Network Diagnostics Using Passive Network Monitoring and Packet Analysis

Martin Holkovič

CESNET a.l.e.
Zikova 1903/4
Prague 16000, CZ
Email: holkovic@cesnet.cz

Ondřej Ryšavý

Faculty of Information Technology
Brno University of Technology
Brno 61266, CZ
Email: rysavy@fit.vutbr.cz

***Abstract*—Finding a problem cause in network infrastructure is a complex task because a fault node may impair seemingly independent components. On the other hand, most communication protocols have built-in error detection mechanisms. In this paper, we propose to build a system that automatically diagnoses network services and applications by inspecting the network communication automatically. We model the diagnostic problem using a fault tree method and generate a set of rules that identify the symptoms and link them with possible causes. The administrators can extend these rules based on their experiences and the network configuration to automatize their routine tasks. We successfully deployed the proof-of-concept tool and found interesting future research topics.**

Keywords–Network diagnostics; passive network monitoring; rule-based diagnostics; fault tree analysis; event-based diagnostics.

I. Introduction

Network infrastructure and applications are complex, prone to cyber attacks, outages, performance problems, misconfiguration errors, and problems caused by software or hardware incompatibility. All these problems may affect network performance and user experience [1] which may cause fatal problems in critical networks (e.g., E-health, Vanet, Industrial IoT).

Many network administrators do not have the proper tools or knowledge to diagnose and fix network problems effectively, and they require an automated tool to diagnose these errors [2]. Zeng et al. [3] provide a short survey on network troubleshooting from the administrators' viewpoint identifying the most common network problems: *reachability problems, degraded throughput, high latency, and intermittent connectivity*. The consulted network administrators expressed the need for a network monitoring tool that would be able to identify such problems.

This paper proposes a system which creates diagnostic information only by performing passive network traffic monitoring and packet-level analysis. Previous research and development provided tools for helping administrators to diagnose faults [4] and performance problems [5]. However, these tools either require *installation of agents on hosts, active monitoring, or providing rich information about the environment*.

One of the most common ways of analyzing network traffic is by using a network packet analyzer (e.g., Wireshark). The analyzer works with captured network traffic (PCAP files) and displays structured information of layered protocols contained in every packet (encapsulated protocols, protocol fields). Administrators work with this information, check transferred content and compare the data with expected values. This process, done manually, is time-consuming and requires a good knowledge of network protocols and technologies.

The main contribution of this paper is a proposal of a tool for automatic diagnoses of network related problems from network communication only. Our approach tries to imitate a diagnostic process of a real administrator using the fault tree method and a popular packet parsing tool tshark. We have also implemented a proof-of-concept implementation to confirm the viability of the approach.

The paper is organized as follows. Section 2 defines the problem statement and research questions. Section 3 discusses related work and describes diagnostic approaches. Our solution consists of three stages and is introduced in Section 4. Section 5 instructs network administrators how to use our system (proof-of-concept) and shows how we model diagnostic knowledge. Finally, Section 6 is the conclusion which summarizes the current state and proposes future works.

II. Research Questions

Our primary goal is to design a system that infers possible causes accountable for network related problems, such as service unreachability or application errors. Offering a list of actions for fixing the errors' cause is the secondary and optional goal. All this information is gathered only from captured network communication.

In our work, we focus on enterprise networks that have complex networking topologies, usually consisting of heterogeneous devices. We expect that the administrators will collect network communication on appropriate places and validate its consistency before the analysis.

To achieve our goal, we need to find answers to the following research questions:

1) How to model different network faults in a suitable way for implementation in a diagnostic system? *Reachability, application specific, and device malfunctioning problems* can cause various networking issues. We need to have a unified approach for modeling these problems to identify the symptoms and link them with root causes.
2) What information should be extracted from the captured network communication to identify symptoms of failures?

 ISBN: 978-1-61208-711-5

In our case, we can passively access the communication in the monitored network and extract the necessary data to detect possible symptoms. An approach that can efficiently detect the symptoms in terms of precision and performance is needed.

3) How to identify the root cause of the problem, if we have a set of identified symptoms? The core part of the diagnostic engine is to apply knowledge gathered from observed symptoms to infer the possible root cause of the observed problem. The result should provide the information in sufficient detail. For instance, if the process on server crashed, then we would like to know this specific information instead of a more general explanation (e.g., a host failure has occurred).
4) What list of actions can we give to the administrator to fix the problems? Based on the observed symptoms and the root cause, the system should be able to provide fixing guidelines. These guidelines are supposed to be easy to understand even for an inexperienced administrator.

