A protocol-independent container network observability analysis system based on eBPF

Chang Liu, Zhengong Cai
Polytechnic Institute, School of Software Technology
Zhejiang University
Hangzhou, P.R.China
{cl.lc, caizhengong}@zju.edu.cn

Bingshen Wang, Zhimin Tang, Jiaxu Liu
Container Service for Kubernetes (ACK)
Alibaba Cloud
Hangzhou, P.R.China
{bingshen.wbs, zhimin.tangzm, jiaxu.ljx}
@alibaba-inc.com

2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS) | 978-1-7281-9074-7/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICPADS51040.2020.00099

***Abstract*—Technologies such as microservices, containerization and Kubernetes in cloud-native environments make large-scale application delivery easier and easier, but problem troubleshooting and fault location in the face of massive applications is becoming more and more complex. Currently, the data collected by the mainstream monitoring technologies based on sampling is difficult to cover all anomalies, and the kernel's lack of observability also makes it difficult to monitor more detailed data in container environments such as the Kubernetes platform. In addition, most of the current technology solutions use tracing and application performance monitoring tools (APMs), but these technologies limit the language used by the application and need to be invasive into the application code, many scenarios require more general network performance detection diagnostic methods that do not invade the user application. In this paper, we propose to introduce network monitoring at the kernel level below the application for the Kubernetes cluster in Alibaba container service. By non-intrusive collection of user application L7/L4 layer network protocol interaction information based on eBPF, data collection of more than 10M throughputs per second can be achieved without modifying any kernel and application code, while the impact on the system application is less than 1%. It also uses machine learning methods to analyze and diagnose application network performance and problems, analyze network performance bottlenecks and locate specific instance information for different applications, and realize protocol-independent network performance problem location and analysis.**

Keywords-Performance Analysis; eBPF; Kubernetes; Cloud Network ; Machine Learning

I. INTRODUCTION

The current cloud-native environment of microservices, containerization, Kubernetes and other technologies to make large-scale distributed application delivery has become easier and easier, but then faced with a large number of applications, troubleshooting, fault location is also more and more complex[14][15][16]. Container-based platforms or systems such as Docker generally contain a series of tools and components, and container networks like ServiceMesh, DNS service discovery, virtual IP load balancing, independent of the host container IP, which greatly increases the complexity of application maintenance and network problem diagnosis.

Tools or methods for enhancing observability in distributed systems are based on three main aspects: distributed tracing, metrics, and logging[32][33]. There are many proven techniques and solutions for cloud-native environments such as Kubernetes. However, in the container environment, these traditional monitoring tools are difficult to meet today's needs, traditional monitoring tools can only monitor data exposed by the kernel, and then due to the virtualization of the container network and the kernel itself, the lack of observability makes it difficult to monitor more detailed data in the cloud platform, so cloud-native scenarios of fine-grained data monitoring and analysis in the traditional monitoring tools. It becomes very difficult to monitor and trace network packets. In addition, many network monitoring and tracing methods require packet sampling and collection, which is very costly for packet replication, parsing and context switching between kernel space and user space, and on the other hand, sampling-based monitoring and tracing methods cannot guarantee complete and accurate data. In addition, many current log analysis and distributed tracing methods require hacking into the system or modifying the application code, which makes operability and security worse in a large-scale distributed environment.

In the face of these challenges, this paper proposes a non-intrusive protocol-independent intelligent analytics system for container network observability based on modern kernel dynamic trace analysis mechanism eBPF. It is a non-intrusive, efficient and accurate, fine-grained end-to-end observability solution based on eBPF that enables transparent data collection for applications, fine-grained metrics such as nodes, applications, pods, containers, etc. in Kubernetes clusters, collects and aggregates data from applications with different network protocols and uses machine learning methods for anomaly detection of different network protocols . For different applications to analyze network performance bottlenecks and locate specific instance pods, which enables protocol-independent localization and analysis of network performance problems. The system solution proposed in this paper has the following features:

- **Fine-grained monitoring data**: to achieve very fine-grained container networks metrics of almost all the kernel operations based on eBPF.
- **Non-intrusive design**: no need to modify any kernel and application code, transparent to the user application.

