Log Analysis and Prediction for Anomaly Detection in Network Switches

Sukhyun Nam, Euidong Jeong, Jibum Hong, Jae-Hyoung Yoo, and James Won-Ki Hong

Department of Computer Science and Engineering, POSTECH, Pohang, Korea
{obiwan96, justicedong, hosewq, jhyoo78, jwkhong}@postech.ac.kr

***Abstract*—In this study, we propose a three-step anomaly detection system for network switches. The proposed system consists of the following steps: 1) Log parsing, where log messages from switches are analyzed to identify patterns and events, 2) Analysis of the identified event flow to distinguish normal and abnormal event sequences, and 3) Prediction of the next log message, with detection of anomalies if the predicted log message differs from the normal log messages. For event classification, a log parser is proposed by modifying existing algorithms, and experimental results confirm that similar log patterns are correctly classified into the same event. To learn normal event sequences, both FSM and LSTM models are trained. Lastly, we proposed a BERT-LSTM model to predict the next log message and detect unexpected log messages. The proposed system is validated using data collected from a constructed testbed and achieves a high-performance level with an F1 score of 83.72%. Notably, our system achieved a recall of 94.74%. Our system has an advantage in that if misclassified cases occur, network administrators can retrain each model to improve precision during system operation.**

***Index Terms*—Anomaly Detection, Machine Learning, Log Analysis**

I. Introduction

Recently, as networks grow in size and complexity, various types of network management problems (e.g. faults, failures, packet loss, and delay increase) are increasing. In order to solve or alleviate these problems, network managers analyze the resource usage, status information, network traffic information, and logs of the equipment and related systems that make up the networks [1].

Anomaly detection refers to the identification of data or patterns that do not conform to the expected behavior or normal operation of a device. Such inappropriate data or patterns are commonly referred to as anomalies or outliers. Anomaly detection techniques are widely used in various fields, including network management, cybersecurity, medical diagnostics, and more. By developing anomaly detection technology for network devices, it is possible to detect and notify any device or network-wide issues before or immediately upon their occurrence during network operation. This enables prompt actions to be taken without the need for network administrators to manually inspect vast amounts of data.

Among various data, logs generated by devices represent the device's state most intuitively. Unlike simple numerical data, log messages provide detailed information about the device's state, which is why network administrators commonly rely on it the most. However, the log messages are not generated in a standardized format. Additionally, logs are natural language, unlike numerical data such as resource usage, making automated analysis challenging. As a result, conventional network management systems often rely on statistical analysis of log data. Such techniques detect anomalies based on rule-based extraction of specific "patterns" or words or by analyzing the frequency of extraction of particular words or log messages.

In recent years, there has been a remarkable advancement in the field of Natural Language Processing (NLP) due to the development of Artificial Intelligence (AI). With machine learning at its core, it has become possible to analyze the meaning of sentences. Similarly, in log processing, machine learning can be employed to analyze the meaning present in log messages, enabling automatic detection of log messages that indicate abnormal states different from normal ones. In fact, research has been conducted to enhance network management performance by applying machine learning techniques [2]–[4]. However, the machine learning approach requires significant computational resources for training and detection. Therefore, combining pattern-based methods with machine learning-based approaches can increase the efficiency of anomaly detection. In this study, we utilized a pattern-based approach and machine learning-based approach, especially Bidirectional Encoder Represents from Transformers (BERT) [5], a language model that has recently shown better performance than existing technologies in the NLP area.

