University of Colorado
Interdisciplinary Telecommunications Program
TLEN 5700

Concept Definition Document (CDD)

Decision-making System driven by Big Data Telemetry for SDN and Traditional Networks

October 2017

1.0 Project Information

Project Customer:

Name: Levi Perigo
Email: levi.perigo@colorado.edu

Team Members

Name: Ankur Jain
Phone: (720) 755-7806
Email: Ankur.Jain-1@colorado.edu

Name: Arohi Gupta
Phone: (720) 318-2046
Email: arohi.gupta@colorado.edu

Name: Ashutosh Gupta
Phone: (720) 251-0684
Email: ashutosh.gupta@colorado.edu

Name: Sanjay Murthy
Phone: (720) 755-7783
Email: sanjay.venkateshmurthy@colorado.edu

Name: Leidy Pérez
Phone: (303) 889-9280
Email: leidy.perez@colorado.edu
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 Project Information</td>
<td>1</td>
</tr>
<tr>
<td>2.0 Project Description</td>
<td>3</td>
</tr>
<tr>
<td>3.0 Design Requirements</td>
<td>7</td>
</tr>
<tr>
<td>4.0 Key Design Options Considered</td>
<td>8</td>
</tr>
<tr>
<td>5.0 Trade Study Process and Results</td>
<td>12</td>
</tr>
<tr>
<td>6.0 Selection of Baseline</td>
<td>14</td>
</tr>
<tr>
<td>7.0 References</td>
<td>15</td>
</tr>
</tbody>
</table>
2.0 Project Description

In the early-to-mid 1990s the Internet’s growth changed the rules and due process in the daily lives of users all around the globe. What started as email for scientists and early file transfer systems, is now what moves every fiber of society regardless of culture, expertise or even direct exposure to connectivity.

As Internet traffic increased over the years, it was soon evident that traditional IP network management practices fall short when it comes to areas such as control over paths used to route traffic. In [1], traffic-engineering capabilities under traditional networks, even in the early 2000s, are characterized by the following statement: “The means for performing traffic engineering using conventional routing protocols were primitive at best.” Researchers foresaw the need to transition to a more programmable networking practice and the foundation for Software Defined Networks (SDN) was established.

The concept of SDN revolutionized the way network engineers approach traffic engineering by allowing a high level of abstraction in comparison with traditional networks. A typical intent-based approach allows a user or application to only specify the intent such as the requirement of low latency path between two nodes, bandwidth reservation, and real-time monitoring for the re-evaluation of selected paths [2]. In theory, this characteristic alone should be a “blanket” solution for efficient routing decisions.

However, SDN adoption has been challenging. Only in recent years, has the industry finally entered a transition stage from traditional networking practices. The global SDN and Network Functions Visualization (SDN/NFV) market is expected to grow from $3.68 billion in 2017 to $54.41 billion by 2022, at a Compound Annual Growth Rate (CAGR) of 71.4% [3]. While some telecom providers such as AT&T and Verizon have already started implementing an SDN based architecture [4], many other providers are relying on mixed and hybrid models moving forward. When a network is designed to work with both traditional and SDN elements, it impairs the ability of the network manager to use some of the core benefits of SDN, such as making intelligent, global decisions for the network driven by the controller.

The objective of this project is to model an intelligent, decision-making system compatible with traditional, mixed and SDN networks that leverages SDN’s intent based networking mechanics and big data telemetry collection to enable comprehensive analysis and traffic engineering.

In order to achieve this goal, three levels of success have been defined:

**Level 1:**

**Objective:** Create a Python-based framework, which can efficiently perform telemetry data collection from both an SDN controller and traditional network devices.

**Requirements:** Data collection has to be fault tolerant, reliable, and scalable. Once the data is collected, it must be sent to PNDA for storage, processing, and general
trends analysis. Additionally, all steps of the process should be accessible for monitoring via a web-based Graphical User Interface (GUI) interface with a limited control over the process.

**Business/Policy:** Document industry tendencies and trends regarding SDN penetration and traditional vs mixed vs hybrid* vs fully SDN. Study the policy case implications of data collection in the main telecommunications markets around the globe to determine if it would affect a widespread implementation of a data collection and analysis system.

**Level 2:**

**Objective:** In addition to fulfilling all Level 1 requirements, the system will progress to analyze data in order to predict basic networking decisions using the parameters collected described in sections below in this same document.