2690-5965/20/$31.00 ©2020 IEEE
DOI 10.1109/ICPADS51040.2020.00099

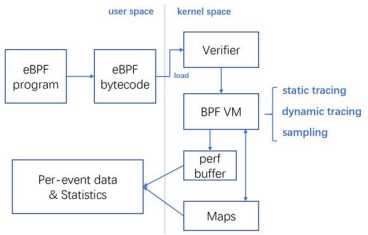


Fig. 1 eBPF mechanism in Linux

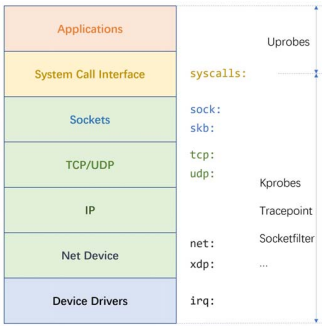


Fig. 2 Linux network stack and eBPF program

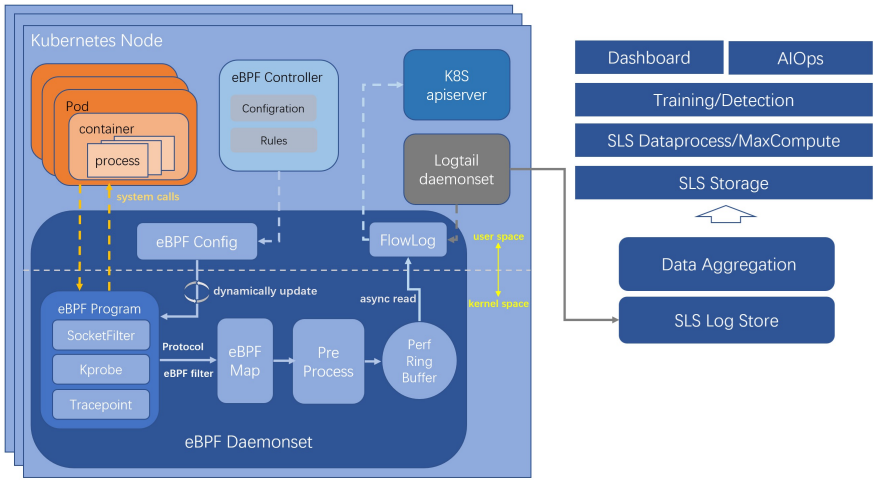


Fig. 3 Architecture of System

- **Scalability and low overhead**: the use of eBPF program can be dynamically updated to monitor the required events or system calls, while the system has low overhead.
- **Accurate and detailed performance analysis**: located the Pod instance and correlate the analysis of event-related context based on fine-grained data.

Another contribution of this article is based on this system to show the application scenarios in the Alibaba Cloud Container Service Kubernetes clusters. By analyzing the fine-grained time series container network data of the micro-service application in the Kubernetes clusters, anomaly detection is performed and the specific Pod instance where the network anomaly occurs and the context of the event occurrence are located.

The rest of this paper is organized as follows. Section II describes the system design. Section III and IV presents the evaluation and case study. Section V discusses the related work and Section VI concludes this paper.

II. DESIGN

This section will describe the design of this paper in terms of system architecture and analysis. The Figure shows how the architecture design and work. It is mainly divided into three parts: data collection, aggregate processing and intelligent analysis.

A. Critical data collection

In this paper, we use eBPF[1] technology to collect data from container networks in a cluster, which are completely transparent to the application without any modification to the application code. eBPF is a virtual machine that runs in the kernel and can convert almost any system call into an event, interacting with the kernel through a BPF system call, as shown in Fig.1, and can be implemented in the user state. The eBPF program can be executed on the occurrence of a specified system call, and the key information on the occurrence of that system call is passed back to the user state via a perf buffer or Maps. Currently in the Linux kernel eBPF allows writing static programs to be injected into access points such as tracepoint, kprobe, uprobe, etc., which can provide very strong scalability on static and dynamic traces. As shown in Fig. 2, compared to traditional monitoring on a multi-layer network stack, this mechanism brings a stronger observability, as each system call is based on thread-level data, which can be aggregated upwards to the container, pod or even service level. Those features enable non-intrusive and enhanced observability.