This paper proposes a three-step anomaly detection system using machine learning. We have developed an log parser and built a model to determine whether the event number flow is normal based on the output of the log parser. Additionally, we developed the machine learning model to train on log data representing the normal state and predict the next log message. The contributions of this paper are as follows:

- Developed log parser to analyze log patterns and classify events
- Developed machine learning and Finite State Machine (FSM) based model to detect abnormal events flow
- Developed a machine learning model based on BERT to predict the next log message
- Designed anomaly detection system based on these models and evaluated the performance of the system with the log dataset collected in the testbed

978-3-903176-59-1

II. Related Work

A. Log-based Anomaly Detection

Research on anomaly detection based on log analysis has long been conducted using pattern extraction techniques. Pattern extraction techniques are commonly used for log analysis because they are simple, intuitive, and show high performance [6]. These techniques typically involve extracting events from the extracted patterns. Among the generated log patterns, similar patterns are highly likely to represent the same event. Therefore, many studies classify log patterns into events based on the similarity between log patterns. The event classifier is capable of substituting each log with an event. Anomaly detection studies analyze the substituted event flows and detect anomalies when the observed event flow deviates from normal behavior.

P. He *et al.* [7] proposed a tree-based log pattern analysis technique. They trained a log pattern tree by initially classifying log messages based on the number of words and then reclassifying them based on whether the first word matches. The classified log messages were then measured for similarity with existing log groups, and if a group with a similarity above a certain threshold was found, the log message was included in that group. Otherwise, a new group was created. This approach has the advantage of analyzing and generating patterns in real time. However, it has limitations since it relies on the assumption that log messages representing the same event have the same number of words.

Pattern analysis-based log analysis techniques have the advantage of not requiring complex training processes and demonstrating powerful performance with simple and intuitive algorithms. However, as the volume of log messages increases and the number of events becomes more diverse, the analysis becomes complex, leading to potential performance limitations in large systems with a significant amount of log messages.

Y. Liang *et al.* [8] proposed a technique to predict critical events based on logs by using machine learning. They proposed a model for predicting whether critical events will occur in the next interval based on a fixed observation period of event logs collected from IBM supercomputers. The authors did not apply complex techniques in log analysis; instead, they simply transformed the log data into statistical data based on rule-based approaches (e.g., counting the number of warning events). The transformed data is then used as input for a nearest neighbor search model, which predicts the occurrence of critical log events for the next 12 hours with a performance of F1 score of 0.7.

We also conducted research on failure prediction for virtual machines (VMs) operating Virtualized Network Functions (VNFs) based on log data [4]. To detect failure-related log messages before the actual occurrence, we utilized both BERT and Convolutional Neural Network (CNN) models. BERT converts each word in the input log data into a numerical vector, while CNN plays a role in capturing features related to failures from a large volume of input data. When testing our models with the collected data, we achieved an F1 score of 0.74. While the performance of this study may be somewhat lower for practical application in real systems, it demonstrated the potential of log analysis using BERT-based techniques.

Machine learning-based algorithms require significantly higher computational resources compared to pattern-based algorithms. However, they have the capability to detect abnormal states that are difficult for pattern-based algorithms to identify.

B. BERT (Bidirectional Encoder Represents from Transformers) [5]

To utilize log data in machine learning, it needs to be transformed from natural language into numerical data. In NLP, numerical vectors representing the meaning of the text are referred to as ”embeddings”, and the models that generate these embeddings are called language models. BERT is a powerful language model developed by Google. It is a state-of-the-art technique in NLP that has revolutionized various NLP tasks. BERT is designed to capture the contextual relationships between words in a sentence.

BERT consists of two main components: the BERT tokenizer and the BERT model. The tokenizer is responsible for segmenting each word in the input sentence into sub-word tokens. The BERT model takes the token IDs as input and generates embedding vectors for each token.

The training process of BERT involves pre-training and fine-tuning. In the pre-training phase, BERT is trained on large amounts of unlabeled text from the internet to learn a general language representation. In the fine-tuning phase, BERT is further trained on specific downstream tasks. With this step, BERT, despite being a pre-trained model publicly available, can be fine-tuned to suit our specific task. We trained the BERT model with other machine learning models via fine-tuning, which allowed us to extract word embeddings suitable for log analysis.

III. Design and Implementation

This section describes the design of the proposed anomaly detection system. The proposed system consists of three steps: 1) a log parser that classifies log messages into event numbers, 2) an event number-based anomaly detection model, and 3) a log predicting-based anomaly detection model. The second step uses the results of the first step.