**Requirements:** In order to do this, it must discern data relevant for each type of networking decision. The data analyzed will depend on what objective will be the key driver to traffic decisions such as CPU utilization, load balancing, or traffic engineering. The web-based GUI should offer the option to navigate between types of networking decisions.

**Business/Policy:** Develop a business analysis for the scalability of the project. Including financial implications of handling larger amounts of traffic derived from aspects such as processing power and server storage capacity. Determine barriers for agreements with Internet Service Providers (ISPs) to subject their traffic to a centralized analysis alongside other providers.

**Level 3:**

**Objective:** After testing and ensuring the success of Level 2 technical requirements and completing non-technical objectives, it will be necessary to reinforce and test security within the system.

**Requirements:** Incorporate traffic encryption as it is retrieved from the underlying network and sent to the central repository of information. Once all traffic handling is secure, start aggregating functionalities to the Python framework such as detection and prevention of common threats to networks.

**Business/Policy:** Develop a financial study on benefits of making routing decisions based on non-traditional criteria with a large sample of ISPs. Moreover, define a common policy framework that would be applicable to the top telecommunications markets in order for the project to be applicable to a large sample of global Internet traffic.

*The difference between hybrid and mixed being the equipment used; a mixed layout uses traditional hardware and SDN hardware separately, while a hybrid setting uses equipment compatible with both systems.*
The Overview of the System diagram in Figure 1 and the Concept of Operations (CONOPS) Diagram in Figure 2 illustrate how the components to the system will work in order to achieve the levels of success:

**Figure 1. Overview of the System**

**Figure 2. Concept of Operations (CONOPS) Diagram**
Considering a scenario where Host A in one network wishes to communicate with Host B in another network, Figure 2 shows an example of analyzing the interaction going through the data collection and best path routing system:

1. The Python based software is actively collecting data from all networks regardless of their configuration and sending information to PNDA for storage and analysis.

2. When Hosts A communicates with Host B, when defined by the operator via the GUI, the Python framework queries PNDA’s database for the best networking decision.

3. PNDA sends back the information necessary to decide on the best path the traffic could take when different routing criteria are applied.

4. The Python software will be the decision maker, based on the user’s request, on what solution to send back as a command for the system, considering relevant factors such as latency and throughput depending on the type of traffic. I.e. when handling a video upload request, the script would prioritize throughput as opposed to a file download where latency could be prioritized when making the decision.

5. Once a decision is made, the Python script relays it to the networking devices via an API or socket.

The design has critical elements that define its feasibility and course of action. In order to account for these, a definition of Critical Project Elements (CPEs) has been developed and includes the following:

**Technical**

**CPE 1.1 Python framework driving the collection of data from both SDN and traditional networks.** The Python framework will act as an automated telemetry collection system running as software on a server. It must have simultaneous connectivity to both SDN and traditional networks in order to pull necessary networking data. As the software acquires this data, its algorithm must also orchestrate the transmission of the metrics to PNDA for storage, monitoring, analysis. As the project evolves and moves up the levels described in section 3.0, this Python based software will be the core where every targeted networking decision function originates.

**CPE 1.2 Integration of PNDA as the big data database for data collection and network data analytics.** PNDA will be running on a separate server, acting as the central repository for network data collection and analysis. With the Python based software acting as the link between the underlying network and PNDA, it is crucial to have a fault tolerant and reliable system that can handle large volumes of traffic. Since PNDA is an open source big data analytics platform, its
implementation will enhance the scalability of the project while keeping economic impact at minimum. We would be hosting PNDA server on Ubuntu using the concept of VMotion, enabling high availability and fault tolerance.

**CPE 1.3 Intuitive customer facing interface.** A web-based front-end GUI will be developed to offer the customer an intuitive management interface for the overall system. The user will be able to access the server with all collected data with the option for customizable web dashboards to reflect the telemetry analysis.

*Logistical*

**CPE 2.1 Access to CU Boulder’s ITP telecommunications laboratory and equipment.** In order to complete the proof of concept, access to the on campus facilities with the proper networking devices and resources is sine qua non for the success of the project.

*Non-technical*

**CPE 3.1 Policy and business strategy experience and research:** In order to develop the proposed policy and business analysis, experience and access to research will be necessary to make the proper financial forecasts and policy implications analyses.

*Financial*

Open source software and an original Python framework result in no critical financial elements to develop a proof of concept.

