

An efficient framework to investigate the root cause of SLO violation in 5G Online Charging Systems

Kien Hoang Dang *

*Payment Platform Development Center
Viettel High Technology Industries
Corporation - Viettel Group
Ha Noi, VietNam
kienhd2@viettel.com.vn*

Hieu Cong Nghia

*Payment Platform Development Center
Viettel High Technology Industries
Corporation - Viettel Group
Ha Noi, VietNam*

Minh Nguyen Ngoc

*Payment Platform Development Center
Viettel High Technology Industries
Corporation - Viettel Group
Ha Noi, VietNam*

Che Bui Duc

*Payment Platform Development Center
Viettel High Technology Industries
Corporation - Viettel Group
Ha Noi, VietNam*

Duc Nguyen Minh

*Payment Platform Development Center
Viettel High Technology Industries
Corporation - Viettel Group
Ha Noi, VietNam*

Thuyet Tran Van

*Payment Platform Development Center
Viettel High Technology Industries
Corporation - Viettel Group
Ha Noi, VietNam*

Thai Nguyen Trong

*Viettel Networks – Viettel Group
Ha Noi, VietNam*

Abstract—Rapid deployment of 5G networks presents challenges in ensuring service quality and reliability. Service Level Objective (SLO) violation incidents critically affect 5G network functions, resulting in service interruptions, revenue loss, and reduced customer satisfaction. These incidents can arise from network congestion, misconfigurations, and hardware/software errors. SLO violations can be quickly detected through metrics and alerting rules; however, an efficient solution is required to investigate the root cause and restore the system to a healthy state. In this paper, we propose a generalizable framework for root cause investigation that enables timely analysis of SLO violation incidents in 5G Online Charging Systems, including the Policy Control Function (PCF) and the Charging Function (CHF). The framework integrates with multiple data sources to provide topology-based context and collect information about related objects. It also incorporates an analytic service for time series pattern recognition and anomaly detection based on service profiles. Experimental results demonstrate that the framework helps operators identify incident root causes efficiently.

Index Terms—5G, Online Charging System, SLO violation investigation, root cause analysis

I. INTRODUCTION

5G Network is an evolution of wireless communication technologies that aims to deliver ultra-high-speed data transmission, ultra-low latency, and massive connectivity, thus generating a wide range of new services. Up to 100 times faster than 4G [1], it drives innovation in fields such as healthcare, transportation, and the Internet of Things, ultimately reshaping global digital connectivity.

The charging function (CHF) and the policy control function (PCF) are two crucial network functions that work closely

together in the 5G core network [2]. While the charging function (CHF) manages online and offline charging for network usage, the policy control function (PCF) is responsible for policy decisions that govern how network resources are used by subscribers, applications, and services. Those two functions can be combined to enable QoS differentiation, monetize 5G services, and support flexible business models such as network slicing, IoT billing, and application-aware charging.

Our Online Charging System (OCS) was built based on 3GPP's standards and implements the PCF and CHF of 5G networks. It was deployed in the form of many Container Network Functions (CNFs) whose life cycle are managed by the NFV MANO (Management and Orchestration) platform [3]. An example architecture of a CNF in our Online Charging System is illustrated in Fig. 1. The CNF receives charging and policy control requests from other Network Functions via the Diameter Routing Agent (DRA) [4] and Service Communication Proxy (SCP) [5] to manage bandwidth and apply user tariffs. Diameter requests from the DRA and RESTful requests from SCP are delivered to our system through standard interfaces (N7, N40, Gx, Gy, Rx, etc.) as defined by 3GPP. The gateway for each interface will transform the received message to an internal format and forward it to the corresponding CHF/PCF to charge tariff or control policy. All customer profile updates in the database will be performed by the Account Balance Management Function (ABMF), which is responsible for tracking and managing the remaining balance or credit available to users.

In a system, SLI (Service Level Indicator) is a defined quantitative measure of some aspect of the level of service that is provided [6]. Not all metrics collected should be