A. Log Parser

In the first step, the log parser analyzes patterns of log data to extract events from each log message and convert them into event numbers. The second step model includes a model that has learned the event number sequence of the normal

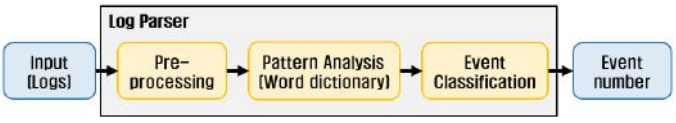

Fig. 1. Log parser structure

state, allowing it to detect anomalies when the input event number sequence deviates from normal. To train the model on the normal states event sequence, we utilized FSM and LSTM (Long Short-Term Memory). Lastly, in the third step, the BERT-based ML model is utilized to receive the previous log messages as input and predict the next log message. If the predicted log message differs from the actual log message, it triggers an anomaly.

The following provides detailed explanations of each step. Each model, including the log parser, needs to undergo a training phase using a large number of normal log data before it can be utilized for anomaly detection. The following explanations assume that the models have undergone pre-training. The results of pre-training will be described in the next chapter.

Figure 1 illustrates the structure of the log parser we used. The log parser involves identifying identical patterns in the log messages through log pattern analysis and utilizing an improved algorithm based on the Minimum Edit Distance algorithm to classify the generated patterns as events.

Initially, log messages undergo a pre-processing stage where rule-based techniques are applied to remove numbers, dates, file paths, location names, and interface names and add corresponding tokens. For example, the log "UI_NETCONF_CMD: User 'root' used NETCONF client to run command 'close-session'." is changed to "ui netconf cmd user root used netconf client to run command close session." and the log " Line protocol on Interface Giga0/49, changed state to down" is changed to "line protocol on interface [interface] changed state to down" in the pre-processing step.

Pre-processed log messages go through pattern analysis based on word dictionaries. The word dictionaries can be created based on the collected log data, where words that appear three or more times in the pre-processed log data are included in the dictionaries. Unconverted words in the pre-processed log data are transformed into the '[UNK]' token, representing unknown words not present in the word dictionaries. This process removes irrelevant content such as process IDs. Through this process, the log messages are transformed into log patterns, enabling the grouping of different log messages into the same format.

The generated log patterns undergo event classification to transform them into event numbers. In the event classification process, a pre-generated event dictionary is utilized. The event dictionary is created during the pre-training of the log parser phase, where log patterns generated from the pre-training process are classified into events based on the similarity between sentences.

```
event_list=[]
for single_pattern in log_patterns
   for single_event in event_list
      edit_distance_sum = 0
      for log_pattern in single_event
         edit_distance_sum +=
             edit_distance(log_pattern,
         single_pattern)
      if edit_distance_sum /
          len(single_event) /
          len(single_pattern) < 0.5
         put single_pattern to single_event
      end for
return event_list
```

Table 1. pseudo code of pattern classification algorithm

Table 1 is a pseudo code of the minimum edit distance algorithm-based pattern classification algorithm. We improved the minimum edit distance algorithm to use as a similarity measurement. The minimum edit distance algorithm calculates the minimum number of edits (deletions, insertions, and substitutions) required for two sentences to match. When a new log pattern is inputted, it is compared to all previously classified log messages within the same event. If when comparing the newly entered log pattern to all existing patterns within the event, the average edit distance is less than half of the sentence length, the new log pattern is included in that event.

Additionally, to classify log patterns that are similar but have opposite meanings into different events (e.g. 'interface [interface] down' and 'interface [interface] up'), we modified the minimum edit distance algorithm. In the general minimum edit distance algorithm, the edit distance increases by 1 when a word is transformed. However, by incorporating a publicly available antonym dictionary from WordNet [9], when two words to be modified are in an antonym relationship, the edit distance is increased by 6. This approach allows for the classification of log patterns that are similar but have opposite meanings into different events.

