LogDTL: Network Log Template Generation with Deep Transfer Learning

Thieu Nguyen
Hanoi University of Science and Technology
nguyenthieu2102gmail.com

Satoru Kobayashi
NII
sat@nii.ac.jp

Kensuke Fukuda
NII/Sokendai
kensuke@nii.ac.jp

***Abstract*—Analyzing network logs is increasingly playing an essential role in system management and maintenance. As a result, more and more new techniques and models have been proposed for automatic log analysis. Log template generation is the essential first step to apply such sophisticated techniques. This article presents an automatic log template generation framework (LogDTL) in which transfer learning technique is used in the deep neural network (DTNN model) to overcome the trade-off between the accuracy of the generated template and human resources for manual labeling. Evaluation results show that DTNN significantly outperforms a well-known supervised method (CRF). DTNN achieves 91% of word accuracy with only one training example though the CRF achieves 78% of word accuracy.**

***Index Terms*—System Log Template Generation, Transfer Learning, Deep Neural Network, Conditional Random Field.**

I. Introduction

Network log (i.e., Syslog) contains records of events showing the detailed system runtime information such as system errors, warnings, system changes, and abnormal shutdowns. By reviewing logs, network operators can identify the cause of a problem that happened in their network. However, modern networks have become so massive and complex to investigate such logs manually. For instance, a research and education network in Japan (SINET5 [1]) reports around one hundred of thousands of log messages in a day. Thus, there is a need for an effective method for analyzing such large-scale log data.

Log message, in general, is a line of text printed by logging statements (e.g., `printf()`, `logging.info()`) written by developers. Mining the information of logs enables us to conduct a wide variety of system management and diagnostic tasks, such as anomaly detection [2]–[4] and root cause analysis [5].

Most log analysis techniques applying data mining models to get insights into system behaviors are built on log templates. A log template is an organized format of an unstructured statement in the log message. It includes a variable (e.g., IP address, interface name) and a fixed component called description. A group of log messages are clustered into a log template representing the common behavior of a system. Hence, generating log templates is an effective way to classify log messages with their behaviors. The log template reduces the burden of the operator's tasks by reading structured logs and controlling system behaviors. Furthermore, log templates are essential to build time series for such log data to apply the sophisticated analysis methods.

A simple approach for generating log templates is to apply regular expression-based matching rules to logs. However, making appropriate rules is a time-consuming task [6]. Instead, there are several approaches to (semi) automatically extract log templates from raw logs. In particular, machine learning (ML) and artificial intelligence model is applied for this task [7], [8]. There are two ML approaches for log template generation: supervised learning and unsupervised learning. The supervised learning methods usually show better accuracy than unsupervised learning methods in a specific system, but it requires large labeled data for the training process. Labeled data could be obtained from the source code of the system or manually generated by human power. However, source code is often inaccessible in practice. Besides, for large systems that emit massive data, it is very laborious to create labeled data manually.

This paper proposes an alternative method for generating log templates from log messages to overcome the trade-off between log template accuracy and human resources (i.e., manual labeling). The key idea is to leverage transfer learning that applies log templates available from open-source software to unknown vendor's logs to obtain its log template. Thus, our approach enables us to avoid the burden of the manual labeling process with high accuracy. We design and implement a full framework of log template generation (called LogDTL) using a proposed deep transfer neural network (DTNN) model. We evaluate our method with (known) log templates from open-source router software and raw log data generated from proprietary routers. Our evaluation demonstrates that our approach outperforms a state-of-the-art supervised model CRF (Conditional Random Field) in four metrics. More specifically, our DTNN model achieves more than 90% accuracy in word-level, while the CRF-based algorithm shows only 78% accuracy. The contribution of our work is as follows:

- We propose the full framework of log template generation (LogDTL) in which the concept of transfer learning is used with the deep neural network (DTNN model)(§ III).
- Our intensive evaluation with real log data shows that DTNN outperforms the supervised learning model (CRF) in accuracy and efficiency (§ V).