### 3.0 Design Requirements

Aligned with the technical requirements of the project, the following Functional Requirements (FNC.X) along with its Design Requirements (DES X.X) are organized as follows:

**FNC.1** Develop a Python based framework able to pull data from the underlying network and push it to the PNDA server.

**DES 1.1** The framework will be a Python script running as software on a server.

**DES 1.2** It must orchestrate the pulling of information from both traditional and SDN, and send it to the PNDA server for storage.

**DES 1.2.1** The telemetry collection function of the algorithm must be scalable and reliable. Scalability and reliability will be assessed by simulating via code new devices in the network and testing the performance.

**DES 1.3** It must communicate with the following components of the system:

**DES 1.3.1** Full Duplex logical link to the PNDA server.

**DES 1.3.2** Full Duplex logical link to the web GUI.
**DES 1.3.3** Full Duplex logical link to the SDN controller.

**DES 1.3.4** Full Duplex logical link to the traditional network devices.

**FNC.2 PNDA** will be running on its own separate server.

Ubuntu would be used as the operating system, and PNDA would be deployed over it. This setup would be replicated on another server which would maintain states of reads and writes for the database. This would provide high availability and fault tolerance between our applications.

**FNC.3 Scheme** a web-based Graphical User Interface (GUI) interface:

**DES 3.1** The user must have visibility to the stages the framework is running. These stages are defined from data collection to analysis. Flask would be used to create the web UI and Bootstrap for dynamic pages.

**DES 3.2** The user will select from a list of available networking decisions (Throughput, Latency etc.) which the system must perform.

### 4.0 Key Design Options Considered

For the system components illustrated in Figure 1, it is necessary to study the existing design options in the industry in order to make an educated decision on which variation to use for the success of the project within the requirements.

**3.1 SDN Controller**

The SDN controller acts as a logically centralized command and control station for the underlying network devices and links between them. It resides on the Control Layer of the SDN Layered Architecture (see Figure 3) and serves as the Operating System (OS) for the entire network [5]. When performing data collection from an SDN, the controller acts as a central source of telemetry data, becoming one of the targets of the Python framework. Choosing the right SDN controller to build a proof of concept is paramount to the ability to scale the project and expand its capabilities.

![Figure 3. SDN Layered Architecture](Source: [5])
The options considered for analysis are the following:

3.1 **Open Network Operating System (ONOS):** ONOS is an open source Java-based OS hosted by the Linux Foundation which can act as the controller for a SDN architecture. It also comes equipped with a web-based GUI destined to control and monitor flows in the network which are reflected in real time. It supports applications to control and retrieve SDN-flows such as REST API, gRPC, RESTCONF, etc.

3.1.2 **Floodlight:** Floodlight is a Java-based OpenFlow controller forked from beacon. It has a set of REST APIs which can be leveraged to control and retrieve SDN-flows. An aggregated to Floodlight is its compatibility with OpenStack’s cloud orchestration platform.

3.1.3 **OpenDayLight (ODL):** ODL is an open source Java-based controller. Similar to Floodlight, it supports OpenFlow. It is also equipped with a web-based GUI to monitor and control the network.

3.1.4 **RYU:** RYU is a Python based open source SDN controller designed specifically to improve the agility of the network. The fact that the base language is Python makes it an ideal solution for programmable networking purposes. As Python is acknowledged as a developer friendly language, RYU applications are easy to develop and manipulate.

3.2 **Programming Language:**

The programming language for development and maintenance plays a crucial role in productivity and efficiency of a networking project. There are a variety of programming language that have been designed with different requirements in mind. Some languages, like Python, are structured thinking of a developer friendly solution, while languages like C++/C# are high-performance oriented. Moreover, the selected language has to support the features to be incorporated as part of the project, which is why a comparative study is helpful to determine if the project has been built upon the right CONOPS.

3.2.1 **Python:** Python is defined as an interpreted language. It can make efficient use of generators, which are iterators computed in a single pass. They do not require any additional space in the process memory model. While Python’s space complexity is one of the most efficient models, the trade off is time efficiency. This makes Python an user-friendly option with a slower execution compared to other state of the art solutions. However, Python also comes with the advantage of being the language
PNDA was written on, providing an easy interface to couple with an aggregate Python framework.

3.2.2 Java: Java is a platform independent, compiled programming language. It is regarded as an easy to learn resource. Java is also highly portable and is equipped with powerful development tools such as Eclipse SDK. Where Python is an user friendly option due to its space complexity, Java is best suited for production environments due to its object-oriented fundamentals. It’s also important to note that Java needs a Java Virtual Machine (JVM) to run on.