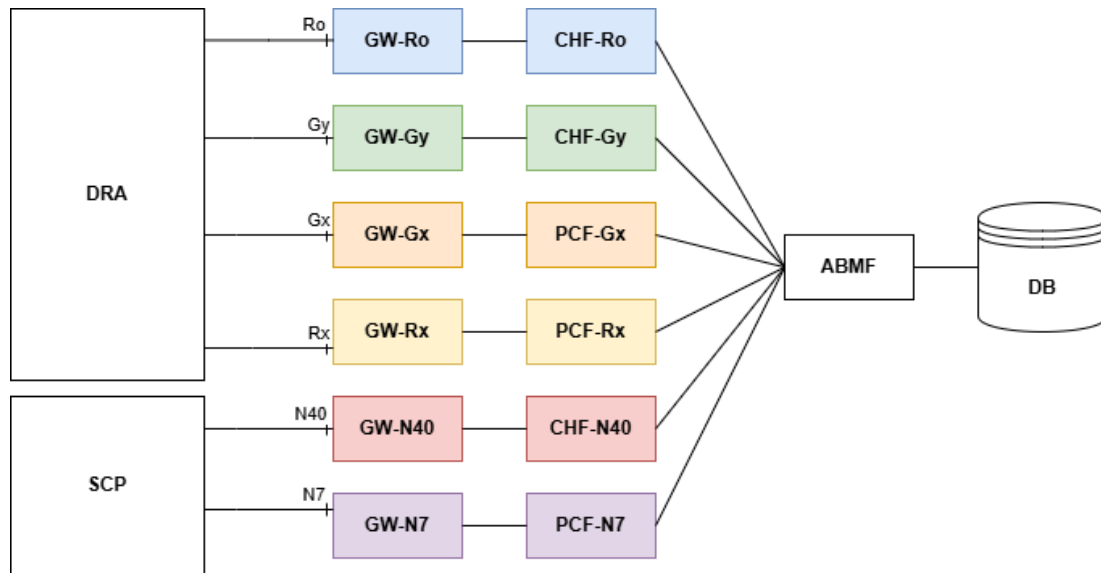


Fig. 1: Architecture of 5G charging and policy control system

considered as SLIs because we need to pay the right level of attention to the important indicators. With user-facing serving systems, some important SLIs are the availability, latency, and throughput. SLO (Service Level Objective) defines a range of values for a service level that is measured by an SLI. Some examples of SLO are the percentage of requests processed successfully by a service having to be greater than 99% or the 98th percentile of database latency having to be less than 10 ms.

An SLO violation occurs when a Service Level Indicator (SLI) falls outside its defined target range and leading to a decline in service quality. SLO violation incidents critically affect the service quality of our Online Charging System and other network functions of 5G networks, which always require almost zero downtime. It directly affects the consistency and real-time of charging and policy enforcement, leading to telecommunication service disruptions (e.g, call interruptions, loss of connection to the internet), revenue loss, and reduced customer satisfaction. SLO violations can be alerted in real-time or near-real-time using monitoring systems (e.g., using SLI metrics and alerting rules), but it is just a symptom to indicate that the system is unhealthy. To recover the system to normal operation, we need an efficient solution to locate the root cause quickly and resolve it.

An efficient root cause investigation framework will help us speed up the process of finding the root cause, and further can be integrated with an automation platform to resolve or isolate it automatically. In this paper, we will propose an efficient framework to investigate SLO violation incidents in 5G Online Charging System. Our contribution in this paper can be summarized as follows:

- An efficient framework architecture to investigate the root cause of SLO violation incidents in 5G Online Charging System using predefined playbooks

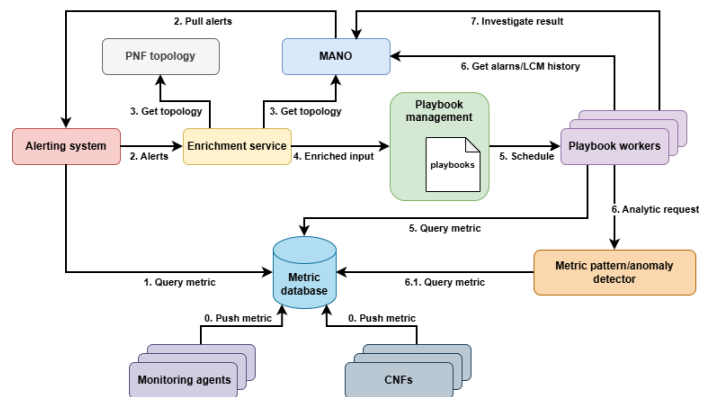


Fig. 2: Architecture of investigation framework

- Data enrichment method to provide additional contexts about objects related to the SLO violation incident
- Algorithms to detect common patterns/anomaly types in system metrics and resource utilization
- Experiments to demonstrate the efficiency of our algorithms and the investigation framework

II. INVESTIGATE FRAMEWORK ARCHITECTURE

A. Architecture overview

Our investigation process starts when an SLO violation is detected by the monitoring system, which depends on SLI metrics and alerting rules. Our investigation system can be represented in Fig. 2, which contains the following main components/services:

- **Enrichment service:** The service that provides additional context for alarms using data from the topology of existing sources in our systems, like file-based, API of **PNF Topology**, and **MANO**.