As shown in Fig. 3, The entire framework is running in the Kubernetes cluster, data collection module in the form of daemonset deployed in the Kubernetes cluster on each worker node, with eBPF Controller to set up configuration information and eBPF filtering rules. It is controlled by the Controller to update the eBPF Config information. The eBPF program is stored in eBPF Config, which supports JIT features and can be dynamically updated to the kernel. eBPF is the smallest running unit in a Kubernetes cluster, and the pod can contain multiple containers and processes. In this paper, three types of eBPF programs are used to implement container network data collection: socketfilter, kprobe/kretprobe and tracepoint. After the initial protocol matching and field filtering, the packets are stored in the eBPF Map and then pre-processed for initial aggregation according to

different protocols. To avoid frequent data copying from the kernel space, this paper outputs the aggregated flow information from the eBPF map to the user-state Flowlog via a perf buffer. The flowlog will call the interface in the kubernetes apiserver to further match the aggregated flow pod and other topological information, and finally the logtail daemonset collects flowlog to access the Alibaba SLS service logstore analysis.

B. *Data aggregation and analysis*

The network flow information collected in the cluster based on the eBPF program can be further aggregated into the data shown in the following Table 1. In addition to the metrics in Table 1, it can also collect network data in context and other basic information like `(pid, comm, ip_version, src_ip, src_port, dst_ip, dst_port, src_pod, src_ns, dst_pod, dst_ns)` based on Kubernetes apiserver.

The Linux kernel provides a number of trace points for injecting eBPF code, and for each protocol the kernel selects a trace point function, such as `tcp_v4_connect()` when an instance initiates a TCP connection, and dst_ns when an instance initiates a TCP connection. And the use of eBPF provides the helpler function `bpf_ktime_get_ns()` to record the nanosecond level time stamp. When the TCP connection state changes we can use `inet_sock_set_state()` to record before and after the change in state and time delay. UDP protocol-related events are tracked and collected similarly. For HTTP and DNS protocols, socket filter is used to analyze payloads and other content. Table 1 gives a description of each protocol metric and its details.

TABLE. 1 Protocol Metrics

Metric	Description
tcp_old_state	The old state of TCP connection
tcp_new_state	The new state of TCP connection
tcp_rx_bytes	The number of bytes received from client to server
tcp_tx_bytes	The number of bytes sent from client to server
tcp_span_time	Totol time of TCP connection state transferring
tcp_drop	Packet drop occurs during connection
tcp_tlp	The number of tail loss occurs
tcp_retransmit	The number of retransmission packets
tcp_rtt	TCP round trip time
udp_rx_bytes	The received bytes during UDP connection
udp_tx_bytes	The sent bytes during UDP connection
udp_duration	UDP connection duration time
http_request_method	The request method of HTTP request
http_request_uri	The request URI of HTTP request
http_request_version	The request version of HTTP request
http_status_code	HTTP connection response status code
http_duration	HTTP connection duration time
dns_transaction_id	The transaction id of DNS request
dns_query_domain	DNS request query domain
dns_query_type	DNS request query type
dns_response_code	The response code of DNS query
dns_duration	DNS request query duration time

The data analysis part is based on rules and manual labeling network anomaly data,then query the corresponding periodic data based on Aliyun SLS log aggregation, and further train relevant models for anomaly data to detect anomalies and locate specific pod locations and relevant contextual information according to different container network protocol timing data classification.

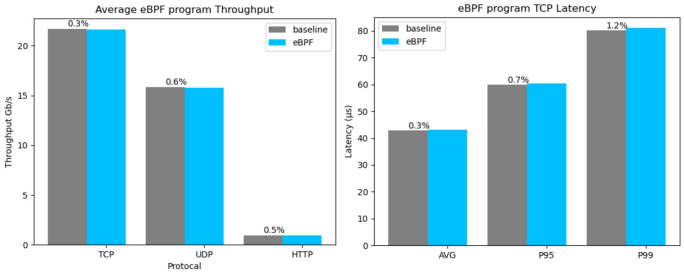
III. EVALUATION

A. *Overhead settings*

The test servers are all Aliyun ECS ecs.c6.xlarge (4vCPU 8GB I/O optimized) servers with Aliyun Linux2 64-bit operating system, kernel version 4.19, Docker version 18.09.2, and all test instances are running on Kubernetes. v1.16.6-aliyun.1 on the eBPF daemonset instance. When testing, we deploy ethr (version 1.0.6)[5] and sockperf on another regional server to generate multiple sets of test eBPF daemonset instances for network performance.