Through the minimum edit distance-based pattern classification algorithm, a large number of log patterns can be classified into events. In the anomaly detection process, during the event classification stage of the log parser, if the input log pattern is found to be part of a classified event, the corresponding event number is outputted. However, if the input log pattern does not match any existing patterns, it can be classified as an anomaly, indicating the detection of a previously unseen log pattern. When an anomaly is detected, the network administrator can review it. If it is determined to be a normal log message that was not previously identified, the pattern classification algorithm is used to check if it can be included in any existing events. If it cannot be included in any existing events, a new event is created, and the pattern is added to it.

B. Event Number-based model

Once the Log Parser converts each log message in the log data into an event number, the log data is transformed into an event number sequence. We hypothesized that if the model learns the event number sequence representing the normal state, it can detect event number sequences corresponding to abnormal states. To train the model on the event number sequence representing the normal state, we experimented with two models: FSM and LSTM.

FSM is an automaton with a finite number of states, and it can have only one state at a time. Each state can transition to another state based on specific events, and an FSM is

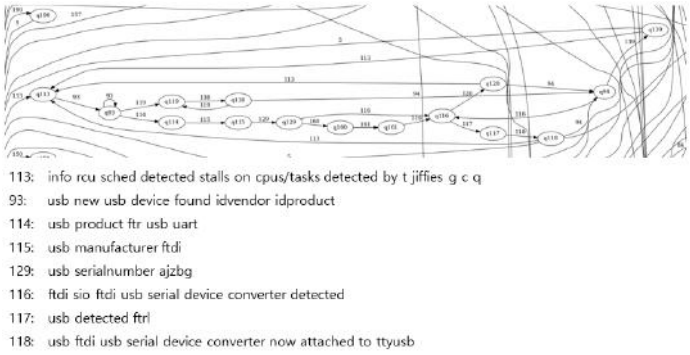

Fig. 2. Normal event flow example in FSM

composed of states, transition states, and the set of conditions that trigger the transition states.

To learn the normal event flow FSM, each event number is defined as a state. Corresponding states are defined for each event number. For example, Figure 2 represents a partial FSM learned in this study and Event 113 corresponds to state q113. The FSM receives a log event flow in a steady state and learns each event number flow as a transition. We can find the event flow sequence of $113 \rightarrow 93 \rightarrow 114 \rightarrow 115$ indicates a normal flow. By examining the log messages corresponding to these event numbers in the event dictionary, we can analyze that the log flow shown in Figure 2 is a normal log.

We decided to judge the newly entered event flow as normal if it is an event flow that can move within two times in the FSM, considering that logs are generated by multiple systems even in the same normal state. That is if the input event flow cannot move within two times in the FSM, the FSM model detects an abnormal state. Similar to log parser-based anomaly detection, if the network administrator responds to an anomaly alert as normal, the FSM can be expanded by adding the corresponding transition, thereby expanding the database for normal states. Since there are 209 states, the complete FSM diagram is too extensive to be included in this paper, but it can be found in the README of our repository [10].

Figure 3 illustrates the structure of the machine learning model to predict the next event number which consists of a simple model combining an LSTM layer and a fully connected layer. LSTM is a well-known machine learning algorithm that is particularly suitable for analyzing sequential data. It performs well in tasks that involve predicting the next sequence when data is inputted sequentially. The proposed machine learning model in this study takes the last n event numbers as input. In the output layer of the LSTM, a fully connected layer is added, connecting all neurons to perform event number classification. The output of this layer generates a k-dimensional vector, where k represents the total number of events. For each event number, the model calculates the probability of the next log message belonging to that event. To ensure that the probabilities sum up to 1, the output passes through a softmax layer. Similar to FSM, log data does not consistently generate the same log messages even in a normal state, so instead of considering only the highest probability event number predicted by the LSTM model, we conducted anomaly detection based on a number of top predicted event numbers. Anomaly detection occurs when consecutive incoming event numbers differ from the predicted event numbers.