III. Related Work

A lot of research activities were dedicated to the diagnoses of network faults. Various methods were proposed for different network environments [4], in particular, home networks [6], enterprise networks [7]–[10], data centers [5], backbone and telecommunications networks [11], mobile networks [12], Internet of Things [13], Internet routing [14] and host reachability. Methods of network troubleshooting can be roughly divided into the following classes:

Active methods use traffic generators to send probe packets that can detect the availability of services or check the status of applications [15]. Usually, generators create diagnostic communication according to the test plan [7]. The responses are evaluated and provide diagnostic information that may help to reveal device misconfiguration or transient fail network states. Diagnostic probes introduce extra traffic, which may pose a problem for large installations [10]. Also, active methods may rely on the deployment of an agent within the environment to get information about the individual nodes [8].

Passive methods detect symptoms from existing data sources, e.g., traffic metadata [11], traffic capture files, network log files [14], performance counters. Passive methods can utilize the data provided by network monitoring systems.

Of course, the proposed systems also combine passive traffic monitoring to detect faults with active probing to determine the cause of failure. Identifying anomalies related to network faults and linking them with possible causes can be done by using one of the following approaches:

Inference-based approach uses a *model* to identify the dependence among components and to infer the faults using a collection of facts about the individual components [8], [16].

Rule-based approach uses *predefined rules* to diagnose faults [9]. The rules identify symptoms and determine how these contribute to the cause. The rules may be organized in a collaborative environment for sharing knowledge between administrators [6].

Classifier-based approach *requires training data* to learn the normal and faulty states. The classifier can identify a fault and its likely cause [17].

Network diagnostics based on traffic analysis can also use methods proposed for anomaly detection as some types of faults result in network communication anomalies.

Main contributions of our solution:

- automation of the tool Wireshark - Wireshark is a well-known protocol analyzer but lacks any task automation;
- the result is well understandable - the result contains steps which a real administrator would execute;
- easily extendable list of rules - the rules use Wireshark display filter language [18].

IV. Proposed System Architecture

We have built a proof-of-concept expert system to analyze network traffic. The system combines rule-based and inference-based approaches. We will not use a classifier-based approach [19], because it requires too much training data and only returns the root cause of the problem and not how it relates to the detected symptoms. Another benefit of the rule-based approach is that we can cover very specific situations for which getting training data could be very problematic.

We are focusing purely on passive methods because active methods are generating additional traffic into diagnosed networks (which is not acceptable for us) and also because this way, an administrator can perform an offline analysis on a computer not connected to the diagnosed network.

The proposed system processes the input data in several stages as shown in Figure 1. The first stage labeled as *Protocols Analyzer* filters and decodes input packets using an external tool. The second stage named *Events Finder* executes simple rules to identify events significant from the diagnostics point of view. In the third stage (*Tree Engine*), decision trees identify the possible problem cause and create a diagnostic output. All stages are easily extendable by the administrator who can add new rules and definitions.

Our proposed approach can also use different data sources (e.g., log files) as shown in Figure 1. *Events Finder* searches through data using analyzers specific to each data source. Additional analyzers could increase the diagnostic capability, however in our research, we are focusing only on network data, and we leave other possibilities for future research.

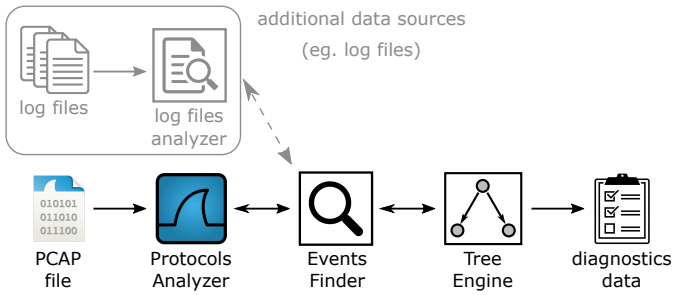

Figure 1. Top level architecture design of all the proposed system stages. The gray area represents optional extensions — additional data sources.

