Designing a Cloud-Based DOCSIS Time Protocol Calibration Database

A Technical Paper prepared for SCTE by

Roy Sun
Ph.D., Lead Architect
CableLabs
858 Coal Creek Cir, Louisville CO, 80027
303-661-6789
r.sun@cablelabs.com

Rahil Gandotra, Ph.D., and Mark Poletti, CableLabs, Inc.

Jennifer Andreoli-Fang, Ph.D., Amazon Web Services (AWS)

Elias Chavarria Reyes, Ph.D., Hitron Technologies, Inc.

John Chapman, Cisco Systems, Inc.

Table of Contents

Title **Page Number**

List of Figures

Title **Page Number**

List of Tables

Title **Page Number**

1. Introduction

This paper will detail the design and implementation of a cloud-based application to enable DOCSIS Time Protocol (DTP) deployment. This application is unique to the cable environment and its implementation can serve as a reference design for future cable cloud applications.

The key components of the app will be a database and an application programming interface (API). The application can be implemented on multiple cloud platforms. We chose to prototype our application on Amazon Web Services (AWS). The architecture design will follow the AWS Well-Architected Framework to leverage the benefit of the cloud and will provide horizontal scalability, pay-as-you-go cost structure, and high availability. The application will also be designed for service assurance, which is critical in field deployments. All cloud-based components and a CMTS emulator with the API client will be implemented on AWS.

DTP was invented in 2011 by John Chapman of Cisco [1], in anticipation of using the DOCSIS network to provide timing as a service (TaaS). In 2012, DTP was standardized as part of DOCSIS 3.1 [2]. This version of DTP defined the core algorithm and functionality. Since the primary use case for DTP is mobile backhaul over DOCSIS, CableLabs introduced the SYNC specification [3] in 2020 to address this use case. As part of the SYNC specification initiative, Elias Chavarria and John Chapman from Cisco redesigned the DTP algorithm to bypass a set of limiting assumptions in the original DTP design [4]. Also, in 2020, a group of companies, including Cisco, Hitron, Charter, and CableLabs, did a proof of concept (PoC) to validate the performance of DTP. The test results of the proof of concept are reported in a separate Society of Cable Telecommunications Engineers (SCTE) paper this year [5] and a CableLabs technical report [6].

At its core, DTP allows a DOCSIS network to interface its native timing and frequency to external timing protocols. To do this, DTP establishes a set of techniques and DOCSIS signaling messages between the cable modem (CM) and the cable modem termination system (CMTS).

In the original DTP design, the DTP messages contained CM and CMTS timing parameters. The underlying assumption was that the timing parameters for the CM could be measured separately from the CMTS ones before deployment. The timing parameters in the CMTS and CM would have been scalable in this distributed measurement model. For the assumption to be valid, a testing device capable of measuring those timing parameters needs to exist. However, no such testing device exists. Therefore, the usability of the original DTP design was limited. For this reason, DTP was redesigned as part of the SYNC specification effort [2].

In the redesigned DTP, the timing parameters are no longer measured separately for the CM and the CMTS, they are instead measured jointly. With this change, existing testing devices in the market can be used. A consequence of this change is that the number of measurements grows from linear to exponential. If n number of CMTS products and m number of CM products support DTP, the original DTP design required n+m measurements of timing parameters. The redesigned DTP calls for $n * m$ measurements of timing parameters.

The values of the DTP timing parameters that a DOCSIS network should use also depend on the CMTS and CM configuration, e.g. the interleaver configuration. For example, if there are w number of different interleaver configurations, the total number of measurements of timing parameters grows to $n * m * w$. In Phase 2 of the DTP PoC that Cisco, Hitron, Charter, and CableLabs are conducting [5], they will assess the impact of other configuration parameters, e.g., modulation, cyclic prefix, and frame size. If all these configuration parameters impact the DTP timing parameters, the total number of measurements will continue growing exponentially. Assuming x different modulations, y different cyclic prefixes, and z different frame size, a total of $n * m * w * x * y * z$ timing measurements would be required.

The cable industry has two options to handle the measurement, storage, update, and accessibility of all the DTP timing measurements. One option is for every CM and CMTS vendor to do everything by itself, which would lead to a replication of effort and resource investments with little added value for the vendors. The second option is for a common entity, such as CableLabs, to lead the measurement, storage, update, and accessibility of the DTP timing parameters. This second option allows the cable ecosystem – both vendors and operators – to leverage a shared pool of resources. The opinion of the authors is that the second option is more feasible for the cable industry. To make this second option a reality, the authors propose a cloud application, as discussed in the following sections.