We utilized both FSM and LSTM models for event-based anomaly detection.

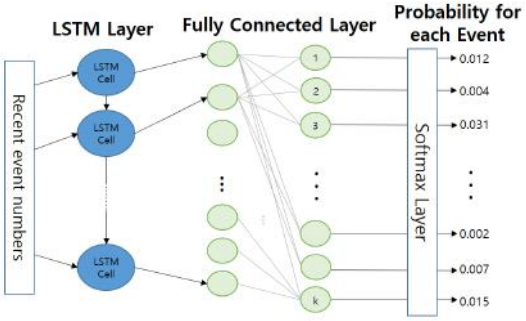

Fig. 3. LSTM model structure

C. BERT-LSTM based model

In addition to pattern analysis, we also conducted anomaly detection using a non-pattern-based approach. BERT generates a sentence vector by grasping the context of each word within a sentence in both directions, and LSTM learns the flow of the generated sentence vector. Since the transformed sentence vectors represent sequential data, similar to event flows, we trained an LSTM model using these vectors.

Figure 4 illustrates the flowchart of the BERT-LSTM-based process for predicting the next log message and performing anomaly detection. In this process, lines of consecutive log sentences are used as the input to the model. Based on this input, the model predicts the next log message and determines

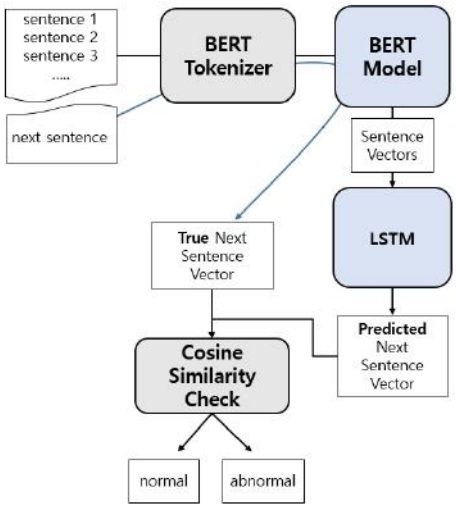

Fig. 4. BERT-LSTM log prediction model structure

whether it is normal or anomalous by comparing its similarity to the actual next log message.

The BERT model converts each word to embeds, but also generates sentence vectors. We have trained the machine learning model to predict the next sentence vector by taking this sentence vector as input, rather than predicting the next sentence by taking the sentence as input. The LSTM calculates the next sentence vector, and we can obtain the actual next sentence vector by inputting the actual next sentence into the BERT model. We measure the cosine similarity between the two sentence vectors for similarity comparison. If the calculated cosine similarity is below a predefined threshold, it indicates that the predicted sentence deviates significantly from the actual log message, suggesting an abnormal state.

The parts highlighted in blue in Figure 4 represent the segments where back-propagation occurs, indicating the parameter modification during the training process. Although the BERT model is pre-trained and publicly available, back-propagation allows for parameter adjustments to generate vectors suitable for log analysis.

IV. Experiments and Evaluation

A. Data Collection and Log Parser

To validate the proposed technique, we collected log data and conducted performance evaluation experiments. Figure 5 represents the testbed used for log collection in this study. The testbed consisted of clients that requested HTTP traffic to a web server, and the L2 switch from Ubiquitous and the L3 switch from Juniper was configured to collect logs from both switches. We considered the collection of anomaly state data, so we connected Client 1 directly to the L3 switch to further overload the L3 switch. And we collected logs from all switches with rsyslog [11].

We used only normal logs to train our models. Later, we collected abnormal logs as test data. As a result, a total of 741,387 lines of normal logs were collected. Based on this data, we created a word dictionary containing 690 words. Using the word dictionary, we generated 658 log patterns. The following are the top log patterns along with their occurrence frequencies.