II. Related Works

Log template generation (log parsing) has been intensively studied in past literature and classified into two approaches.

Traditional approach: Log templates are generated with custom rule-based parsers [9], [10] or source code/binary analysis [11], [12]. This approach generally generates a wide variety of accurate log templates. However, building the parsers requires huge human resources, and also source code is not always available (i.e., proprietary software).

Data mining approach: The other approach is to rely on recent advances in data mining (or machine learning) techniques. This approach generates log templates from raw logs produced by systems. Thus, more raw log data is available, more chance to generate accurate templates. Besides, it is tolerant of log format changes. Some logs may change their format with system updates, and traditional approaches are largely affected by the changes. Generally speaking, this data mining approach can be categorized into unsupervised or supervised techniques.

Unsupervised methods cover several types of methods such as frequent pattern mining (FQM), clustering, and neural networks. FQM based methods (e.g., SLCT [13] and LogMine [14]) extract frequently appeared patterns of tokens from a set of log data. This simple technique is useful, but it may misclassify multiply appeared variables (e.g., IP addresses) as descriptions. Clustering based approach (e.g., LKE [15], SHISO [16]) merges raw logs to log templates by clustering them with distance measure. Thus, similar raw log messages (with many same descriptions) are merged as a common log template. This approach has been well studied in the literature. A major drawback of this approach is that we need enough raw logs to cluster them, i.e., it is impossible to determine variables/descriptions from just one raw log message in theory. This disadvantage is especially crucial for network log analysis because the distribution of the log's appearance is highly skewed (or long-tailed). The neural network used in the NLP (Natural Language Processing) field has also recently been studied in log template generation [17].

Supervised methods achieve better accuracy than unsupervised ones thanks to labeled datasets. In particular, NLP-based technique has been applied to log template generation. Ref. [18] demonstrates that a CRF-based method outperforms unsupervised methods. However, the biggest concern is how to prepare an appropriate number of labeled data.

Our approach: As seen in the literature, we point out a clear trade-off in log template generation: accuracy (i.e, quality of generated template) and human resources (i.e., labeling). The novelty of our work is to rely on transfer learning to overcome this trade-off. Transfer learning has been a promising approach in many fields, e.g., image processing and NLP [19]. The key idea of transfer learning is to apply knowledge learned previously to solve new problems faster or with a better solution. In our context, this means that known log templates available from open-source software help in generating new log templates for proprietary network equipment. The transfer learning is also recently applied to anomaly detection in different vendor's log outputs [20], though our approch aims at generating log templates.

III. Design and Implementation

In this section, we describe our proposed deep transfer neural network (DTNN) model for log template generation problem. We first introduce an abstract framework and then explain each component.

A. Overview

Figure 1 shows an overview of LogDTL consisting of four main components: Dataset, Preprocessing, Deep Neural Network (DNN), and Label Mapping component.

The key idea of DTNN model is to transfer knowledge learned from a dataset to another dataset. Thus, we have two original datasets, including the source dataset (Source task - ST) and target dataset (Target task - TT). We first parse a raw log message into the header information (i.e., timestamp and hostname) and sequence of words corresponding to the free-format log statement. Next, we prepare training and testing for both source task and target task datasets. After that, we put the dataset into the DNN component in which we learn both tasks continuously, for transferring knowledge from source task to target task happened here when both models share the same parameters of DNN. We will discuss this in § III-C. The output of the DNN component is the labeled sentences, which containing Description or Variable. The Label Mapping component transforms labeled sentences into Log Templates.

B. Preprocessing

In this component, at first, we extract the "Header" part and "Message" part from the raw log as in [21]. The header part has a structured format and contains information about log recordings such as timestamp, source, and hostname depending on the configuration of the logging system. The message part corresponds to an unstructured statement and presents the system behaviors in a given device. The message part is free-format but partially forms a kind of formal languages, because the statement part is output by filling the replacers (e.g., format specifiers) with variables. Therefore, it can be split into a sequence of words like other languages in many cases. In our case, we will use this message part to form our log template generation model.