B. *Rresults analysis*

In order to measure the impact of eBPF procedures on the performance of container networks, the performance impact of multiple sets of eBPF procedures on the throughput and latency of different network protocols is tested for typical performance metrics of container networks. As shown in the Fig.4, the baseline is not open eBPF program case, the figure shows that, compared to the baseline eBPF program to open the overall performance of the system performance impact on the overall performance, TCP network protocol system throughput impact of 0.3%, UDP and HTTP network protocol performance overhead of 0.6% and 0.5% respectively, while the TCP average response delay impact the overall impact on system overhead is 0.3% and the 99th percentile response delay overhead is 1.2%.



(a) (b)

Fig. 4 eBPF Program Overhead

C. *Summary*

Based on eBPF, data collection that is transparent to the application is realized, and the impact on the overall system performance of the container network is about 1%. The Kubernetes cluster implements fine-grained indicators such as nodes, applications, pods, and containers, collects and aggregates data from different network protocols for anomaly detection, analyzes network performance bottle-

necks for different applications and locates specific instance pods, achieving protocol-independent network performance Problem location and analysis.

IV. Case Study

This section mainly describes the application of the system in Alibaba Cloud Container Service, performance analysis and diagnosis of microservices.

A. Experiment Settings

The test applications are modified microservices-demo of weaveworks[6], all the tested application instances are running on Kubernetes v1.16.6-aliyun.1. The Locust program (version 1.1) [8]is deployed on a Kubernetes cluster on another regional server to generate workload access application instances, and chaosmesh (v0.8.1)[7] is used in the cluster to simulate injection network failure during the test.

B. Design

In this paper, as shown in the Fig.5, we use chaos testing tool chaosmesh to simulate container network failure. Chaosmesh on kubenetes platform uses chaos operator to configure the network error or failure. The workload generator is used to generate workloads based on the eBPF scheme, which is used to collect data and train the access to Alibaba Cloud log storage SLS instances after data processing and aggregation. The experimental test application uses workload generator deployed in other regions to generate workload. eBPF-based scheme completes data collection and access to Alibaba Cloud log storage SLS instances, after data processing and aggregation to train the model and container network anomaly detection.

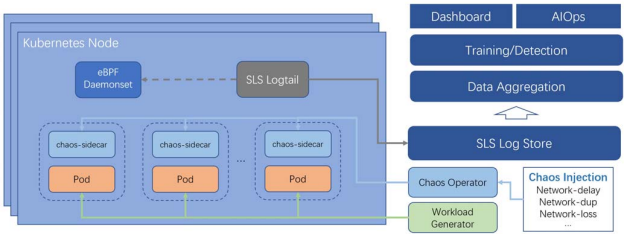


Fig. 5 Experiments Design

C. Result analysis

The experimental analysis scenario is based on the periodic data queried by Aliyun SLS aggregation for further training, and can complete anomaly detection and locate the specific pod location and related context information for different container network protocol timing data. As shown in the Fig.6, the front-end and orders connections of microservice applications are detected to drop at three points in time A, B and C, and anomalies in the retransmission number of carts are detected. The number of retransmissions for both front-end and order applications has increased significantly. Accordingly, the system can determine that A and B are caused by a significant delay in the front-end application, and C is caused by a packet loss anomaly in the carts application, resulting in network failure. As shown in Fig.7, through the fine-grained monitoring data of the HTTP protocol, the system can analyze the specific request path of the requesting front-end exception such as /detail/808a2de1, etc., and thus can locate the specific application and code. In addition, the system at C also locates the specific request path that causes severe network timeout problems due to packet loss in the carts application, i.e. URI /cart.