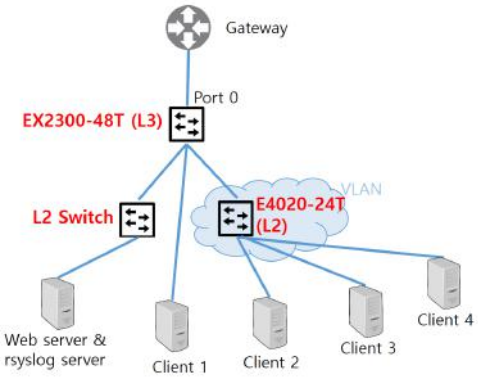

Fig. 5. Log collection testbed

- adt set pwm ic id pwm [number] : 260,268
- fan set speed slot fanid speed pwm [number] : 260,268
- br leaf received packet on em with own address as source address addr e d vlan : 24,200
- authentication from for admin success : 13,779
- started session of user root : 13,271

Among the log patterns, the 'adt ∼' log pattern and 'fan set ∼' log pattern occurred most frequently and accounted for more than half of the entire log dataset. However, these logs, which indicate the fan running too fast, were excluded from the training and experiments because they resulted in overly simplistic event flows and excessively high performance. (Since more than half are two types of log messages, performance increases rapidly when the corresponding predictive model predicts as that log messages.)

As a result, out of the 656 log patterns, they were classified into 209 events during the event classification process. Table 2 is part of the classified patterns. As intended, it can be seen that logs representing similar content are classified into the same event. The log parser performs the function of converting each input log message into an event number ranging from 1 to 209.

```
[11]
pvidb attribute ifinfo mru support not
    present in db
pvidb attribute ifinfo interface [interface]
    not present in db
pvidb attribute ifinfo ifinouterrors stats
    not present in db
-----------------------------------------
[12]
interface [interface] changed state to up
line protocol on interface [interface]
    changed state to up
line protocol on interface [interface]
    channel changed state to up
```

Table 2. sample pattern and logs

B. Log Event Prediction

We solely conducted individual performance evaluations for the LSTM model since FSM is not designed for predicting the next log event. However, we utilized the FSM in anomaly detection performance evaluation.

Based on the generated log parser, the existing data of 741,387 lines of log messages can be transformed into event flow data. Based on this data, we created input-output pairs for machine learning, where the input consists of the recent n event numbers and the output is the next event number. We split the generated event flow data into training and test data sets in an 8:2 ratio to train the machine learning models and measure their performance based on how well they predict the next event in a normal state.

We measured the performance and training time according to the number of event numbers used as input. For accuracy measurement, we considered the prediction to be correct if the actual event number was among the top 5 event numbers with the highest predicted probabilities. In the anomaly detection

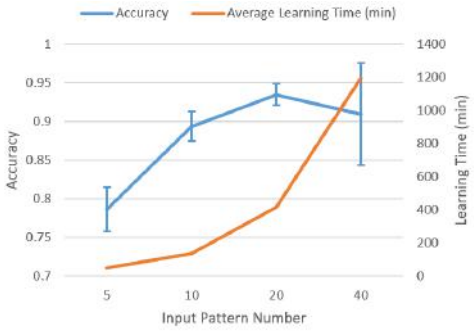

Fig. 6. Event number prediction results

Fig. 7. Log prediction results

phase, we also compare the real event number to the top 5 event numbers.

Figure 6 represents the experimental results. We conducted performance comparison experiments by varying the number of input events in LSTM. We performed 6 experiments for each input size, and for accuracy, we indicated the error bars based on the maximum and minimum values, shown in blue, and the average training time is displayed in orange. The experimental results show that as the number of inputs increases, the overall performance generally improves, but the training time also increases rapidly. For example, when using 40 inputs, the average training time is 1,197 minutes, which is approximately 20 hours. Additionally, in the case of using 40 inputs, there are instances where the accuracy is less than the result from 20 inputs. This can be attributed to overfitting, where using too many inputs leads to a decrease in accuracy. Therefore, we used 20 as the input pattern number in our anomaly detection system.