After splitting each log message into a sequence of words (tokens) in source task and target task, we split each task into a preliminary annotated dataset (e.g., training set, X_{st} or X_{tt}) and a remaining raw dataset (e.g., testing set, Y_{st} and Y_{tt}) as in Figure 1. (X_{st}, Y_{st}) are used for learning DNN's source task. (X_{tt}, Y_{tt}) are used for learning DNN's target task. Transfer learning leverages the sharing parameters process from DNN's source task to DNN's target task. Note that, this transfer model can theoretically predict the testing set of target task without the training set of target task by using shared knowledge from source task (§ III-C4 in detail). However, the performance of the model may be less effective since it still needs a piece of specific information from the target domain (§ V in detail).

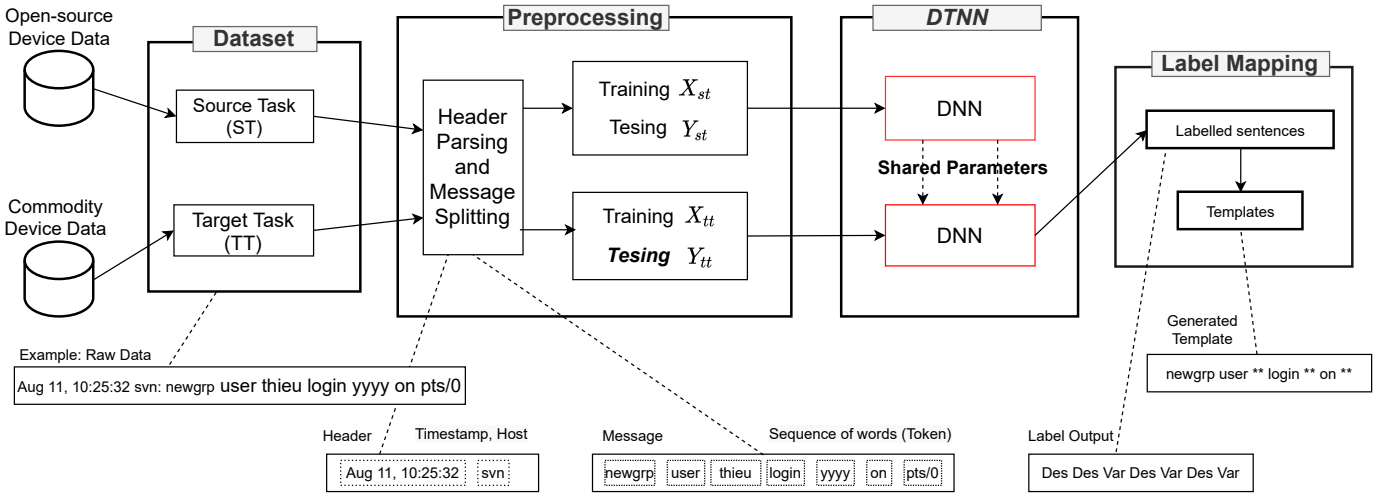

Fig. 1. LogDTL: Log template generation using Deep Transfer Learning

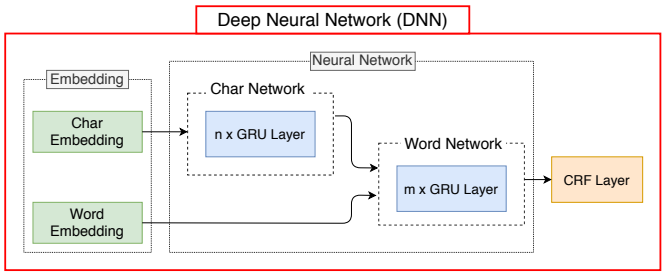

Fig. 2. Deep Neural Network (DNN) for sequence tagging

C. Deep Transfer Neural Network

Since the log template estimation classifies each word of each log message into Description or Variable, we can consider this problem as the sequence tagging task (two tags: Description and Variable [18]). Therefore, our transfer learning component can be designed based on a well-known hierarchical recurrent neural network model (including Embedding, Neural Network, and CRF) as shown in Figure 2.