- **PNF topology:** Maintain the topology of PNFs (physical network function) and other devices in our infrastructure. An example of information in our topology is physical location and connectivity between Firewall, Load balancer, Switch, Rack, and server devices in our datacenters.
- **MANO (Management and Orchestration):** Our platform is responsible for orchestration & managing the life-cycle management and configuration of CNFs. Some types of data that are provided by the MANO platform are CNF's topology, metadata, configuration, alarms, and life-cycle management events (e.g. instantiate, scale-in, scale-out, upgrade,...) of CNFs
- **Playbook management:** The center service of our investigation framework that manages playbooks. It receives execution requests from users and schedules to run the target playbook on the Playbook workers.
- **Playbook workers:** Worker to execute playbooks which is integrated with many data sources and analytic services. The enriched data from the alarm of the previous step will become the parameters for the playbooks to run the investigation checklist that queries data from data sources and analytic services.
- **Time series database:** Store collected metrics of objects in our infrastructure, virtualization platforms, MANO, and CNFs. Metrics stored in a time series database in our system can be queried easily using PromQL [7].
- **Metric pattern/anomaly detector:** The service that is responsible for detecting anomalies/patterns of time series data (stored in the time series database). In this service, we have implemented time series pattern detection algorithms and anomaly detection methods using pre-built profiles. Several APIs are provided for playbooks and other services to send analysis requests, each API supporting customizable parameters depending on the type of metric being analyzed.
- **Other data sources:** Other types of data in our system, E.g. log storage, Other APIs (e.g. Kubernetes [8])

B. SLO violation detection

In operation, the SLI metrics that represent the health state of our services were collected continuously and stored in a time-series database, which makes it easy to query and generate alerts using monitoring rules. We are interested in four golden signals in monitoring defined by Google SRE [6], which include latency, traffic, errors, and saturation. Some example data types of each category were presented as follow:

- **Latency:** The time to process requests for each interface of our system, measured in milliseconds. Currently, we are interested in the averages, 90th, 95th, and 98th percentiles of the request process time.
- **Error:** Number of requests processed unsuccessfully in a time window. In our system, error is classified into various types: timeout, dropped, processing error codes that follow the Diameter Base Protocol [9], and other internal error codes.

- **Traffic:** The number of requests received in our interfaces, which is measured by Transactions Per Second (TPS).
- **Saturation:** The SLI that represents how "full" the system is. An example saturation metric is the average TPS of a service divided by the maximum estimated TPS that service can serve following the design document.

To detect SLO violation in our system, we focus on SLI metrics of the GWs (see fig. 1). We have built a strong monitoring system, in which almost all important errors occurring in an internal service (e.g., CHF, PCF, ABMF) will also be seen at SLIs of the gateway. We store collected metrics in a high-performance time-series database, which can query and generate alarms easily based on monitoring rules.

C. Playbook management

Each playbook is a predefined checklist that specifies the steps and actions to be performed when a type of SLO violation occurs. Steps focus on querying or fetching data from data sources or requesting analytic services. Actions are directed toward notifying or triggering outcomes, such as creating emails, sending messages, generating notifications, or opening tickets based on the results of the steps. An action is triggered when the result returned by a step matches a predefined condition, such as exceeding a threshold or producing a result from analytic services. For example, a step may query the average memory usage of a server over the last 5 minutes, and an action to send an email will be triggered if the average CPU usage exceeds 90%.

Currently, our playbook management supports querying or fetching data from the following services:

- Time series database: to query collected metrics of objects in the system
- MANO: to retrieve alarms and life-cycle management (LCM) events
- Log storage: to query log records of applications, servers, and the virtualization platform
- Data analysis service: to request analysis of time series stored in the time series database

D. Anomaly and pattern detection

Our anomaly and pattern detection service focuses on two main problems:

- **Time series pattern detection:** We use algorithms to detect patterns in time series metrics of objects in our system. Different types of metrics exhibit different signals of system health. For example, a continuous increase in a cumulative counter over time is normal, whereas a continuous increase in the memory usage of a server may indicate a memory leak. Our data analysis service currently supports detecting only common patterns in time series data. The selection of algorithms and the types of data to which they are applied are determined by the playbooks.

- **Profile-based anomaly detection:** This approach is mainly used to detect anomalies in the resource consumption of services by building resource profiles. A resource profile of a service is represented by a regression model that maps the number of requests to the amount of resources it consumes at a given time. Currently, we focus on CPU and disk I/O resources.