C. Log Prediction

In measuring the performance of the model, how similar the predicted sentence vector and the actual sentence vector were measured as cosine similarity, and when the cosine similarity was 0.9995 or higher, we judged as correctly predicted. Since cosine similarity was used as the criterion, there was a problem that the BERT model tended to generate similar vectors for all log messages when trained repeatedly, requiring a higher threshold.

Figure 7 represents the experimental results. Despite setting a high threshold, it can be observed that the model achieves 100% accuracy on test data from around 60 epochs onwards. However, beyond this threshold, it can be concluded that the model is not making accurate predictions, but rather the BERT model is generating similar vectors for all sentences. Recognizing this issue, we decided to use the model around 45 epochs, as it was deemed to have already undergone sufficient training based on the analysis of the loss graph. This model achieves an accuracy of 97.5%.

D. Anomaly Detection System

Based on the models we have built, we constructed a 3-step anomaly detection system. Figure 8 illustrates the anomaly detection process in this system. 3 Steps proceed concurrently, and if an anomaly is detected at any stage, it can be terminated early. This allows the three models to be complementary and to create a system with high efficiency and performance. In addition, each model can also easily learn new forms of log messages, which means our models are not vendor-specific. In the first step, when a log message is input, the log parser extracts patterns from the log message. In this step, if an input log message does not exist in the pattern dictionary, it can be detected as an anomaly. If the log message exists in the pattern dictionary, the log parser can convert the log message into an event number.

In the second step, the converted event number is then passed to the event-based anomaly detection model. This model consists FSM-based model and an LSTM-based model. If either of the models determines that the input event number sequence is not normal, it is flagged as an anomaly. In the FSM-based model, if the following event number is unreachable from the previous event number in FSM, it is considered an anomaly. In the LSTM model, if the predicted event numbers are consistently incorrect in sequence, it is detected as an anomaly. The reason for requiring ”consecutive” incorrect predictions is that although our developed models demonstrate a high average accuracy of 93.5%, this still indicates that out of 100 log messages, approximately 6.5 log messages are predicted inaccurately. Since the actual number of log messages generated in a day exceeds 10,000 lines,

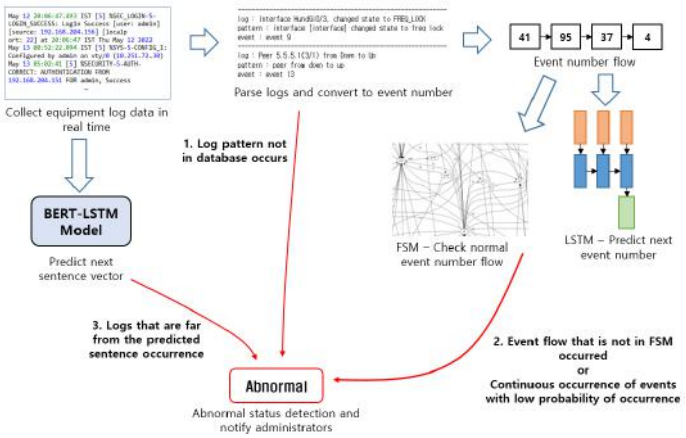

Fig. 8. 3-step Anomaly detection system structure

there is a possibility of more than 650 false anomaly alarms occurring in a day even in the normal state. Therefore, we set the criterion for detecting an anomaly as the LSTM model detecting consecutive anomalies at least 7 times, based on multiple experimental results.

Finally, in the third step, the BERT-LSTM model predicts the next log message. The BERT-LSTM model predicts the next sentence vector, and if the cosine similarity between the predicted vector and the actual input is low, an anomaly alarm will occur.

In all three steps, if the network administrator confirms that a log message identified as abnormal by the anomaly detection system is actually a normal log message, it can be incorporated into the model. Through this process, our system has the advantage of being able to increase precision in real-time.