The advantage of DTNN is the use of both character-sequence and word-sequence as input. The character-level network receives a series of letters (encoded by char-embedding component) and outputs a new representation of morphological information at the character level. The word-level network takes a series of words (encoded by word-embedding component) and then combines them with feature representation from the character-level network to produce a new feature representation. This feature is then fed into a CRF layer that outputs the label for sequence. With this designation, the word-network and char-network can be deployed as recurrent neural networks (RNNs) or convolutional neural networks (CNNs) ([22], [23] and [24]).

1) Embedding technique: We need to represent the log sequences as a vectorized (numerical) representation for the input. There are two ways frequently used for vectorizing a text: one-hot encoding and distributed encoding. Due to the disadvantages of one-hot encoding representation such as dimension explosion [25], we use distributed representation to vectorize the log sequences in this paper. In this way, each log message is presented by a low-dimensional dense vector, such as Word2Vec [26]. This is helpful to avoid the dimension explosion, especially for long sequences.

2) Neural Network: **For the advantages of RNNs, such as easy to understand and simple implementation, we select the GRU (Gated Recurrent Unit) [27] as a core network for both char-network and word-network. Our char-network is the stack of n GRU layers, and the word-network is the stack of m GRU layers as in Figure 2. The input of the char-network is the embedding at character-level while the input of the word-network is the concatenation of the last char-network's hidden state and the embedding at word-level. In this paper, both GRU networks are bi-directional and have two layers ($m = n = 2$).**

3) CRF: **Based on the max-margin principle [28], the CRF layer defines an objective function to maximize. The technical details is shown in [29]. With the input sequence of words, the output of the CRF layer will be the sequence of label (including Des and Var in our case).**

4) Transfer Learning: There are several architectures of transfer learning corresponding to DNN [29] such as cross-domain transfer, cross-application transfer, or cross-lingual transfer. Our task, generating log templates from raw log messages, can be considered as a sequence labeling problem in which each word in a log message can be classified into Description or Variable. Thus, we design the DTNN as cross-domain transfer and assume that we have a few labels in the target domain. We share all the model weights and feature representation in the DNN (the red rectangle in Figure 1), including the word and character embedding, the word-network, the char-network, and the CRF layer. We perform a label mapping step on top of the CRF layer, as shown in Figure 1.

The difference between DNN and DTNN is as follows:

- DNN treats source task and target task separately. The training algorithm for DNN is the same as other neural networks: using gradient descent and back-propagation algorithm or its improved version such as AdaGrad [30].
- DTNN trains both the source task and target task jointly at the same time.

Next, we explain how to train DTNN. Suppose we want to transfer the knowledge learned from a source task st to a target task tt, with the training set X_{st} and X_{tt}. Suppose W_{st} and W_{tt} are the set of model parameters for the source and target tasks respectively. The model parameters are divided into two sets, task-specific parameters $W_{task,spec}$ and shared parameters W_{shared}, i.e.,

$$W_{st} = W_{st,spec} \cup W_{shared}, W_{tt} = W_{tt,spec} \cup W_{shared} \quad (1)$$

where shared parameters W_{shared} are jointly optimized by the two tasks, while task-specific parameters $W_{st,spec}$ and $W_{tt,spec}$ are trained for each task separately.