1) *Time series pattern detection:* We focus on four specific patterns: spikes/dips, level shifts, and trends. The solutions for each are as follows:

- **Spikes/Dips:** Spikes and dips are data points with abnormally high or low values compared to the overall distribution. We distinguish between two forms: *short* and *long*. A short spike or dip is an isolated deviation that appears at one or a few scattered indices without temporal continuity. In contrast, a long spike or dip appears as a contiguous subsequence of consecutive deviations, where values remain consistently above or below the baseline for multiple time steps before returning to normal. For short cases, we apply the traditional *z-score*, as given in Equation 1:

$$z_i = \frac{x_i - \mu}{\sigma}, \quad (1)$$

and for extreme outliers, we use the *Median Absolute Deviation (MAD)* [10], using the definition in 2:

$$MAD = \text{median}(|x_i - \text{median}(x)|). \quad (2)$$

Both methods are robust but often misclassify long spikes/dips as trends or shifts. To address this, we use the PELT algorithm [11] to segment the series, which allows us to identify long spikes/dips when the median temporarily rises or falls before returning to its previous level.

- **Level Shifts:** Level shifts represent long-lasting changes in the mean or variance of a series. When the time of change is unknown, we apply PELT to identify candidate changepoints. For each candidate, we test whether the two segments differ in distribution using the *Mann-Whitney U test* [12]. The test statistic is defined as 3-5:

$$U = \min\{U_a, U_b\}, \quad (3)$$

with its components given by

$$U_a = n_a n_b + \frac{n_a(n_a + 1)}{2} - R_a, \quad (4)$$

$$U_b = n_a n_b + \frac{n_b(n_b + 1)}{2} - R_b, \quad (5)$$

where n_a and n_b are the sample sizes of the two segments, and R_a and R_b are the sums of ranks for each segment after ranking all observations together.

A changepoint is considered a *level shift* if

$$U < U_\alpha, \quad (6)$$

where U_α is the critical value for the significance level α . Once a distributional difference is confirmed, we further

evaluate whether the shift is significant by computing both relative and absolute changes in mean values, as computed by 7 and 8:

$$\Delta_{\text{rel}} = \frac{\mu_{\text{after}}}{\mu_{\text{before}}}, \quad (7)$$

$$\Delta_{\text{abs}} = \mu_{\text{after}} - \mu_{\text{before}}. \quad (8)$$

Using both relative and absolute thresholds is essential to avoid false positives. For example, a CPU rising from 1% to 3% is less critical than one rising from 30% to 90%, even though both represent a threefold relative increase. Therefore, both conditions must be satisfied 9:

$$\Delta_{\text{rel}} > \tau_{\text{rel}} \quad \text{and} \quad \Delta_{\text{abs}} > \tau_{\text{abs}}, \quad (9)$$

where τ_{rel} and τ_{abs} are predefined thresholds. Only then is the shift considered actionable and an alert triggered.

- **Trends:** Trends are gradual monotonic increases or decreases. We first estimate the slope β with *linregress* [13], where a positive β indicates an upward trend and a negative β a downward one. To validate monotonicity, we apply the *Spearman rank correlation* [14], with ρ calculated as in Equation 10:

$$\rho = \frac{\text{cov}(R_x, R_y)}{\sigma_{R_x} \sigma_{R_y}}, \quad (10)$$

where R_x and R_y are the rank variables of time and values, respectively. A strong ρ confirms monotonic change, which helps distinguish genuine trends from level shifts.

2) *Profile-based anomaly detection:* In addition to pattern-based detection, we analyze the relationship between workload and resource utilization by modeling service consumption profiles. A regression model is trained on historical input-output pairs, so that for any incoming feature vector $\mathbf{x}$, the expected resource consumption can be estimated as in Equation 11:

$$y = f(\mathbf{x}; \theta) + \epsilon, \quad (11)$$

where $f(\cdot)$ is the regression function parameterized by θ , and ϵ represents noise. In practice, we use both linear regression and decision tree regression [15] to capture linear and non-linear dependencies. For anomaly detection, predicted values $\hat{y}_i$ are compared with actual values y_i on a point-by-point basis, and an anomaly is flagged if both the relative and absolute deviations exceed predefined thresholds, following the same principle of relative and absolute change introduced in the level-shift section. Here, i denotes the index of a data point in the time series, and the deviations are given by 12 and 13:

$$\Delta_{\text{rel},i} = \frac{y_i}{\hat{y}_i}, \quad (12)$$

$$\Delta_{\text{abs},i} = y_i - \hat{y}_i. \quad (13)$$

The scenarios for applying our algorithms are summarized in Table I.