The DTP calibration includes three major steps: 1) collect the calibration data; 2) build a cloud app that distributes the data; and 3) a CMTS to access and apply the calibration data. The DTP calibration test could be conducted in many test labs. For example, CableLabs/Kyrio established a Network Timing Lab that could evaluate the DTP performance and collect the calibration data. One of the key contributions of this paper is to present the design of API and the AWS cloud server that distributes the calibration data. The CMTS will access the database to obtain calibration values in real-time via the API. Using these values, the CMTS will calculate the timing offsets for the CM and the 5G radios. CableLabs expects to continuously sponsor the application to enable the commercial deployment of DTP.

2. DTP Calibration Method

In this section, we further explain why DTP needs calibration and how to do the calibration. The DTP timing diagram is shown in Figure 1, where DS-T and US-T denote downstream (DS) and upstream (US) delays inside the CMTS, DS-H and US-H are DS and US delays in the hybrid fiber-coaxial (HFC) plant, and DS-C and US-C are DS and US delays caused by the CM. The CMTS sends the DOCSIS 3.1 timestamp. The timestamp is delayed when it arrives at the CM. In other words, the timestamp that arrived at the CM represents an early version of the CMTS timestamp. The time error (TE) is in the DS only. Ideally, DS-T, DS-H, and DS-C should be measured separately and used by DTP. However, DOCSIS does not provide the reference points to measure these delays. DTP provides a practical way to calibrate the DOCSIS 3.1 timestamp using the true ranging offset (TRO).

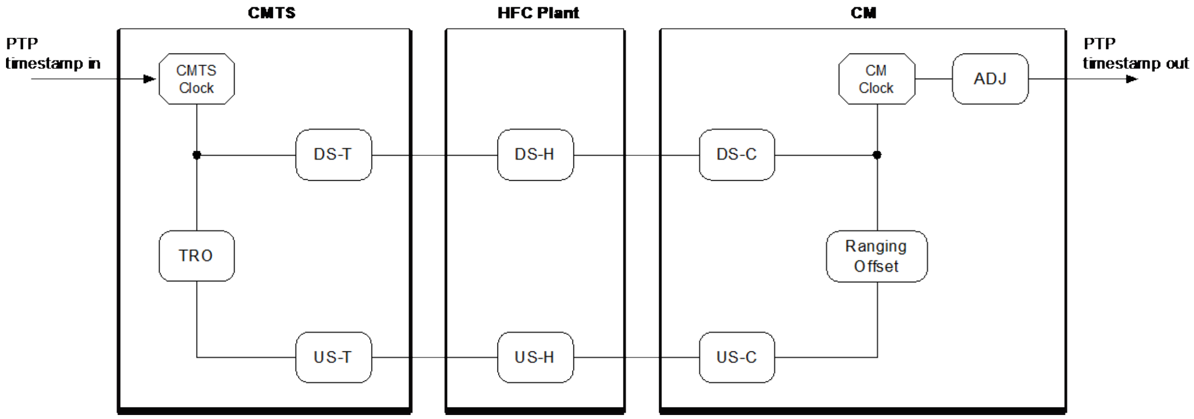


Figure 1 - DTP Timing [3]

DTP messages are exchanged between CMTS and CM, see Figure 2 for the case when the CMTS is the DTP master and the CM is the DTP slave. The CM measures the round-trip delay as the TRO. The CM reports the TRO to the CMTS in the "DTP-Response" message. The CMTS sends the time adjustment *t-adj*, or *t-cm-adj*, to the CM using the "DTP-Info" message. The *t-cm-adj* is approximately equal to half of the TRO. The CM timestamp over its CM to CPE interface (CMCI) ports is equal to the DOCSIS 3.1 timestamp plus *t-cm-adj*. The *t-cm-adj* corrects the time error in the network. Note that the propagation delay through coaxial cable and fiber is theoretically symmetrical.