The training procedure consists of four steps: (1) At each iteration, the sampled task is randomly selected between st and tt. (2) A batch of training instances is generated and then performed a gradient update according to the loss function of the given task (The AdaGrad algorithm above is used for computing the learning rates dynamically in each generation). (3) Updating both task-specific parameters and shared parameters. (4) Repeating the step (1) to (3) until stopping. Since our focus is on the target task and there is a difference between the source task's convergence rate and target task's one, the early stopping is used on the target task.

In the case that the training set of the target task does not exist, the source task can be considered as a pre-trained model. Also it will be used to classify the label for the testing set of target tasks only. The training procedure in this case is composed of three steps: (1) At each iteration, updating gradient using the loss function in the source task. (2) Classifying the testing set of the target tasks. (3) Repeating the step (1) and (2) until the training procedure of source task finishes. As can be seen, the shared parameters are updated based on the source task only. So the performance for the testing set of the target tasks may be less efficient. Since it learns nothing from target tasks.

D. Label Mapping

The output of the DNN component is labeled sentences, consisting of a sentence of mixed Description and Variable. Finally, we generate log templates by matching the labeled sentences to the sequence of words in Preprocessing module.

IV. Experiment Settings

A. Dataset

We use two raw log datasets (AV and S4 datasets) collected at backbone routers in two research and education networks in Japan. The AV dataset (203490 raw logs) is generated by a Vyatta open source router in APAN-JP, and the S4 dataset (149406 raw logs) is generated by proprietary routers in SINET4. Two different types of logs are included in the dataset corresponding to two vendors of equipment. We focus on routing and interface-related log templates that are commonly used in both networks.

For the AV dataset, we have both raw logs and its labeled templates from the source code of open source Vyatta [12]. We group all labeled templates into 45 independent and different clusters. Meanwhile, for the S4 dataset, we have to manually generated labeled templates for all raw logs and use them as ground truth. We also clustered its labeled templates into 75 clusters. Note that these log datasets are parsed into a sequence of statement words before log template generation with amulog [21].

TABLE I
Dataset

	Source Task (AV)		Target Task (S4)	
	Train (SX)	Test (SY)	Train (TX)	Test (TY)
# Logs	100000	100000	74496	74910
# Templates	100000	100000	74496	74910
# Clusters	38	45	41	75

Since our goal is to generate log templates for proprietary network equipment based on transfer learning, the AV dataset will be used in the source task, and the S4 dataset will be used in the target task. We split both datasets into training and testing sets (Table I). **We use random selection for each training and testing set with the rate of 50% in both datasets. After that, we re-calculate the clusters for each set in both tasks. The number of clusters for each set is shown in Table I.**

B. Performance Metrics

We evaluate the performance of the test models in two aspects: accuracies and processing time. We compare our model DTNN with two state-of-the-art models: CRF and DNN. For each test case, we perform ten trials with train data selected randomly and independently, and we use their average (mean) and standard deviation (std), considering dependencies on the selection of the training set.

The performance metrics we used is as follows:

- Word accuracy [18]: Assigned label is validated in word level. This checks whether the label of a description word (or variable word) is "description" ("variable") or not.
- Line accuracy [18]: Assigned labels are validated in a log message. It is failed if at least one label in a log message does not match.
- Template word accuracy: is the average line accuracy weighted by the number of the appearance of log messages in a log template.
- **F accuracy in this paper is one of the clustering metrics and is commonly called Pair-wise F-measure [31]: for each pair of log messages (ground truth and prediction), the score checks whether the relation of**

