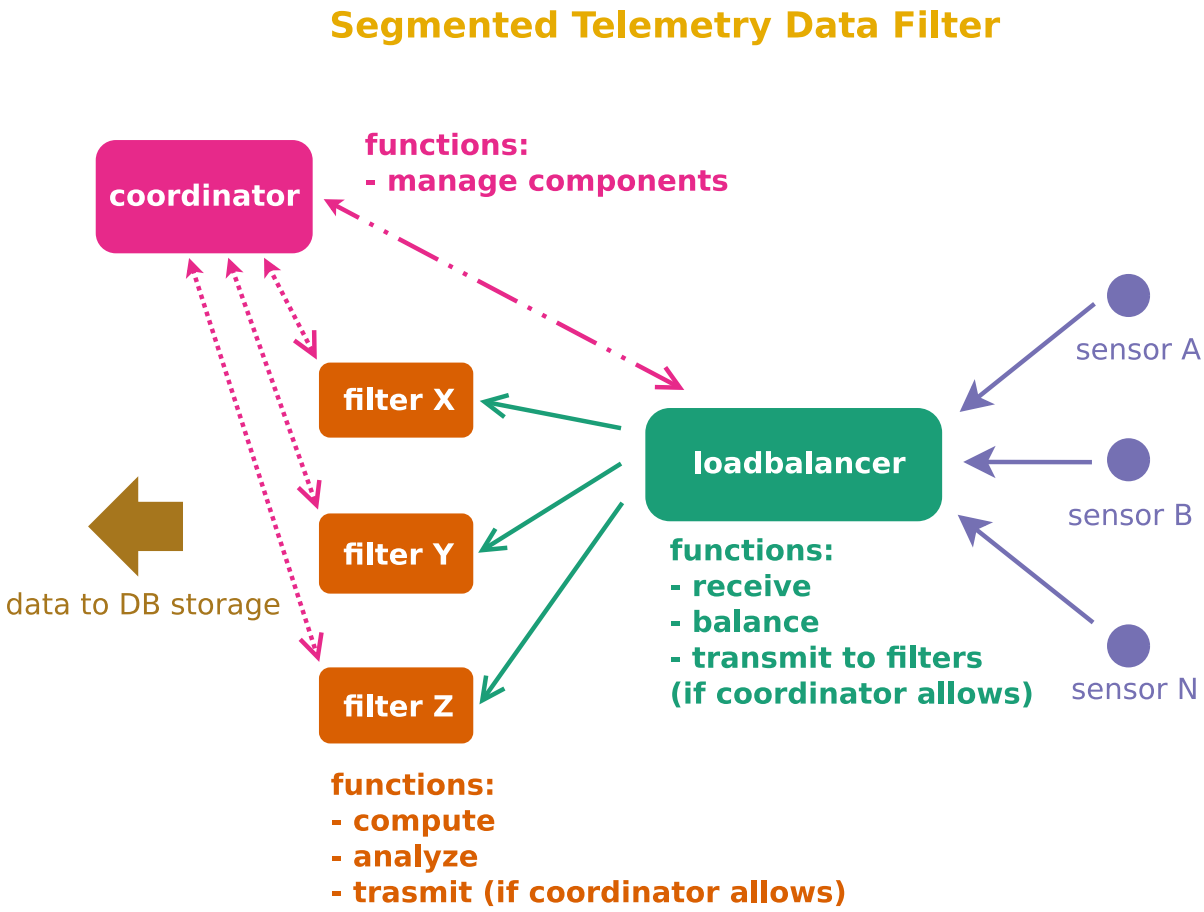
In the heart of the STDF is a proprietary protocol that was developed by Teliota company. This protocol can be used between components to coordinate data manipulation, calculation on individual filters, running on each server, and data migration between filters.

The typical workflow includes the following steps:

1. loadbalancer component receives all-raw data from external sources (i.e. sensors) and transmit it further to filters based on coordinator's current workload rules and internal logic;
2. filter component receives an independent dataset from the loadbalancer and asks a cluster's coordinator to supply a filtering rules;
3. coordinator provides a rules to the filter and then rules are applied on-the-fly onto the incoming data, received from the loadbalancer;

Each filtering component can talk to a coordinator component about the data it is processing or wishes to process. The coordinator component steers the loadbalancer component what data a loadbalancer should provide to which filter node.