TABLE I: Application scenarios of the proposed algorithms

Algorithm	Scenario
Spikes/Dips	Identifying sudden changes in incoming requests or CPU usage
Level Shifts	Detecting increases in packet loss, memory page fault rates, and other error indicators of CNFCs and servers; capturing latency changes after a software upgrade event
Trends	Identifying potential memory leaks in servers
Profile-Based	Detecting configuration errors or code logic bugs that cause applications to consume excessive resources when handling requests

III. EXPERIMENT

A. Experimental settings

1) *Pattern detection*: To evaluate the pattern detection algorithms, we generated time series data and stored them in *VictoriaMetrics* [16], a high-performance time series database. Each time series contained 100 data points with a fixed sampling interval of 30 seconds. For each anomaly type (spike/dip, trend, and level shift), as well as baseline data with noise, we generated 1000 sample time series.

To construct patterns, we first built a baseline time series from a baseline value b and then added uniform noise to obtain the base data, as in 14:

$$x = b + \epsilon, \quad \epsilon \sim \mathcal{U}(-\alpha b, \alpha b), \quad (14)$$

where α was a scaling factor (e.g., $\alpha = 0.01$ for 1% noise relative to the baseline). This formulation ensured that the noise range scaled with the magnitude of b , reflecting real-world systems where variability typically grew with measurement magnitude. We used uniform noise instead of Gaussian noise because the uniform distribution had a bounded range, while the Gaussian distribution had unbounded tails; this prevented extreme outliers and made the injected anomalies easier to isolate and interpret during evaluation.

Based on this baseline data, we injected specific patterns corresponding to different pattern types (see the parameter summary in Table II):

- **Spikes/Dips**: deviations from the baseline, either *short* (randomly separated indices I_s) or *long* (contiguous segments I_ℓ), as defined in 15:

$$x'_i = \begin{cases} x_i + \delta, & i \in I \text{ (spike)}, \\ x_i - \delta, & i \in I \text{ (dip)}, \\ x_i, & i \notin I, \end{cases} \quad (15)$$

where $I \subset \{t_0, \dots, t_0 + L - 1\}$ denotes the affected indices, and $\delta > 0$ is the deviation magnitude.

- **Trends**: linear deviations from the baseline x_i with slope β , starting at changepoint index j , as in 16:

$$x'_i = \begin{cases} x_i + (i - j) \cdot \beta, & i \geq j, \\ x_i, & i < j, \end{cases} \quad (16)$$

where $\beta > 0$ indicates an increasing trend and $\beta < 0$ a decreasing trend.

- **Level Shifts**: sudden offsets of $\pm\Delta$ after changepoint index j , as given in 17:

$$x'_i = \begin{cases} x_i, & i < j, \\ x_i \pm \Delta, & i \geq j, \end{cases} \quad (17)$$

where $\Delta > 0$ is the magnitude of the shift.

TABLE II: Summary of parameters for pattern generation

Symbol	Description
x_i	Baseline value at index i
x'_i	Modified value at index i after anomaly injection
I	Set of indices affected by spikes/dips (short I_s , long I_ℓ)
δ	Deviation magnitude for spikes/dips
j	Changepoint index (randomly selected during time series generation)
β	Slope of trend ($\beta > 0$: upward, $\beta < 0$: downward)
Δ	Magnitude of level shift (positive offset applied after j)

We constructed a dataset of 4,000 labeled time series, generated from the baseline formulation and pattern injection rules described above. Each algorithm was designed to detect a specific type of pattern: a time series with the corresponding label was treated as a positive instance, while a time series with other labels was treated as a negative instance. In this way, the evaluation of each algorithm was formulated as a binary classification problem. We used two common metrics—the true positive rate (TPR) and the false positive rate (FPR)—computed as in 18:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}; \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad (18)$$

where TP, FP, TN, and FN denote true positives, false positives, true negatives, and false negatives, respectively.

2) *Profile-based anomaly detection*: To evaluate the efficiency of anomaly detection in resource consumption, we collected the TPS (transactions per second) and CPU usage of a service named CHF-Gy under normal operation and trained a regression model to predict CPU usage from TPS. To test the algorithm, we collected TPS and CPU usage over two periods: normal operation and abnormal operation. For the abnormal case, we simulated a misconfiguration in which DEBUG mode was enabled for all service loggers. This configuration forced the application to write an excessive number of log records to disk and was applied only in our development environment, since heavy disk I/O reduces application performance.