TABLE II
PERFORMANCE EVALUATION

# Train. Data	Methods	Accuracy Metrics				Timing Metrics	
		Word Acc.	Template Word Acc.	F Acc.	Line Acc.	Training T.	Testing T.
0	DTNN	0.898 ± 8.73E-7	0.838 ± 5.45E-6	0.797 ± 1.07E-5	0.341 ± 7.04E-6	0.341 ± 7.04E-6	0.341 ± 7.04E-6
1	CRF	0.779 ± 0.025	0.772 ± 0.063	0.828 ± 0.115	0.228 ± 0.081	2E-7 ± 0.0008	4.31 ± 0.07
	DNN	0.677 ± 0.064	0.603 ± 0.08	0.888 ± 0.11	0.163 ± 0.13	27.80 ± 0.2099	82.28 ± 0.232
	DTNN	**0.912 ± 0.02**	**0.871 ± 0.022**	**0.949 ± 0.088**	**0.442 ± 0.103**	1877.56 ± 3.996	171.34 ± 9.362
10	CRF	0.932 ± 0.025	0.818 ± 0.067	**0.991 ± 0.005**	0.786 ± 0.027	0.01 ± 0.0004	4.3 ± 0.093
	DNN	0.958 ± 0.031	0.804 ± 0.03	0.956 ± 0.045	0.801 ± 0.101	31.47 ± 0.1487	82.69 ± 0.452
	DTNN	**0.967 ± 0.028**	**0.869 ± 0.033**	0.950 ± 0.057	**0.858 ± 0.111**	2169.65 ± 3.780	185.88 ± 1.455
100	CRF	0.990 ± 0.004	0.901 ± 0.04	0.998 ± 0.001	0.940 ± 0.021	0.02 ± 0.0029	4.39 ± 0.114
	DNN	0.994 ± 0.003	0.895 ± 0.014	**0.999 ± 0.0002**	**0.976 ± 0.011**	74.74 ± 0.0955	82.80 ± 0.462
	DTNN	**0.996 ± 0.001**	**0.926 ± 0.007**	0.993 ± 0.004	0.972 ± 0.011	2305.77 ± 11.915	186.39 ± 0.891
1000	CRF	**0.999 ± 0.0003**	0.933 ± 0.002	**1.000 ± 5.4E-05**	**0.993 ± 0.001**	0.08 ± 0.028	4.26 ± 0.084
	DNN	0.998 ± 0.0002	0.921 ± 0.009	0.999 ± 0.0002	0.989 ± 0.002	74.70 ± 0.091	82.74 ± 0.29
	DTNN	**0.999 ± 0.0003**	**0.937 ± 0.003**	**1.000 ± 0.0003**	**0.993 ± 0.0016**	2704.20 ± 8.487	186.17 ± 0.591
10000	CRF	0.999 ± 4.36E-5	0.943 ± 0.002	1.000 ± 5.1E-9	0.996 ± 0.0003	4.32 ± 1.2473	4.42 ± 0.088
	DNN	0.998 ± 0.0003	0.927 ± 0.007	1.000 ± 0.0003	0.989 ± 0.0017	132.66 ± 0.058	83.11 ± 0.258
	DTNN	**1.000 ± 1.52E-5**	**0.961 ± 0.002**	1.000 ± 1.32E-5	**0.998 ± 9.3E-5**	4032.86 ± 7.79	185.42 ± 0.884

the pair is correct in terms that"the pair is in the same cluster" or "the pair is not in the same cluster".

V. Results and Discussion

A. Prediction Accuracy

Table II shows the (mean ± std) results of tested models with different performance metrics. Training data represents the number of training instances of target task used for each model in each test case. Since CRF and DNN require a training dataset, in the case of 0 training dataset, only our transfer learning model DTNN has the results.

1) DNN vs CRF model: First, we compare the results of DNN and CRF models in order to evaluate the effects of CRF extensions such as neural network, word embedding, and character embedding in DNN. From Table II, we see that DNN only outperformed CRF by F accuracy metric for small training datasets (N $\leq$ 100). Meanwhile, with enough training dataset (N = 1000), CRF got better results than the DNN model in most performance metrics. Thus, thanks to the extensive component in DNN (word embedding, character embedding, and neural networks), DNN is more accurate than CRF for the small training dataset. On the other hand, many input features for word and character embeddings might make noise to the model for enough training data.