Figure 1. Overall design of STDF



If a filter component gets overloaded by the data, its tasks can be offloaded to another filter node. Due to the nature of the workflow, the algorithm assumes that:

- a sufficient number of such redundant servers (filter modes) exists in the pool as during an overload situation;
- the offloaded data is similar to the original data and can be filtered with same rules.

An offloaded filter node is, therefore, not "independent". It have to process the same data and instructions as its peer until the moment an overload situation is resolved.

New processing (filter) nodes can be added into the processing cluster on the fly by:

1. adding new server hardware;
2. installing the filter component software onto it;
3. configuring the coordinator server address.

The filter node will register itself to the coordinator and the coordinator will instruct the loadbalancer to forward traffic to this new node.

Telemetry data and filter operations are defined with a definition file that in turn is written in a proprietary filter rule language. The language defines in details:

- what the incoming data is stands for;
- how the data may be aggregated and filtered out in case of outliers or unwanted values are found.

The coordinator reads the filter language files and runs them on its own logic processing engine. This engine is connected to all the filtering nodes, which receives processing instructions in the form of a proprietary, compressed command protocol. The protocol is bidirectional:

- filter nodes and the loadbalancer inform the coordinator about data they receive and their status.
- coordinator instructs:
  - loadbalancer - where to deploy initial raw-based data;
  - filters - what data is and how that data should be manipulated over.

3. Installation of the software

3.1. System requirements

To successfully install and run STDF, your base hardware/software installation have to be complied with the following requirements:

- Two (2) dedicated hardware servers for a coordinator and a loadbalancer components;
- no other application software (i.e. MTA, DB, etc.), except of an operating system and system utilities should be installed on the above servers;
- required amount of servers that will be used as hosts for a filtering components (nodes);
- network connectivity with all sensors that gather information for your application - your firewall rules should allow sensors to access the STDF cluster (loadbalancer component);
- network connectivity within all components of the STDF installation and data receivers beyond the STDF deployment (DB or third-party application servers);
- any recent Linux distribution with a kernel 2.6.32 or later;
- standard (base) Linux utilities, including:
  - `tar` - utility to work with `.tar` files;
  - `wget` - utility to get packages from the distribution server;
  - any console text editors to edit configuration files - i.e. `vim`, `nano`, etc.

3.2. User qualification

To install and maintain STDF system administrator have to have:

- skills equals to those, that are enough to successfully pass the LPIC-2 exam;
- some knowledge of Erlang language syntax to write filtering rules.
- read throughly a "STDF filtering rules language reference" manual (supplied by Teliota separately).

3.3. Installation process of components

3.3.1. Getting packages of components

All packages are to be downloaded from a Teliota distribution web server: https://download.teliota.com .

3.3.2. Installation of a coordinator component

To install a coordinator component:

1. Go the the top level installation directory.
2. Make a directory for coordinator's files:

```
$ mkdir stdf_coordinator
```

3. Change a directory to the recently created one:

```
$ cd stdf_coordinator
```

4. Download the package with a coordinator component:

```
$ wget https://download.teliota.com/bin/stdf_coordinator.tar.bz2
```

5. Untar coordinator component files:

```
$ tar -xjf stdf_coordinator.tar.bz2
```

6. Open configuration file `config.ini` in any text editor and set up the IP and port that coordinator component should listen on:

```
COORDINATOR_SERVER_LISTEN_IP=192.168.2.53
COORDINATOR_SERVER_LISTEN_PORT=8860
```

7. Change directory the `bin/` folder:

```
$ cd bin/
```

8. Check if the file `stdf_coordinator.sh` have an execution bit turned on.
9. Run the coordinator:

```
$ ./stdf_coordinator.sh
```

The coordinator is needed to be fed by filtering rules. The coordinator includes a separate language parsing and debugging tool which validates a filter rule.