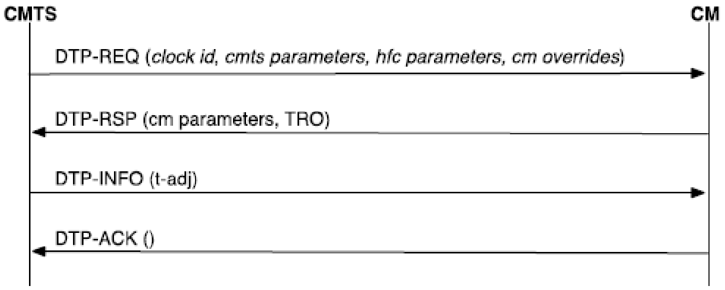

CMTS
CM
DTP-REQ (clock id, cmts parameters, hfc parameters, cm overrides)
DTP-RSP (cm parameters, TRO)
DTP-INFO (t-adj)
DTP-ACK ()

Figure 2 - DTP Message Flow [3]

If *t-cm-adj* is set to be half of the TRO, *t-cm-adj* cannot correct asymmetrical delay (different in DS and US). The asymmetrical delay is introduced by devices like CMTS, CM, remote physical RF layer (R-PHY) and remote physical and MAC layers (R-MACPHY), and any HFC elements. This asymmetrical delay needs to be measured in lab for each pair of CMTS and CM combos. The measured asymmetrical delay could be computed at the CMTS. For example, the following method is supported by the Cisco integrated CMTS (I-CMTS) cBR-8:

$$t\text{-}cm\text{-}adj = t\text{-}tro/2 + y, \tag{1}$$

where y is an additional time adjustment that applies in the CMTS to calibrate the DS delay in the DOCSIS 3.1 timestamp. This approach is described as method 2 in Section 5.4 in [6]. CM location, plant length, and other symmetrical TE are taken care of by the TRO. The asymmetrical TE is addressed by the additional time adjustment y. This additional time adjustment y can be mapped to formula (18) in the SYNC spec [3], which we copy here:

$$t\text{-}cm\text{-}adj = t\text{-}cm\text{-}adj\text{-}R + [t\text{-}tro + t\text{-}hfc\text{-}ds\text{-}o - t\text{-}hfc\text{-}us\text{-}o - t\text{-}tro\text{-}R]/2, \tag{2}$$

where *t-cm-adj* is the live time adjustment that the CMTS sends to the CM, while *t-cm-adj-R* is the value of the DTP time adjustment used in the calibration test that brings the average PTP two-way time error to zero [3]. Similarly, *t-tro* is the live TRO that the CM measures and sends to the CMTS. *t-tro-R* is the TRO reported by the CM in the lab calibration test. *t-hfc-ds-o* and *t-hfc-ds-o* represent any fixed delay elements in the HFC plant that contribute to delay [2], in DS and US, respectively. *t-hfc-ds-o* and *t-hfc-ds-o* are provided by the CMTS to the CM.

The above formula can be rearranged as:

$$t\text{-}cm\text{-}adj = t\text{-}tro/2 + [t\text{-}cm\text{-}adj\text{-}R + t\text{-}hfc\text{-}ds\text{-}o/2 - t\text{-}hfc\text{-}us\text{-}o/2 - t\text{-}tro\text{-}R/2], \tag{3}$$

Comparing Eq. (3) to Eq. (1), we get that:

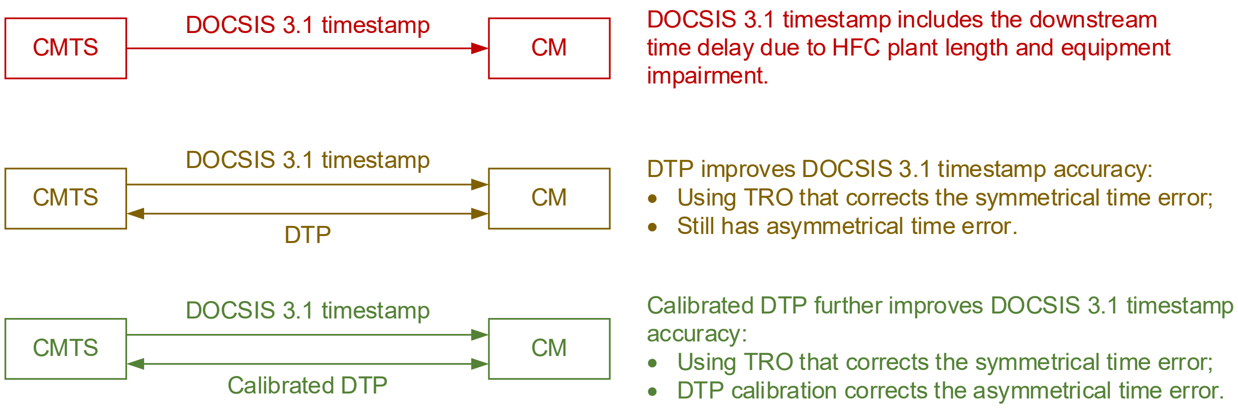
$$y = t\text{-}cm\text{-}adj\text{-}R + t\text{-}hfc\text{-}ds\text{-}o/2 - t\text{-}hfc\text{-}us\text{-}o/2 - t\text{-}tro\text{-}R/2. \tag{4}$$