2) DTNN vs CRF model: Next, we compare the results of the DTNN model and CRF model to evaluate the impact of transfer knowledge from the source task to the target task. In our design, the source task is from open-source software, and the target task is from proprietary software. From Table II, for the small training dataset, DTNN always gives better results than CRF. In particular, with just one training example, DTNN outperforms CRF on all performance metrics. However, with a large training dataset, the results between CRF and DTNN are in-significantly different. For example, for N = 1000, both CRF and DTNN got 99.9% of word accuracy, 100% of F accuracy, and 99.3% of Line Accuracy. Besides, when comparing the Template Word Accuracy, we see that DTNN always achieves better results than CRF, regardless of the larger or small amount of training data, demonstrating that knowledge from source task has helped bring efficiency to the target task in our model.

Note that one might think that DTNN achieves the best performance due to a possibility that source and target templates could be overlapped. We tested CRF with S4 raw logs and Vyatta's templates, but the accuracy of this model was quite low. Thus, The overlap of two sets of the templates is small, and knowledge transfer is the essential part of the performance gain.

B. Processing time

Next, we discuss the processing time overhead of the proposed method. All the experiments are conducted with a commodity workstation (CPU: Xeon W-2102 2.90GHz, Memory: 256GB, GPU: GTX1080). As shown in Table II, DTNN requires 2000-4000s for training and 180s for testing; they are 10-100 times larger than those in CRF and DNN. Thus, this result demonstrates another trade-off, i.e., accuracy/human resources vs. processing time.

C. Case study

Finally, as a case study, we compare ground truth templates (manually labelled) with generated log templates (N=1) as listed in Table III. Templates 1 and 2 are related to multicast and MTU change, respectively. In both templates, DTNN generates the correct templates. On the other hand, CRF misclassifies interface names and addresses as Description due to less enough training data, and DNN remains one digit as Description likely due to lack of enough knowledge transfer. These results clearly demonstrate the validity of our transfer learning approach.

VI. Conclusion

We propose a template generation method based on deep transfer neural network in order to overcome the accuracy issue of unsupervised learning models and the drawback of lacking training data in supervised learning models in the

TABLE III
CASE STUDY: "**" REPRESENTS VARIABLE, AND WORDS WITH BRACKET SHOW MISCLASSIFICATION.

method	**template1**	**template2**
ground truth	rpd ** EVENT MTU ** index ** Up Broadcast P2P Multicast addr ** **	/kernel MTU for ** reduced to **
CRF	rpd ** EVENT MTU (ifname) index ** Up Broadcast P2P Multicast addr ** (v6 addr)	/kernel MTU for (v6 addr) reduced to **
DNN	** ** EVENT MTU ** ** ** Up Broadcast ** ** ** (num) **	/kernel MTU for ** reduced to **
DTNN	rpd ** EVENT MTU ** index ** Up Broadcast P2P Multicast addr ** **	/kernel MTU for ** reduced to **

template generation problem. Our approach opens up new perspectives for the log template generation.

We validated the effectiveness of our approach with datasets from backbone routers at research and education networks in Japan. Our evaluation results show that DTNN outperforms two state-of-the-arts (DNN and CRF) models across different test cases (small/medium/large training resources) and with different performance metrics. We also confirmed that DTNN works in reasonable processing time for testing.

In the future, we will investigate the validity of our approach for the larger class of log template categories.

ACKNOWLEDGEMENTS

This work is supported by the NII Internship program, JSPS KAKENHI Grant number JP19K20262, and the MIC/SCOPE #191603009.

REFERENCES

[1] T. Kurimoto *et al.*, "Sinet5: A low-latency and high-bandwidth backbone network for sdn/nfv era," in *IEEE ICC'17*, 2017, pp. 1–7.

[2] T. Li, J. Ma, and C. Sun, "Dlog: diagnosing router events with syslogs for anomaly detection," *The Journal of Supercomputing*, vol. 74, no. 2, pp. 845–867, 2018.

[3] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 207–218.