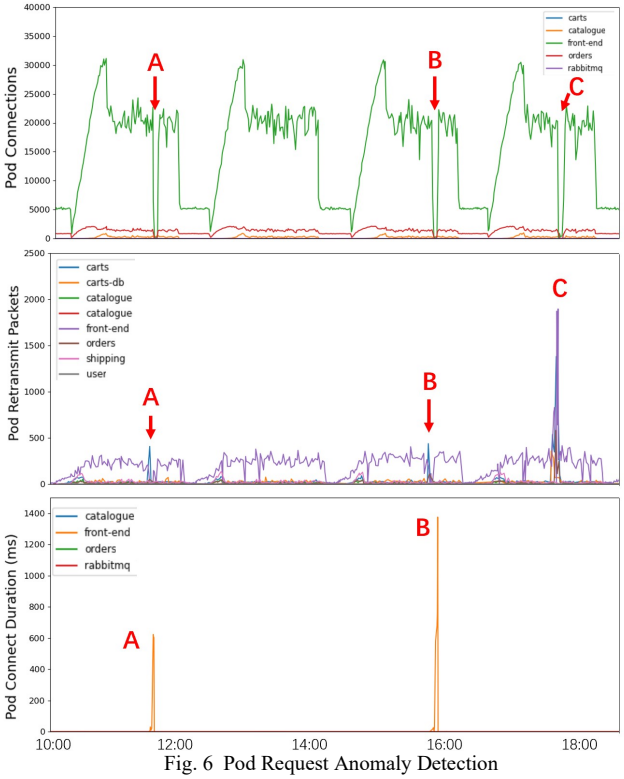


Fig. 6 Pod Request Anomaly Detection

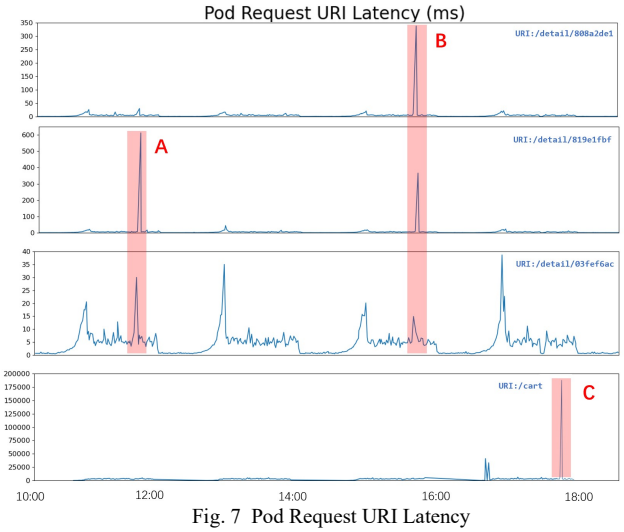


Fig. 7 Pod Request URI Latency

V. Related Work

A. System monitoring and data acquisition

There are many tools or monitoring systems that provide non-intrusive system monitoring data collection and log analysis capabilities, and these log or indicator-based solutions also provide the ability to analyze performance and diagnose problems[33][34][35]. However, these tools cannot provide granular data in today's mainstream cloud platforms, and many of the system performance metrics do not truly reflect the bottlenecks and issues in system performance. There are also a number of tools that provide some granular data at the kernel level[2][3][4], such as Strace[1] which provides a performance analysis and diagnostic Linux tool, Ftrace[3] also supports Linux kprobe, uprobe, tracpoints, etc., but these tools mainly provide tools for analysis or experimentation, extensibility and It is less programmable and, in addition, does not better support cloud platforms such as Kubernetes. For example, Dtrace[2] provides dynamically scalable, programmable eBPF-based system performance analysis tools for BSD systems, and Sysdig[9] provides a tool for monitoring system calls and interacting with user space information, which now also supports eBPF to monitor and analyze system performance. vNetTracer[21] proposes an eBPF-based distributed traceability solution, but these eBPF-based solutions do not natively support Docker, Kubernetes and other environments, and also do not provide a complete and fine-grained monitoring solution for container network environments, nor do they support performance analysis and anomaly detection methods.

B. Performance analysis and anomaly detection

Observability and performance analysis in massively distributed systems is composed of metrics, logging and tracing, but traditional observability tools face more and more bottlenecks in complex container environments[15][16][17][18]. Traditional monitoring tools such as cAdvisor[10], Atop[11] can only monitor the data exposed by the kernel, the kernel itself is not observable enough to monitor the container environment makes it difficult to monitor more detailed data, the container environment of fine-grained data monitoring and analysis in the traditional monitoring tools becomes very difficult.