3.2.3 Go: Go is described as a “fast, statically typed, compiled language that feels like dynamically typed, interpreted language” [6]. The main benefit to Go is its time complexity as it’s known to be have a fast compiler. However, it is not supported by PNDA, which is a design driver of this project.

3.3 Data Collection Tools:

Since part of the FNCs for this project is the functionality to retrieve data logs and key parameters from underlying networks, the selection of an appropriate tool will define the flexibility of the project. The resource used to run this task has to be supported by most vendors, provide consistent data retrieval and perform at reasonable speeds. Two methods the better aligned with these goals:

3.3.1 Paramiko – Out of Band IP address: Python has a Paramiko library used to SSH into devices using an out of band IP address. However, one of its major tradeoffs is speed. Paramiko uses SSHv2, known to have slower speeds due to higher encryption policies. Another issue that could arise would be timeout. SSH uses a longer timeout to determine the out of band IP address is unreachable, which can cause degradation in performance in case of failover switches. Parsing is another aspect that would need to be treated with care. It can lead to exponential inefficiencies in the case of log collection performing at a slower rate.

3.3.2 APIs: all vendors have built-in support for REST APIs within their devices, which would provide the parameters to be collected. This is done by not parsing logs but maintaining an illusion of a File System (FS) called the proc file system. The proc file system monitors for real-time data and has a faster time to output. One of the disadvantages of this method is that it wouldn’t support legacy devices that do not come equipped with REST API support.

3.3 Network Parameters:

Going off of the assumption that most networks nowadays are handled using Border Gateway Protocol (BGP), the baseline for network parameters to optimize set to achieve minimum level of success are as follows:
3.3.1 **Latency:** for purposes of inter-ISP communication, BGP selects the shortest path based on Autonomous System. However, within the AS there might exist a route which leads to different end-to-end latency. The intention would be to provide the user with the option of performing a networking decision based on latency.

3.3.2 **CPU Throughput:** as soon as CPU utilization spikes, a BGP session starts fluctuating and routers start dropping control packets, resulting in loss of integrity. A networking decision based on the path with lowest CPU utilization of the AS edge router would prevent this scenario from happening.

3.3.3 **Link Utilization:** given different paths for networking, each will perform at a different speed. BGP is designed to select the shortest path based on the number of AS. In case a link in the vicinity goes down for another traffic, all traffic will be moved to a single link which makes it vulnerable to over-utilization. This results in traffic degradation. A decision based on lowest link utilization would provide a solution in which an unused link will be utilized for this scenario.

The above parameters provide solutions that are not currently being leveraged in BGP to optimize networking decisions.

3.4 **PNDA:**

While PNDA is a fixed resource in the design of this project, it is also a key design parameter, which is the reason why an overview of it is in order.

PNDA is an open source big data tool designed for the storage and analysis of network data. It is managed by the Linux Foundation and comes with built in support for platforms such as Open Day Light. Since this project is driven by data collection from both SDN and traditional networks, it must support large quantity and high complexity of data. PNDA offers a solution that aligns with the project’s needs, specifically the scalability to large amounts of data collected per unit of time.

The nature of big data is that it must be analyzed in clusters and synchronized with the multithreaded driving algorithm of a centralized script, which in our case is a Python framework. PNDA supports this multithreaded approach through its multi-process lifecycle management.

Additionally, it provides room for growth in the number of additional capabilities the project might acquire given the first level of success is attained in a convenient timeline. These additional capabilities include data collection process and high availability without compromising performance of the system. PNDA supports data sharing and maintaining state between hosting servers which ensures high availability.
5.0 Trade Study Process and Results