To measure how well the predictions $\hat{y}$ matched the actual observations y , we calculated the coefficient of determination, see 19:

$$R^2 = 1 - \frac{\sum_t (y_t - \hat{y}_t)^2}{\sum_t (y_t - \bar{y})^2}, \quad (19)$$

where $\bar{y}$ is the mean of the observed values. Since R^2 may increase with additional predictors regardless of their relevance, we also used the adjusted R^2 , as in 20:

$$R^2_{\text{adj}} = 1 - (1 - R^2) \cdot \frac{n - 1}{n - p - 1}, \quad (20)$$

with n denoting the number of samples and p the number of predictors. Low R^2 or R^2_{adj} values indicate that the predicted and actual resource usage do not match, which we considered an anomaly.

To detect anomalies in the CPU usage of CHF-Gy based on the trained regression model, on the CPU utilization scale (0–100%), we set the relative threshold $\Delta_{rel,i} = 1.15$ and the absolute threshold $\Delta_{abs,i} = 10\%$ in the playbook step.

B. Experiment results

1) *Pattern detection*: Figure 3 summarizes the true positive rate (TPR) and false positive rate (FPR) for each pattern.

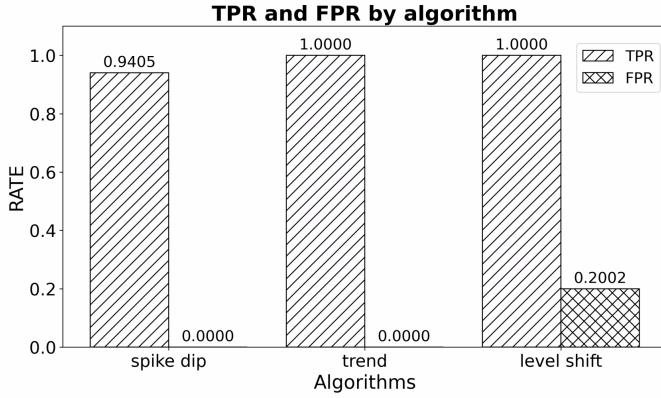


Fig. 3: TPR and FPR by algorithm

The results show that spike/dip detection achieved a high true positive rate (94.05%) with no false positives. Trend detection performed perfectly, with 100% TPR and 0% FPR. In contrast, level-shift detection achieved full sensitivity (100% TPR) but suffered from a high false positive rate (20.02%). This issue arose because the algorithm often confused gradual trends with actual level-shift patterns, leading to misclassification.

2) *Profile-based anomaly detection*: The results of our algorithm using two regression models are shown in Table III. For abnormal data, the resulting R^2 scores were very low, indicating the presence of anomalies. Since the relationship between input requests and CPU usage is mostly linear, the linear regression model provided better performance; consequently, evaluations on the normal test set yielded significantly higher R^2 values. Illustrative outputs are given in Figures 4 and 5.