A. Protocols Analyzer

The first step in the processing pipeline is decoding captured network traffic in the PCAP format into a readable JSON format. We employ the tool *tshark*, which is a command

line version of the widely-used network protocol analyzer Wireshark. Because tshark follows the field naming convention used by Wireshark, we can use Wireshark Display Filter Expressions to select packet attributes. Tshark supports all packet dissectors available in Wireshark. Using tshark brings the following benefits:

- huge support of network protocols and when a new protocol is created, community can implement parsers very quickly and for free;
- adds tunneled, segmented and reassembled data support;
- tshark marks extracted data with the same names as displayed inside the Wireshark GUI. This allows a creation of easy-to-read API for diagnostics;

B. Events Finder

Events finder aims to identify events useful for network diagnosis (for example, a successful SMTP authentication event). An event rule consists of two parts: a list of packet filters and a list of assertions to express additional constraints. Both filter expressions and assertions use Wireshark's display filter language. Using this language, the expressions can be first tested in Wireshark before we use them in event finder rules.

The system evaluates the event rule as follows: (i) Each packet filter returns a list of packets matching the filter. (ii) Assertions are evaluated to select pairs of packets satisfying the constraints. A result has the form of a collection of pairs of packets, e.g., a rule that identifies DNS request-response pairs asserts that the transaction ID in both the request and response packets match. Assertion expressions use the display filter language extended with basic mathematical operations.

The event rules have the declarative specification written in YAML format. This format is described in subsection V-B. Rules are organized into modules. New modules can be easily added extending the rule database.

C. Tree Engine

The tree engine infers the possible error cause by evaluating a decision tree that contains expert knowledge about supported network protocols and services. Each node of the tree contains a diagnostic question. Questions refer to events identified by the *Event Finder*. Paths in the tree represent gathered knowledge and lead to the possible cause of the problem. Along the path, a diagnostic report is created to provide additional information for experienced users. The diagnostic report is produced in a human-readable format, as well as in a machine format useful for further processing or visualization.

The decision tree is comprised of the declarative specification of tree nodes enriched by Python code. Injection of Python code into the tree node definitions enables us to do complex knowledge processing. The idea is to keep the declarative part simple enough for most of the use-cases. The Python code is needed for specific use-cases, where a custom processing logic is necessary. The tree is defined using the YAML format rules, and subsection V-A describe its syntax.

V. Rule Specification

Diagnostic engine defines each protocol as a decision tree. The tree consists of nodes representing administrator questions, and edges representing answers to these questions. The edge

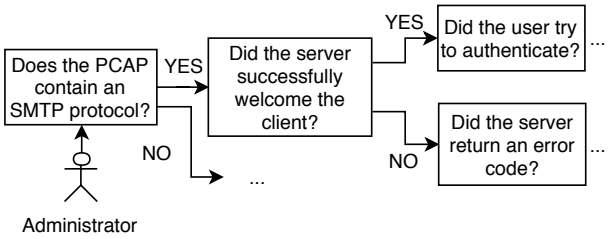

Figure 2. A simple illustration of a binary decision tree. Administrator diagnoses SMTP problem by checking questions in the predefined order.

can move the diagnostic process from one question to another or finish the process with the discovered result.

The questions simulate thinking of a real administrator. Typically, an administrator starts to search for certain network packet values and after the search for them is finished, the administrator searches for next values based on the result. In our solution, each question can only have two answers: success or fail. This yields a binary decision tree. Figure 2 shows an example of a small portion of the SMTP tree.

We need to convert the decision tree to a format understandable by our system. This conversion is split into two steps: 1) defining tree nodes (*Tree node rules*) and 2) defining conditions for choosing tree nodes (*Event condition rules*). The following subsection describes the syntax for both tree node rule and event condition rule. The reason why a node rule does not contain a condition code directly is that multiple rules would not be able to use the same condition code (reusability).

Conversion assigns a name to each node (*label_id*). We use the node names as labels for switching from one node to another. Each node has a condition (*condition_rule_id*), defined as an *Event condition rule*, used for choosing the next diagnostic step. Each rule can have one or none success and fail branch (*branch_code*). Branches contain executable Python code and the next node rule name. After the execution of the Python code, the analysis switches to the next node. Figure 3 shows the pseudocode for writing tree nodes.

```
1 label_id:
2   if (condition_rule_id):
3     success branch_code
4   else:
5     fail branch_code
```

Figure 3. Pseudocode for writing a tree node. Each node should have a unique id, condition, and branch codes.

A. Tree Node Rules

Each rule consists of an *event condition rule* name which should be executed, next states and blocks of Python code. The Python code can process packet data, make logical decisions and most importantly, generate diagnostic output. Instead of writing the whole output inside these rules, the rule contains only the name of the event. Each rule can switch to another protocol rule to diagnose problems across several protocols, e.g., if an SMTP communication is not detected, we will check if there are any ICMP unreachable messages, failed TCP connection attempts or incorrect DNS resolutions. Figure 4 shows an example of one rule defying the middle node from the tree in Figure 2.