Note that if the HFC plant is assumed to introduce no asymmetry, i.e., *t-hfc-ds-o/2* = *t-hfc-us-o/2*, then Eq. (4) is further simplified as:

$$y = t\text{-}cm\text{-}adj\text{-}R - t\text{-}tro\text{-}R/2. \tag{5}$$

Eq. (5) captures the asymmetry in the reference-length plant (see Section 6.4.1.1 in [3]) introduced jointly by the I-CMTS and CM (in an I-CMTS architecture), or jointly by the RPD and the CM (in an R-PHY architecture). In DTP calibration, the values for *t-cm-adj-R* and *t-tro-R* that make the average PTP two-way time error to zero will be measured in the lab and the additional time adjustment *y* will be distributed by the AWS database. *t-tro* and *t-cm-adj* in Eq. (1) will be calculated lively in the field.

In summary, as shown in Figure 3, DOCSIS 3.1 timestamp is used in DOCSIS 3.1 networks that include a delay in DS. DTP uses TRO to correct the symmetrical part of the DS delay. DTP calibration further corrects the asymmetrical part of the DS delay. For example, DTP calibration reduced the time error from 3,223,800 ns to 13-31 ns as reported in Section 8.4 in [6].



DOCSIS 3.1 timestamp includes the downstream time delay due to HFC plant length and equipment impairment.

DTP improves DOCSIS 3.1 timestamp accuracy:

- Using TRO that corrects the symmetrical time error;
- Still has asymmetrical time error.

Calibrated DTP further improves DOCSIS 3.1 timestamp accuracy:

- Using TRO that corrects the symmetrical time error;
- DTP calibration corrects the asymmetrical time error.

Figure 3 - DOCSIS 3.1 Timestamp, DTP and DTP Calibration

3. DTP Calibration Cloud Database Design

On a high-level, the primary components of the DTP calibration cloud database are – a database, a web-based graphical user interface (GUI) to provide a human interface for the lab engineer to add, read, and delete the DTP calibration entries, and an API framework to provide a machine interface for the CMTS to fetch the DTP calibration entries. Since the components to be implemented were functionally fairly uncomplicated, the decision to select a cloud provider was also relatively uncomplicated. AWS was selected as the cloud provider for DTP calibration since it is the most mature and enterprise-ready provider [7].

AWS provides multiple different methods to implement the abovementioned three components depending on the amount of modularity required in the system design. Figure 4 shows three possible system designs to implement a database, a web GUI, and an API in AWS with varying levels of modularity. Figure 4(a) illustrates implementing the three components within a single compute instance. While this approach provides vertical scaling capabilities, it offers limited flexibility and availability benefits. In Figure 4(b), the database is transitioned out to its own module while the web GUI and API reside on a single compute instance. Disaggregating the database offers added benefits of leveraging a database management system to abstract away the setup, operation, and scaling tasks. In Figure 4(c), all three components are implements in their own module. This approach provides the most modularity in terms of allowing independent management and innovation of each component while hiding the complexity of each part behind an abstraction and interface.