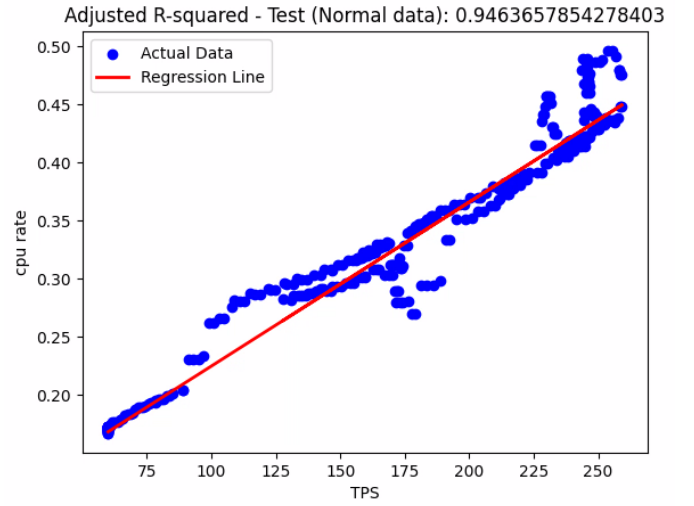


Fig. 4: Results on normal data

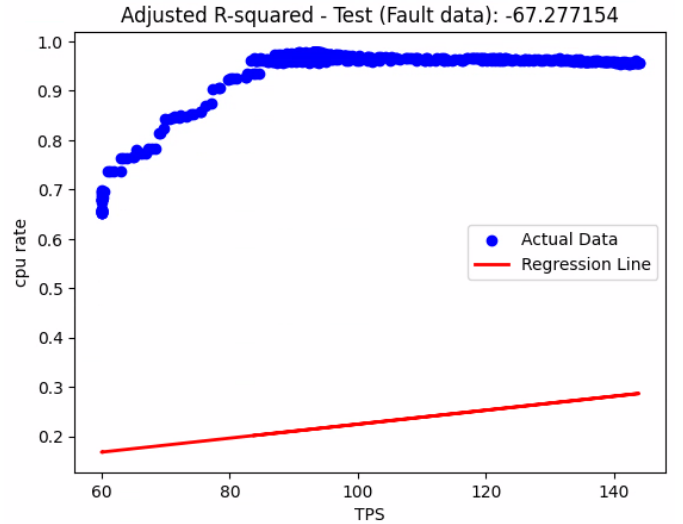


Fig. 5: Results on fault data

TABLE III: Results of two regressors measured by Adjusted R^2

Regressor	Adjusted R^2		
	Train	Test (Normal data)	Test (Fault data)
Linear	0.9391	0.9464	-67.2772
Decision Tree	0.9291	0.7979	-69.1759

3) *SLO violation investigation*: To evaluate the efficiency of our investigation framework, we set up an experiment with three services: GW-Gy, CHF-Gy, and ABMF. Each service contained multiple CNFCs, and each CNFC was deployed as a Kubernetes pod. SLO violations were monitored at GW-Gy, where any faults originating from CHF-Gy, ABMF, or the database were also observable. Once an SLO violation was detected at GW-Gy, our enrichment service added context for the alarm as described in Table IV.

TABLE IV: Example enriched data and data sources

Additional Data	Data Source
Names of VNFCs of GW-Gy	MANO
Servers hosting VNFCs of GW-Gy	MANO
GW-Gy connected to CHF-Gy	Design document
Names of VNFCs of CHF-Gy	MANO
Servers hosting VNFCs of CHF-Gy	MANO
CHF-Gy connected to ABMF	Design document
Names of VNFCs of ABMF	MANO
Servers hosting VNFCs of ABMF	MANO

After enrichment, the enriched data became the input for playbooks to investigate the incident. An example playbook input is shown in Table V.

TABLE V: Example enriched playbook input

Parameter name	Value
Original data from alarms	
cnf_name	voice-charging-01
gw_name	gw-gy
timestamp	2025-07-10T15:28:05+07:00
Additional data after enrichment	
gw_cnf_names	gw-gy-01 gw-gy-02
gw_server_names	server-app-01 server-app-03
chf_name	chf-gy
chf_cnf_names	chf-gy-01 chf-gy-02 chf-gy-03
chf_server_names	server-app-02 server-app-04 server-app-05
abmf_cnf_names	abmf-01 abmf-02 abmf-03 abmf-04
abmf_server_names	server-app-01 server-app-03 server-app-06 server-app-07

Using the parameters from the previous step, our playbook executed a checklist to investigate the incident. Real-world playbooks for this scenario can contain dozens to more than one hundred steps to collect data, but in this experiment we implemented a simplified playbook with the following steps to collect data within 60 minutes before the incident was detected:

- Query the average TPS of GW
- Check for spikes in incoming requests of GW (using the spike detection algorithm)
- Query the CPU usage of CNFCs (pods) of GW, CHF, and ABMF
- Query the request success rate of GW, CHF, and ABMF, defined as the number of requests processed on time divided by the total number of requests
- Detect anomalies in CPU usage of CHF-Gy based on its TPS
- Count the number of database error alarms reported by ABMF

The playbook results showed the following:

- Incoming requests at GW-Gy were **NOT HIGH**
- **No spikes** found in incoming requests of GW-Gy
- CPU usage of GW-Gy and ABMF was **OK**
- CPU usage of CHF-Gy was **HIGH**
- The request success rate of GW-Gy and CHF-Gy was **LOW**
- The request success rate of ABMF was **OK**
- CPU usage relative to requests at CHF-Gy was **HIGHER** than normal

- **No database error alarms** were reported by ABMF

From these results, we conclude that the bottleneck of the system was CHF-Gy (CPU usage was high and the success rate was low). The low success rate at GW-Gy was a symptom originating from the low success rate of CHF-Gy. CPU usage relative to the number of incoming requests at CHF-Gy was also higher than normal, indicating that the problem likely came from a programming logic bug or a misconfiguration.

IV. CONCLUSION

In this paper, we introduced an efficient framework to investigate the root cause of SLO violations in the 5G Online Charging System. Our framework integrates with multiple data sources to enrich alarm context and to collect information about objects related to each incident. Additionally, it has been designed to support pattern-based anomaly detection and profile-based analysis, enabling operators to locate problems more quickly and accurately.

In the future, we envision applying large language models (LLMs) to automatically infer the root cause of incidents from the diagnostic results provided by our playbooks. To enable this capability, we plan to build a retrieval-augmented generation (RAG) system [17] that incorporates knowledge of our system architecture as additional context for inference. Another direction is fine-tuning commercial off-the-shelf (COTS) models with domain-specific data to improve reasoning in telecom environments. Furthermore, the root cause inferred by the LLM could be used as input to an automation platform that resolves the problem directly or isolates the faulty component from the system.