This section explains the selection and tradeoffs resulting in analyzing options in a quantitative metric fashion in order to select the better suited resources that align with the project’s objectives. Table 1 is intended to provide a standard of parameters considered crucial to the success of the project in different scales (reflected in the ‘weights’ column). Then, it will be applied to all options.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Weight</th>
<th>Scale (1-5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficiency</td>
<td>The efficiency of the system depends on the type of the programming language used. The language which has APIs supported by the SDN controllers and PNDA would be beneficial for the data retrieval and decision making. Since the project involves different technologies integrating them with a common framework becomes crucial.</td>
<td>35%</td>
<td>1: Inefficient 5: Powerful and Compatible</td>
</tr>
<tr>
<td>Time/Frequency of fetching operation.</td>
<td>It becomes critical to decide the frequency of the fetching operation in order to respond dynamically to the changes. The code should be devised in a way to synchronize between fetching and decision making. Input and output frequency.</td>
<td>30%</td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td>The complexity of the algorithm is related to the type of data fetched and the challenges faced in its manipulation. For example, it would be relatively easy to find the best path using parameter like CPU utilization etc. than by using link metrics between different Autonomous Systems.</td>
<td>20%</td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td>Server acts as the host to the software which is responsible for data retrieval, interaction with PNDA and the web GUI. The specifications of the server are critical for the speed of the overall system. The efficiency of the</td>
<td>10%</td>
<td>1: Inefficient</td>
</tr>
</tbody>
</table>
The project is aimed to utilize open source software and virtualized environment. Features that are not supported in free software and virtualized environment can be deployed using commercial software and hardware equipment.

<table>
<thead>
<tr>
<th>Cost</th>
<th>Description</th>
<th>5%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The project is aimed to utilize open source software and virtualized</td>
<td></td>
</tr>
<tr>
<td></td>
<td>environment. Features that are not supported in free software and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>virtualized environment can be deployed using commercial software and</td>
<td></td>
</tr>
<tr>
<td></td>
<td>hardware equipment.</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Tradeoff Criteria

Now, applying this criteria to the elements in Section 4, the selection process is simplified:

5.0 SDN Controller

<table>
<thead>
<tr>
<th>Controller</th>
<th>Efficiency</th>
<th>Frequency of Data Collection</th>
<th>Complexity</th>
<th>Scalability</th>
<th>Cost</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.35</td>
<td>0.30</td>
<td>0.20</td>
<td>0.10</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>ONOS</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4.35</td>
</tr>
<tr>
<td>Floodlight</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4.35</td>
</tr>
<tr>
<td>ODL</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4.35</td>
</tr>
<tr>
<td>Ryu</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4.4</td>
</tr>
</tbody>
</table>

5.1 Programming Language

<table>
<thead>
<tr>
<th>Language</th>
<th>Efficiency</th>
<th>Frequency of Data Collection</th>
<th>Complexity</th>
<th>Scalability</th>
<th>Cost</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.35</td>
<td>0.30</td>
<td>0.20</td>
<td>0.10</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4.2</td>
</tr>
<tr>
<td>Python</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4.35</td>
</tr>
<tr>
<td>Java</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3.75</td>
</tr>
<tr>
<td>GO</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3.95</td>
</tr>
</tbody>
</table>
5.2 Data Collection Tools:

<table>
<thead>
<tr>
<th>SDN Controller</th>
<th>Efficiency</th>
<th>Frequency of Data Collection</th>
<th>Complexity</th>
<th>Scalability</th>
<th>Cost</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights</td>
<td>0.35</td>
<td>0.30</td>
<td>0.20</td>
<td>0.10</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>Paramiko</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4.2</td>
</tr>
<tr>
<td>API</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4.35</td>
</tr>
</tbody>
</table>

For the remaining design consideration in Section 4, a tradeoff study is not necessary.

6.0 Selection of Baseline:

After taking into considerations all options as described in Section 4, and doing a qualitative study in Section 5 in order to select the better suited option, these are the results:

6.1 Ryu Controller:

The RYU SDN controller scored the highest in the tradeoffs study. One of the main reasons for its success was the advantage of it being written in Python, which is both our design language and PNDA’s source language. RYU will provide all required interface capabilities.

6.2 Python:

Python has been a strong preference as the programming language since the beginning stages of this project’s design. However, the tradeoff study served to further reinforce Python as the chosen language since it uses less complexity space and can be easily interfaced with hardware devices. Moreover, it supports a number of web-based GUIs, which can be tailored to this project’s needs. Finally, it’s the language in which PNDA was written and it’s a developer friendly resource.

6.3 Data Collection Tools:

APIs scored the highest on the tool to be used for telemetry data collection from traditional and SDN. APIs used in traditional networking perform at high speeds and low latency which helps with failover. Additionally, they are not dependent on SSH for compatibility. In the case of SDN, APIs come shipped with the RYU controller.

6.4 Network Parameters:

The user would be provided with three network parameters (CPU utilization, latency and link utilization) to select from via the web based UI.

6.5 PNDA:

PNDA remains the big data tool of preference.
7.0 References