To validate the performance of our proposed anomaly detection system, we conducted experiments using a dataset that includes both normal and abnormal data. The normal logs used in the experiments were collected from our testbed (Fig. 5), and each log data represents a day's worth of log messages. To collect abnormal logs, we overloaded the network switches with DoS [12] and Portscan attacks generated from clients to web servers in the testbed. After maintaining the overload for approximately one week, we collected switch logs in the overloaded state from each network switch for two weeks. Additionally, real abnormal log data from the Samsung testbed, which occurred when switches terminated abnormally during production, was also included. A total of 18 normal log datasets and 19 abnormal log datasets were used in the experiments. In the experiments, when a day's worth of log data was inputted into the anomaly detection system and any step detected an anomaly in any part of the data, we considered our system to have detected an anomaly in the corresponding data.

The experimental results showed an accuracy of 81.08%, an **F1 score of 83.72%**, a precision of 75%, and a recall of 94.74%. As tagging abnormal states as normal is more critical than tagging normal states as abnormal in the anomaly detection system, recall is more important than precision and our system shows a high recall.

V. Conclusion and Future Work

In this work, we investigated a technique for detecting abnormal states in network switch equipment using log pattern analysis and log prediction. Experimental results using collected switch log data demonstrated that the proposed anomaly detection system achieved an F1 score of 83.72%. The system exhibited a high recall of 94.74%, and any misclassified cases can be promptly addressed through additional model training, so it has potential for practical application. In addition, our model learns and uses logs in a normal state, so it can be used through learning on switches other than those we used.

However, the machine learning models proposed in this study require significant training time, necessitating high-performance equipment for practical deployment. Thus, future research is planned to explore more efficient algorithms to replace the existing models. Additionally, we identified the issue of BERT generating similar vectors for all log messages during the experiments, so algorithmic improvements to mitigate this problem are necessitated.

All the code used in this study is publicly available on GitHub [10], providing insights into the implementation process of these models. Additionally, although not included in this paper, the diagrams of FSM constructed using the collected data can be found in the README file of the corresponding GitHub repository.

Acknowledgment

This work was supported by Korea Evaluation Institute Of Industrial Technology (KEIT) grant funded by the Korea Government (MOTIE) [(No.2009633) Development of AI network traffic controlling system based on SDN for ultra-low latency network service], Samsung Electronics Co., Ltd and Smart HealthCare Program funded by the Korean National Police Agency(KNPA, Korea) [Project Name: Development of an Intelligent Big Data Integrated Platform for Police Officers' Personalized Healthcare / Project Number: 220222M01].

References

[1] Raouf Boutaba, Mohammad A Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):1–99, 2018.

[2] Doyoung Lee, Jae-Hyoung Yoo, and James Won-Ki Hong. Deep q-networks based auto-scaling for service function chaining. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2020.

[3] Seyeon Jeong, Nguyen Van Tu, Jae-Hyoung Yoo, and James Won-Ki Hong. Proactive live migration for virtual network functions using machine learning. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 335–339. IEEE, 2021.

[4] Sukhyun Nam, Jae-Hyoung Yoo, and James Won-Ki Hong. Vm failure prediction with log analysis using bert-cnn model. In *2022 18th International Conference on Network and Service Management (CNSM)*, pages 331–337. IEEE, 2022.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, pages 207–218. IEEE, 2016.

[7] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE, 2017.

[8] Yinglung Liang, Yanyong Zhang, Hui Xiong, and Ramendra Sahoo. Failure prediction in ibm bluegene/l event logs. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 583–588. IEEE, 2007.

[9] Princeton University. WordNet. https://wordnet.princeton.edu/, 2005. [Online; accessed 9-July-2023].

[10] Sukhyun Nam. LogEventParsing. https://github.com/obiwan96/LogEventParsing, 2023. [Online; accessed 9-July-2023].

[11] Adiscon GmbH. The rocket-fast Syslog Server. https://www.rsyslog.com/. [Online; accessed 09-July-2023].

[12] Salvatore Sanfilippo. hping3. https://linux.die.net/man/8/hping3, 2006. [Online; accessed 09-July-2023].