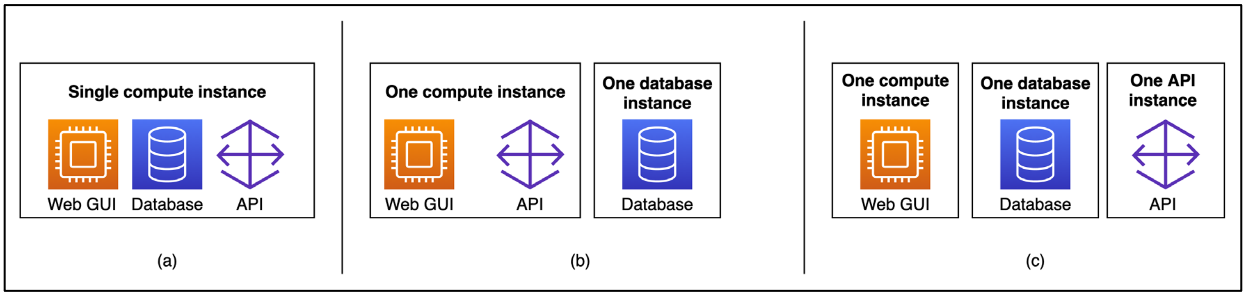


Figure 4 - Possible System Designs of the DTP Calibration Cloud Database

The complete framework of the DTP calibration cloud database is shown in Figure 5. The database is hosted using Amazon Relational Database Service (RDS), the web GUI is running on an Elastic Compute Cloud (EC2) instance, while the API framework includes Amazon API Gateway as the frontend and AWS Lambda service as the backend. To validate the end-to-end functionality of the cloud application, a CMTS emulator was developed to test sending requests and receiving responses from the cloud application.

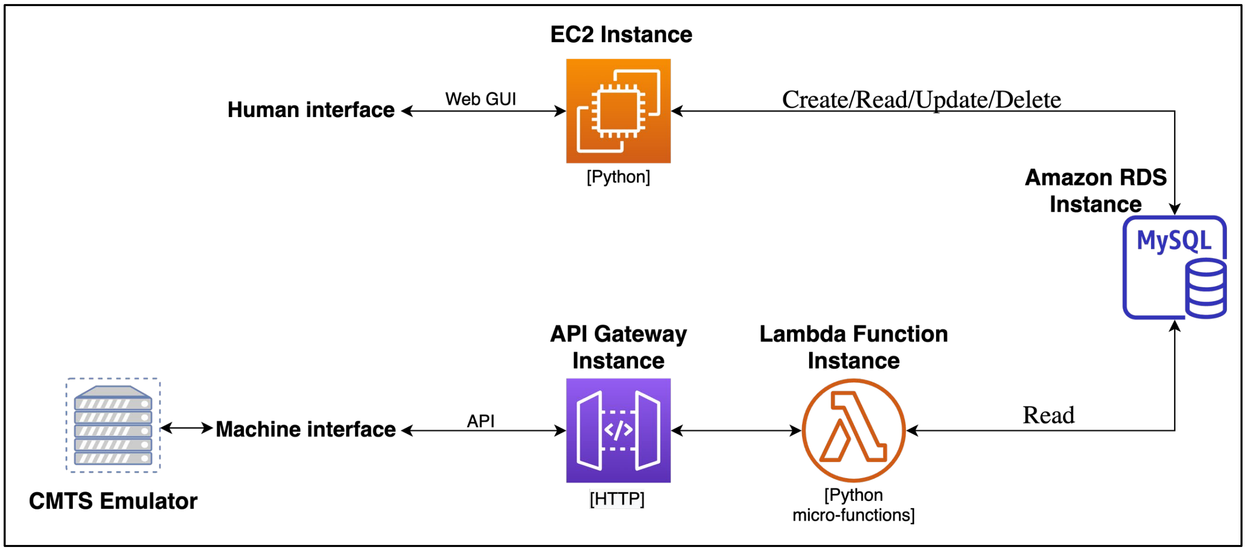


Figure 5 - Functional Design Diagram of DTP Cloud Calibration Database

Since the structure of the data to be stored in the database is not expected to change frequently, a relational database was selected as the database type. Amongst the different relational database engines available, MySQL was chosen as it is open-source and provides sufficient flexibility to run on any operating system. A Python script was developed to instantiate and create the database schema allowing for any possible changes in the future. Database reliability could be enhanced by utilizing the RDS Multi-AZ (Availability Zone) functionality provided by AWS wherein a standby database instance is automatically created in another AZ and data is synchronously replicated between the two instances.

The web GUI is implemented in Python using Flask [8]. Flask was selected as the web framework as it allows for easy addition of libraries or plugins for an extension and comes with a built-in development server and fast debugger. Additional modules for handling forms and enabling login using username and passwords were implemented using Flask extensions. The web GUI provides the user with the capability to add new calibration entries to the database, read existing entries from the database, update existing entries in the database, and delete any existing entry from the database.