> **Note**
>
> It is assumed that you have filtering rules already written. If you haven't any rule written yet, first check the section Authoring filtering rules.

To deploy a filtering rule:

1. Check the filtering rule:

```
$ ./stdf_parser.sh -i [rulefile1]
```

2. If there are any output messages - read them carefully. These messages also saved within a log file for the future analysis.

3. Copy the rule file to a `filter_rules` directory within the coordinator installation:

```
$ cp [rulefile1] ../filter_rules
```

4. Open configuration file `config.ini` in any text editor and add recently copied file into the coordinator's configuration file:

```
COORDINATOR_RULES_FILES=rulefile1,rulefile2
```

5. Restart the coordinator component:

```
$ ./stdf_coordinator.sh restart
```

3.3.3. Installation of a loadbalancer component

To install a loadbalancer component:

1. Change a current directory to the top level installation one.
2. Make a directory for the loadbalancer component files:

```
$ mkdir stdf_loadbalancer
```

3. Change a directory to the recently created one:

```
$ cd stdf_loadbalancer
```

4. Download the package with a loadbalancer component:

```
$ wget https://download.teliota.com/bin/stdf_loadbalancer.tar.bz2
```

5. Untar the loadbalancer component files:

```
$ tar -xjf stdf_loadbalancer.tar.bz2
```

6. Open configuration file `config.ini` in any text editor and point the loadbalancer to the coordinator's IP address and port number:

```
COORDINATOR_SERVER_IP=192.168.2.53
COORDINATOR_SERVER_PORT=8860
```

7. Change directory to the `bin/` folder:

```
$ cd ./bin
```

8. Check if the file `stdf_loadbalancer.sh` have an execution bit turned on.
9. Run the loadbalancer component:

```
$ ./stdf_loadbalancer.sh
```

3.3.4. Installation of a filtering component

To install a filtering component:

1. Change a current directory to the top level installation one.
2. Make a directory for filtering component files:

```
$ mkdir stdf_node
```

3. Change a directory to the recently created one:

```
$ cd stdf_node
```

4. Download the package with a filtering component:

```
$ wget https://download.teliota.com/bin/stdf_node.tar.bz2
```

5. Untar the filtering component files:

```
$ tar -xjf stdf_node.tar.bz2
```

6. Open configuration file `config.ini` in any text editor and point the filtering component to the coordinator's IP address and port number:

```
COORDINATOR_SERVER_IP=192.168.2.53
COORDINATOR_SERVER_PORT=8860
```

7. Change directory to the `bin/` folder:

```
$ cd ./bin
```

8. Check if the file `stdf_node.sh` have an execution bit turned on.
9. Run the filtering component:

```
$ ./stdf_node.sh
```

10. Repeat above steps for all filter components are to be installed.
11. Start feeding data into the data interface of the loadbalancer component.

4. Authoring filtering rules

Note

This section only briefly describes filtering rules structure. For a detailed information take a look into the "STDF filtering rules language reference" manual (supplied separately).

Filtering rules are defined utilizing a filtering language that uses Erlang language syntax as a basis.

Each filtering rule includes three elements (so called "definitions"):

- data definition - describes nature of data to be filtered, including the pattern how the incoming data can be recognized (e.g. port, input url, data header); the data definition assigns an identifier to the dataset so that the data correlation and filter rules can refer to it;
- correlation definition - describes how that data depends on itself or some other identified dataset;
- filter definition - describes what actions are to be taken for the data, when it arrives.

5. Using and verifying filtered data

The filtering cluster appoints one of its nodes automatically as a forwarder, based on the load of the servers. The forwarder collects the data from each filtering node, combines it into one stream, and sends it to whatever server is designated as the final receiver (destination).

Important

The filtering components (nodes) don't store any data - they only perform filtering. You have to define and configure the storage server beyond the STDF deployment that will perform any and all database processing. A connection to a designated DB server is configured within a coordinator component configuration file `config.ini`.