 ISBN: 978-1-61208-711-5

```
id: smtp detected # name of the rule
query: welcome ok? # Events Finder rule
success:
  state: client welcomed # next state
  code: | # Python code follows
    event("client_welcomed")
fail:
  state: check error # next state
  code: | # Python code follows
    event("client_not_welcomed")
```

Figure 4. Simple Tree Engine rule showing what should be done if SMTP server welcomed the client or not.

B. Event Condition Rules

Rules in this section describe *how* the question is converted into packet lookup functions. Each rule may look for several independent packets, which are combined and checked if their relation fulfills the assert condition. Each question returns a list of tuples, where a tuple represents packets fulfilling the assert condition. Figure 5 shows an example of a simple rule for the question *Did the server successfully welcome the client?* Section *facts* looks for any hello commands and OK responses. The system puts founded packets which belong together into touples based on the *asserts* section.

```
id: welcome ok? # name of the rule
facts: # which packets we are looking for
  command: smtp.req.command in {"HELO"
           "EHLO"}
  reply: smtp.response.code == "250"
asserts: # packets relation constrain
 -command[tcp.stream]==reply[tcp.stream]
 -command[tcp.ack]==reply[tcp.seq]
```

Figure 5. Example of SMTP rule for checking if the server welcomed the client or not.

SMTP: Connection detected
SMTP: Server welcomed the client
SMTP: Server is ready
SMTP: Authentication 'gurpartap@patriots.in' - ok
SMTP: The communication is not encrypted
SMTP: No email has been sent
SMTP: Transaction error code 552 - Requested mail actions aborted - Exceeded storage allocation
SMTP: Empty email account storage (check SPAM folder) or increase the account quota.

Figure 6. An example of diagnostic output for an SMTP error. After an error 552 is detected and translated into human-readable error description, the system proposes a list of actions for fixing the error.

Before executing the diagnostic process, it is necessary to define event names from the *Tree engine* rules. A definition is just a simple dictionary which contains a severity and a description message. Part of the event description can be a pointer to another dictionary, which translates error codes to a human readable format. For example, instead of SMTP error code 552, the message "Requested mail actions aborted - Exceeded storage allocation" is displayed. After all rules and events are defined, it is possible to execute the diagnostic process. Figure 6 shows an example of one diagnostic output.

Table 1. Supported protocols and amount of rules and success, warning, error events which describe various protocol behavior situations.

Protocol	Node rules	Condition rules	Events		
			Success	Warning	Error
DHCP	25	23	10	9	4
DNS	12	12	8	2	6
FTP	24	10	17	5	6
ICMP	4	2	0	0	4
IMAP	15	8	7	0	11
POP	21	7	8	5	10
SIP	38	22	15	1	8
SLAAC	8	7	1	5	2
SMB	27	25	20	4	5
SMTP	17	13	10	5	9
SSL	1	1	1	0	1
TCP	11	11	0	8	3

VI. Conclusion

This paper presents a proposal of a system intended for troubleshooting network problems based on a passive network traffic analysis. The primary goal is to automate network diagnostics to help network administrators find causes of problems. The core of the presented approach is a multistage processing pipeline combining rule-based and inference-based methods. We have completed the implementation of a proof-of-concept system that we will use for preliminary evaluation and experiments.

We have implemented diagnostic rules for several application and service protocols. Table 1 shows the current list of supported protocols and their complexity in term of Node and Condition rule count, and their capabilities in term of Event count. After an evaluation of our solution by our partner — a monitoring vendor company, we have concluded, that the system must mark all reports which our tool may have incorrectly detected because of low-quality input data. For example, packet loss can drastically decrease the quality and accuracy of diagnostic results. In the current system, all reports from TCP flows with missing segments are marked as possibly incorrect.

Future work will focus on:

- evaluating the solution (accuracy and performance) and comparing the results with similar monitoring tools;
- analyzing each protocol's rules and based on used protocols and their field names create a filtering unit to reduce the amount of data processed by *Protocol Analyzer*;
- optimizing the performance. The current *Events Finder* combines all packets to check whether they are fulfilling the assert conditions or not. This all-to-all packet check has exponential time complexity ($2^{O(n)}$), which is unacceptable for large PCAP files. We want to optimize checking the asserts to decrease the complexity.