AWS offers two types of API Gateways – HTTP-based and websocket-based. HTTP APIs were selected as the API Gateway type since the communication between the CMTS and the cloud application is expected to be stateless and not based on stateful real-time two-way communication applications such as chat apps or streaming dashboards which require websocket APIs. Amazon's HTTP API Gateway provides API proxy functionality and low-latency, cost-effective integrations with other AWS services. Both HTTP GET and POST methods are implemented to enable sending query parameters via the URL of the GET request and the request body of the POST message. Since the connection between the CMTS and the cloud application is not expected to be persistent and would be infrequent (only when there is a new configuration pairing of CMTS-CM), the API backend also does not require persistent compute. Therefore, serverless Lambda functions are used in the API backend to consume compute resources only when needed – in case of an incoming request from a CMTS. The API Gateway is configured to pass the query/payload received from the client to the Lambda function and return the function's response to the client. A Lambda function is developed to run a database read query based on the received query/payload parameters (CMTS-CM Hardware-Firmware versions) and return the result (timing parameters) along with a valid HTTP status code.

The CMTS emulator is essentially an API client developed to test the responses of the cloud application by sending HTTP requests to the API Gateway with different Hardware-Firmware combinations. The content-type is set to application/json for these requests and responses. JSON is used as the payload format as it is lightweight and is suitable for both human reading and machine parsing. The motivation behind developing this emulator is to demonstrate the work needed by CMTS vendors to enable the remote collection of timing parameters from the cloud application.

The current application design also supports adding calibration entries, in addition to reading them, using the API Gateway and Lambda function framework. This would allow automation in the calibration process wherein a large number of timing entries could be added in the database without requiring manual work. Application security is considered at two levels: (i) API level: Access to the API can be restricted by either using HTTP request parameters-based authorization (such as username/password) or by using token-based authorization (such as JSON Web Token, JWT), and (ii) Network level: Access to the virtual network where the API framework is hosted can be restricted to known CMTS IP addresses only.

4. DTP Calibration Cloud Database API Design

4.1. Cloud App Message Flow

The message flow between the AWS server and CMTS is HTTP-based, as shown in Figure 6. HTTP uses TCP as transport layer to provide reliable network transmission using acknowledgments. Our DTP API uses "HTTP request" and "HTTP response" to exchange information between the CMTS and AWS server. The CMTS (client) sends Cal-data Request to the server that is contained in the HTTP Request. The Cal-data Request message includes network architecture and hardware/firmware combinations, see Section 4.2. The server sends Cal-data Response to the client that is contained in the HTTP Response. The data elements contained in the Cal-data Response message is described in Section 4.2.

The client sets a timer (i.e., 500 ms) using a status code when sending the Cal-data Request message. The client checks the status code and the received calibration data. If failure, the client resends the Cal-data Request message. The AWS server may provide multiple IP addresses for redundancy, and the client may try another server IP address if failure over five times. There could be three failure reasons: 1) timer expired; 2) network architecture and Hardware/Firmware combinations do not match requested data; and 3) calibration data (test configurations or measured time error results) incomplete.

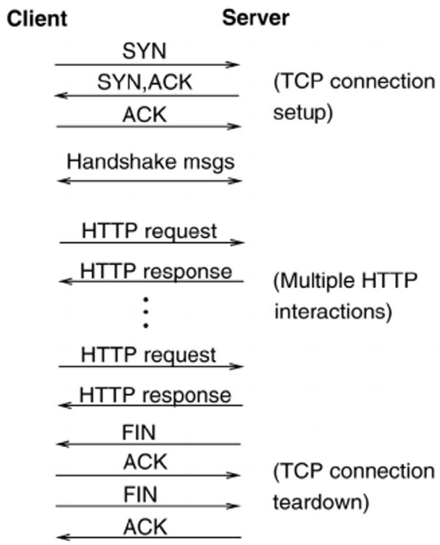


Figure 6 - HTTP and DTP Calibration Message Flow

4.2. Cloud App Data Structure

The data structure is illustrated in Table 1. Cal-data Request message only includes the first two data elements: network architecture and hardware/firmware combinations. Cal-data Response message includes all elements listed in Table 1. DTP could be deployed on the traditional network with I-CMTS and distributed access architecture (DAA). DAA also includes remote-PHY and remote -MACPHY. Network architecture is represented by two bits. Hardware and firmware combinations includes make, hardware and firmware version of I-CMTS chassis, I-CMTS line card, RPD (for DAA only), and CM.