The forwarder can optionally inject additional data headers and trailers into the initial data block for easier recognition of its nature - source transmitter/generator. The trailer may contain a CRC for checking data integrity. The algorithm for the CRC is shown below:

```
def crc16(self, buff, crc = 0, poly = 0xa001):
    l = len(buff)
    i = 0
    while i &lt; l:
        ch = buff[i]
            uc = 0
            while uc &lt; 8:
            if (crc &amp; 1) ^ (ch &amp; 1):
                crc = (crc &gt;&gt; 1) ^ poly
            else:
                crc &gt;&gt;= 1
            ch &gt;&gt;= 1
            uc += 1
        i += 1
    return crc

crc_byte_high = (crc &gt;&gt; 8)
crc_byte_low = (crc &amp; 0xFF)
```

6. Troubleshooting

6.1. Problem: no connection from a filter node to a coordinator

Possible reasons	How to solve a problem
Any of coordinator's node IP settings of a filter node are not correct or were not set.	Check for a correct IP and port numbers of filters.
Firewall rules don't allow filter packets to reach a coordinator	Check if coordinator firewall settings (open ports and IP rules) are correct.
Coordinator node is not running	Check if coordinator is really running.

6.2. Problem: filtering node doesn't receive filtering rules

Possible reason	How to solve a problem
Any of coordinator's node IP settings of a filter node are not correct or were not set.	Check for a correct IP and port numbers (see above problem's first solution).
Errors in filtering language	Check coordinator's log file for errors

Possible reason	How to solve a problem
Issues with network connectivity or software used	Check coordinator's log file for errors; check node firewall settings

6.3. Problem: filtering node doesn't receive data

Possible reason	How to solve a problem
Loadbalancer is not running	Check for errors in loadbalancer log files
Ports are close or filtered by firewall	Check node firewall settings
There are no actual data received	Check loadbalancer log file of transmitted data

6.4. Problem: loadbalancer doesn't receive any data

Possible reason	How to solve a problem
Loadbalancer is not running	Check if loadbalancer is running and check for errors in loadbalancer's log files.
Ports are close or filtered by firewall	Check loadbalancer firewall settings

6.5. Problem: Filter produces incorrect results

Possible reason	How to solve a problem
Incorrect filter initial setup	Run node with higher level of verbosity: start them with `./stdf_node.sh -vvv` and then check log files for possible issues
Incorrect filter rules	Run filter language parser and validate it's actual syntax: run `./stdf_parser.sh --validate [rulefile1]`

A. Technology stack behind this sample document

The source files of this document:

- were completely written in DocBook/XML 5.1[1] format which is OASIS Standard[2];
- were WYSYWYM-authored by using of XMLmind XML Editor[3] version 7.3 by XMLmind Software[4] installed on author's desktop running Debian GNU/Linux 10.11 (buster)[5]. Also author used Dia Diagram Editor[6] for diagrams.
- are freely available at Github as a docbook-samples project[7];
- are distributed under Creative Commons License - for details see License.

To produce `.fo` file of this document the following software were used:

- The local copy of DocBook XSL Stylesheets v. 1.79.1[8] was used.

[1] https://docbook.org/xml/5.1/
[2] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=docbook
[3] http://www.xmlmind.com/xmleditor/
[4] http://www.xmlmind.com
[5] https://www.debian.org/
[6] http://dia-installer.de/
[7] https://github.com/eduardtibet/docbook-samples
[8] http://docbook.sourceforge.net/release/xsl/

- Author's customization layer of the above stylesheets that is now a docbook pretty playout[9] project, freely available at Github.
- `xsltproc` as an engine to produce `.fo` file from the DocBook source `.xml` file (`xsltproc` compiled against `libxml` 20904, `libxslt` 10129 and `libexslt` 817).

To get the result `.pdf` file from a `.fo` file author used Apache FOP 2.3[10] engine with a foponts project[11], created and maintained by the author of this document.

B. License

[9] https://github.com/eduardtibet/docbook-pretty-playout
[10] http://xmlgraphics.apache.org/fop/
[11] https://github.com/eduardtibet/foponts
[1] https://creativecommons.org/licenses/by-nc-sa/4.0/