[4] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *ACM CCS'17*, 2017, pp. 1285–1298.

[5] S. Zhang *et al.*, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *IEEE/ACM IWQoS'17*, 2017, pp. 1–10.

[6] P. Wang, G. R. Bai, and K. T. Stolee, "Exploring regular expression evolution," in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 502–513.

[7] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *IEEE International Conference on Web Services (ICWS)*, 2017, pp. 33–40.

[8] W. Meng *et al.*, "Logparse: Making log parsing adaptive through word classification," in *IEEE ICCCN'20*, 2020, pp. 1–9.

[9] M. Cinque, D. Cotroneo, and A. Pecchia, "Event logs for the analysis of software failures: A rule-based approach," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 806–821, 2012.

[10] W. Shang, Z. M. Jiang, B. Adams, A. E. Hassan, M. W. Godfrey, M. Nasser, and P. Flora, "An exploratory study of the evolution of communicated information about the execution of large software systems," *Journal of Software: Evolution and Process*, vol. 26, no. 1, pp. 3–26, 2014.

[11] M. Zhang, Y. Zhao, and Z. He, "Genlog: Accurate log template discovery for stripped x86 binaries," in *IEEE COMPSAC'17*, 2017, pp. 337–346.

[12] Y. Yamashiro, S. Kobayashi, K. Fukuda, and H. Esaki, "Network log template generation from open source software," *IEICE Technical Report (in japanese)*, vol. 118, no. 204, pp. 15–22, 2018.

[13] R. Vaarandi, "A data clustering algorithm for mining patterns from event logs," in *IEEE IPOM'03*, 2003, pp. 119–126.

[14] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *ACM International on Conference on Information and Knowledge Management*, 2016, pp. 1573–1582.

[15] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *IEEE ICDM'09*, 2009, pp. 149–158.

[16] M. Mizutani, "Incremental mining of system log format," in *2013 IEEE International Conference on Services Computing*, 2013, pp. 595–602.

[17] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised log parsing," *arXiv preprint arXiv:2003.07905*, 2020.

[18] S. Kobayashi, K. Fukuda, and H. Esaki, "Towards an nlp-based log template generation algorithm for system log analysis," in *International Conference on Future Internet Technologies*, 2014, pp. 1–4.

[19] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, 2020.

[20] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu, "Logtransfer: Cross-system log anomaly detection for software systems with transfer learning," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 37–47.

[21] S. Kobayashi, Y. Yamashiro, K. Otomo, and K. Fukuda, "amulog: A general log analysis framework for diverse template generation methods," in *IEEE/IFIP CNSM'20*, 2020, pp. 1–5.

[22] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, pp. 2493–2537, 2011.

[23] J. P. Chiu and E. Nichols, "Named entity recognition with bidirectional lstm-cnns," *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 357–370, 2016.

[24] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF," in *Annual Meeting of the Association for Computational Linguistics*, 2016, pp. 1064–1074.

[25] R. Yang, D. Qu, Y. Gao, Y. Qian, and Y. Tang, "Nlsalog: An anomaly detection framework for log sequence in security management," *IEEE Access*, vol. 7, pp. 181 152–181 164, 2019.

[26] K. W. Church, "Word2vec," *Natural Language Engineering*, vol. 23, no. 1, pp. 155–162, 2017.

[27] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[28] K. Gimpel and N. A. Smith, "Softmax-margin crfs: Training log-linear models with cost functions," in *Human Language Technologies: Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010, pp. 733–736.

[29] Z. Yang, R. Salakhutdinov, and W. W. Cohen, "Transfer learning for sequence tagging with hierarchical recurrent networks," *arXiv preprint arXiv:1703.06345*, 2017.

[30] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[31] S. Basu, A. Banerjee, and R. J. Mooney, "Active semi-supervision for pairwise constrained clustering," in *Proceedings of the 2004 SIAM international conference on data mining*. SIAM, 2004, pp. 333–344.