Acknowledgment

This work was supported by project "Network Diagnostics from Intercepted Communication" (2017-2019), no. TH02010186, funded by the Tech-nological Agency of the Czech Republic and by BUT project "ICT Tools, Methods and Technologies for Smart Cities" (2017-2019), no. FIT-S-17-3964.

 ISBN: 978-1-61208-711-5

REFERENCES

[1] R. Wang, D. Wu, Y. Li, X. Yu, Z. Hui, and K. Long, "Knights tour-based fast fault localization mechanism in mesh optical communication networks," Photonic Network Communications, vol. 23, no. 2, 2012, pp. 123–129.

[2] M. Solé, V. Muntés-Mulero, A. I. Rana, and G. Estrada, "Survey on models and techniques for root-cause analysis," arXiv preprint arXiv:1701.08546, 2017.

[3] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "A survey on network troubleshooting," Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep., 2012.

[4] M. Łgorzata Steinder and A. S. Sethi, "A survey of fault localization techniques in computer networks," Science of computer programming, vol. 53, no. 2, 2004, pp. 165–194.

[5] C. Guo et al., "Pingmesh: A large-scale system for data center network latency measurement and analysis," in ACM SIGCOMM Computer Communication Review, vol. 45, no. 4. ACM, 2015, pp. 139–152.

[6] B. Agarwal et al., "Netprints: Diagnosing home network misconfigurations using shared knowledge," in Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, ser. NSDI'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 349–364.

[7] L. Lu, Z. Xu, W. Wang, and Y. Sun, "A new fault detection method for computer networks," Reliability Engineering & System Safety, vol. 114, 2013, pp. 45–51.

[8] S. Kandula et al., "Detailed diagnosis in enterprise networks," ACM SIGCOMM Computer Communication Review, vol. 39, no. 4, 2009, pp. 243–254.

[9] M. Luo, D. Zhang, G. Phua, L. Chen, and D. Wang, "An interactive rule based event management system for effective equipment troubleshooting," in IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society. IEEE, 2011, pp. 2329–2334.

[10] A. Mohamed, "Fault detection and identification in computer networks: A soft computing approach," Ph.D. dissertation, University of Waterloo, 2010.

[11] D. Brauckhoff, X. Dimitropoulos, A. Wagner, and K. Salamatian, "Anomaly extraction in backbone networks using association rules," in Proceedings of the 9th ACM SIGCOMM conference on Internet measurement. ACM, 2009, pp. 28–34.

[12] L. Benetazzo, C. Narduzzi, P. A. Pegoraro, and R. Tittoto, "Passive measurement tool for monitoring mobile packet network performances," IEEE transactions on instrumentation and measurement, vol. 55, no. 2, 2006, pp. 449–455.

[13] K.-H. Kim, H. Nam, J.-H. Park, and H. Schulzrinne, "Mot: a collaborative network troubleshooting platform for the internet of things," in Wireless Communications and Networking Conference (WCNC), 2014 IEEE. IEEE, 2014, pp. 3438–3443.

[14] T. Qiu, Z. Ge, D. Pei, J. Wang, and J. Xu, "What happened in my network: mining network events from router syslogs," in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement. ACM, 2010, pp. 472–484.

[15] M. Vásquez-Bermúdez, J. Hidalgo, M. del Pilar Avilés-Vera, J. Sánchez-Cercado, and C. R. Antón-Cedeño, "Analysis of a network fault detection system to support decision making," in International Conference on Technologies and Innovation. Springer, 2017, pp. 72–83.

[16] S. Jamali and M. S. Garshasbi, "Fault localization algorithm in computer networks by employing a genetic algorithm," Journal of Experimental & Theoretical Artificial Intelligence, vol. 29, no. 1, 2017, pp. 157–174.

[17] E. S. Ali and M. Darwish, "Diagnosing network faults using bayesian and case-based reasoning techniques," in Computer Engineering & Systems, 2007. ICCES'07. International Conference on. IEEE, 2007, pp. 145–150.

[18] The Wireshark Wiki, "Displayfilters," [Online; accessed 20-April-2019]. [Online]. Available: https://wiki.wireshark.org/DisplayFilters

[19] C. Xu, H. Zhang, C. Huang, and D. Peng, "Study of fault diagnosis based on probabilistic neural network for turbine generator unit," in 2010 International Conference on Artificial Intelligence and Computational Intelligence, vol. 1. IEEE, 2010, pp. 275–279.

 ISBN: 978-1-61208-711-5