Test configurations considered in the initial version of the cloud app include DS modulation scheme, DS interleaver, DS cyclic prefix, US modulation scheme, US cyclic prefix, and US frame size. DTP PoC [5] phase 2 testing is evaluating the impact of each parameter. Parameters that do not strongly impact DTP performance will be removed from Cal-data Response message. Other parameters that strongly influence DTP performance but are not listed above will be added to the Cal-data Response message.

Table 1 - Example Data Structure.

Network Architecture	Hardware & firmware combinations	Testing lab	Testing date	Test configurations	Additional time adjustment y (ns)	Constant Time Error (ns)
I-CMTS	Combo 1	CableLabs	11/1/2020	Config 1	200,000,000	-50
			11/1/2020		300,000,000	…
			11/2/2020	Config N	400,000,000	100
DAA	Combo 2	CableLabs	12/1/2020	Config 1	200,000,000	50
			12/1/2020		300,000,000	…
			12/1/2020	Config N	400,000,000	-100

5. Conclusion

4G/5G mobile networks require a high-accuracy synchronization source in the backhaul where the GPS signals are unavailable. DTP provides such sync signals in the backhaul over HFC networks. DTP needs automated calibration in the field to guarantee time accuracy.

This paper presented the DTP calibration method and design of a cloud app that distributes the calibration data. The cloud app is prototyped on AWS. A modular application design is developed allowing to leverage various cloud benefits such as abstraction, automation, and high-availability. The web GUI is implemented in Python using Flask allowing an engineer to add, read, and delete the DTP calibration data entries. The API uses HTTP protocol with JSON as the data format, and calibration data message flow was designed. Security and reliability enhancement features are considered and will be added based on costumers' requirements. Future work of automated DTP calibration includes collecting calibration data in test labs and adding them into the AWS database via the Web GUI. CMTS will need to add the corresponding feature to inquiry and apply the calibration data automatically. Proof-of-concept test for the AWS cloud app and automated DTP calibration is planned in the near future [5].

Abbreviations

API	application programming interface
AWS	Amazon Web Services
AZ	availability zone
CM	cable modem
CMTS	cable modem termination system
cTE	constant time error
DAA	distributed access architecture
DOCSIS	Data-Over-Cable Service Interface Specification
DS	downstream
DTP	DOCSIS Time Protocol
EC2	elastic compute cloud
GUI	graphical user interface
HFC	hybrid fiber-coaxial
I-CMTS	integrated cable modem termination system
PoC	proof of concept
RDS	Relational Database Service
RPD	remote physical layer device
R-PHY	remote physical RF layer
SCTE	Society of Cable Telecommunications Engineers
TaaS	timing as a service
TE	time error
TRO	true ranging offset
US	upstream

Bibliography & References

[1] John T. Chapman, Rakesh Chopra, Laurent Montini., “The DOCSIS® Timing Protocol (DTP), Generating Precision Timing Services from a DOCSIS System,” *INTX/SCTE Spring Technical Forum*, 2011. [link]

[2] Cable Television Laboratories, Inc., “DOCSIS® MAC and Upper Layer Protocols Interface Specification”, CM-SP-MULPI, December 2020. [link]

[3] Cable Television Laboratories, Inc., “Synchronization Techniques for DOCSIS® Technology Specification,” CM-SP-SYNC, April 2021. [link]

[4] Elias Chavarria Reyes, John T. Chapman, “How the DOCSIS Time Protocol makes the SYNC Specification Tick,” SCTE Cable-Tec Expo Fall Technical Forum, Denver, October 2020. [link]

[5] Ruoyu Sun, Jennifer Andreoli-Fang, Elias Chavarria Reyes, John T. Chapman, et al., “DOCSIS Time Protocol Proof of Concept,” in SCTE-Expo 2021, Atlanta, GA, October 11-14, 2021.

[6] Cable Television Laboratories, Inc., “DOCSIS Time Protocol Proof of Concept Phase I Technical Report CM-TR-DTP-V01-210915,” September, 2021. [link]

[7] M. A. Kamal, H. W. Raza, M. M. Alam, and M. M. Su’ud, “Highlight the features of AWS, GCP and Microsoft Azure that have an impact when choosing a cloud service provider,” International Journal of Recent Technology and Engineering (IJRTE), vol. 8, no. 5, Jan. 2020.

[8] “Welcome to Flask.” Accessed on: Jul. 12, 2021. [Online]. Available: https://flask-doc.readthedocs.io/en/latest/.