ACKNOWLEDGMENT

REFERENCES

- [1] Ericsson, "What is 5g? how will it transform our world?" <https://www.ericsson.com/en/5g>, n.d., accessed: 2025-09-25.
- [2] ETSI, "Policy and charging control framework for the 5g system," European Telecommunications Standards Institute, Tech. Rep. ETSI TS 123 503 V18.9.0, 2025.
- [3] ETSI, "Network Functions Virtualisation (NFV); Management and Orchestration," European Telecommunications Standards Institute (ETSI), Tech. Rep. GS NFV-MAN 001 V1.1.1, 2014, accessed: 2025-09-29. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf
- [4] —, "3GPP TS 123.203: "Policy Control and charging architecture", European Telecommunications Standards Institute (ETSI), Tech. Rep. TS 123 203 V17.2.0, 2022, accessed: 2025-02-03. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/123200_123299/123203/17.02.00_60/ts_123203v170200p.pdf
- [5] 3rd Generation Partnership Project (3GPP), "5G System; Service Communication Proxy Services; Stage 3," 3GPP, Technical Specification TS 29.570, 2025, release 19. [Online]. Available: <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=4389>
- [6] J. P. Betsy Beyer, Chris Jones and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems (1st ed.)*. O'Reilly Media, Sebastopol, CA., 2016.
- [7] P. Authors, "Prometheus query language (promql) documentation," <https://prometheus.io/docs/prometheus/latest/querying/basics/>, accessed: 2025-09-25.
- [8] K. Authors, "Kubernetes documentation," <https://kubernetes.io/>, 2025, cloud Native Computing Foundation.

- [9] V. Fajardo, J. Loughney, G. Zorn, J. Arkko, and J. Arkko, "Diameter Base Protocol," RFC 6733, Oct. 2012. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc6733>
- [10] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median," *Journal of experimental social psychology*, vol. 49, no. 4, pp. 764–766, 2013.
- [11] R. Killick, P. Fearnhead, and I. A. Eckley, "Optimal detection of changepoints with a linear computational cost," *Journal of the American Statistical Association*, vol. 107, no. 500, pp. 1590–1598, 2012.
- [12] R. F. Woolson, "Wilcoxon signed-rank test," *Wiley encyclopedia of clinical trials*, pp. 1–3, 2007.
- [13] P. e. a. Virtanen and SciPy 1.0 Contributors, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, pp. 261–272, 2020.
- [14] J. H. Zar, "Spearman rank correlation," *Encyclopedia of biostatistics*, vol. 7, 2005.
- [15] M. Xu, P. Watanachaturaporn, P. K. Varshney, and M. K. Arora, "Decision tree regression for soft classification of remote sensing data," *Remote Sensing of Environment*, vol. 97, no. 3, pp. 322–336, 2005.
- [16] VictoriaMetrics, "Victoriametrics: Simple reliable monitoring for everyone," <https://victoriametrics.com/>, accessed: 2025-10-03.
- [17] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.



Che Bui Duc received his Engineer's degree in Computer Engineering from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2023. He is currently a Software Engineer at Viettel High Tech, focusing on backend development and system modeling.



Duc Nguyen Minh received his Engineer's degree in Computer Engineering from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2023. He is currently a Software Engineer at Viettel High Tech, focusing on backend development and system modeling.



Thuyet Tran Van is currently the Deputy Director of the Payment Platform Development Center, Viettel High Technology Industries Corporation, Viettel Group. He received his Engineer's degree in Information Systems from Hanoi University of Science and Technology, Hanoi, Vietnam, in 2013.



Kien Hoang Dang is a Software Engineer at Viettel High Technology Industries Corporation, Viettel Group. He received his Bachelor's degree in 2019 and Master's degree in 2023 from the University of Engineering and Technology, Vietnam National University, Hanoi. His current research interests encompass root cause analysis, anomaly detection, and the design of autonomous networks.



Hieu Cong Nghia received his Bachelor's degree in Computer Science from University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam, in 2025. He is currently a Software Engineer at Viettel High Tech, focusing on backend development and system modeling.



Thai Nguyen Trong is currently the Deputy Head of the Information Technology Department, Global Technical Center, Viettel Networks.



Minh Nguyen Ngoc received his Bachelor's degree in Computer Science from the University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam, in 2025. He is currently a Software Engineer at Viettel High Tech, focusing on data engineering and AI.