There is also a lot of work based on machine learning or deep learning to provide performance analysis, anomaly detection and root cause localization, such as Seer[12] complete the root cause localization of SLA violation cases based on distributed tracking and deep learning, Pythia[13] analysis root cause of distributed systems based on OSProfile, etc., but these solutions are not suitable for cloud-native environment. In addition, due to the lack of fine-grained data, it is difficult to locate the specific instance of an application problem and the context of the process in which it occurs. MicroRCA[14] is based on the Kubernetes platform and Istio for data collection and root cause analysis, but these solutions are expensive for the system and do not work well in different application environments.

VI. Conclusion

This paper proposes a non-intrusive container network observability analysis system based on eBPF. This system collects user application interaction information of L7/L4 layer container network protocols in cloud-native environment non-intrusively. The system does not need to modify the kernel and application code to achieve data collection of more than 10M throughput per second, while the impact on system applications does not exceed 1%. Secondly, the machine learning method is used to analyze and diagnose the performance and problems of the application network, analyze the network performance bottleneck for different applications and locate the specific instance pod, and realize the location and analysis of protocol-independent network performance problems.

Acknowledgment

This work was supported by National Key R&D Program of China(No.2019YFB1600700) and Alibaba-Zhejiang University Joint Institute of Frontier Technologies.

References

[1] IO Visor. https://www.iovisor.org/.

[2] DTrace. https://dtrace.org/.

[3] Ftrace. http://elinux.org/Ftrace.

[4] SystemTap. https://sourceware.org/systemtap/.

[5] Ethr. https://github.com/microsoft/ethr.

[6] Microservices-demo.https://github.com/microservices-demo/microservices-demo.

[7] Chaos-mesh. https://github.com/chaos-mesh/chaos-mesh.

[8] Locust. https://locust.io/.

[9] Sysdig. https://sysdig.com/

[10] cAdvisor. https://github.com/google/cadvisor

[11] Atop. https://www.atoptool.nl/

[12] Gan, Yu et al. "Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices." Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (2019): n. pag.

[13] Ates, Emre et al. "An automated, cross-layer instrumentation framework for diagnosing performance problems in distributed applications." Proceedings of the ACM Symposium on Cloud Computing (2019): n. pag.

[14] Wu, Linlin et al. "MicroRCA: Root Cause Localization of Performance Issues in Microservices." NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium (2020): 1-9.

[15] Sigelman, Benjamin H. et al. "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure." (2010).

[16] Mace, Jonathan and Rodrigo Fonseca. "Universal context propagation for distributed system instrumentation." EuroSys '18 (2018).

[17] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat. Wap5: black-box performance debugging for wide-area systems. In Proceedings of WWW, 2006.

[18] R. R. Sambasivan, A. X. Zheng, M. De Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing

performance changes by comparing request flows. In Proceedings of USENIX NSDI, 2011.

[19] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch. The mystery machine: End-to-end performance analysis of large-scale internet services. In Proceedings of USENIX OSDI, 2014.

[20] K. Nagaraj, C. Killian, and J. Neville. Structured comparative analysis of systems logs to diagnose performance problems. In Proceedings of USENIX NSDI, 2012.

[21] Suo, Kun et al. "vNetTracer: Efficient and Programmable Packet Tracing in Virtualized Networks." 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS) (2018): 165-175.

[22] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and IonStoica. 2007. X-Trace: A Pervasive Network Tracing Framework. In Proceedings of the 4th USENIX Conference on Networked Systems Design & Imp lementation (NSDI'07). USENIX Association, USA, 20.

[23] F. C. Eigler and R. Hat. Problem solving with systemtap. In Proc. Of the Ottawa Linu K. Suo, J. Rao, L. Cheng, and F. C. M. Lau. Time capsule: Tracing

[24] packet latency across different layers in virtualized systems. In Proceedings of the 7th ACM APSys, 2016. x Symposium, 2006.

[25] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting

[26] large-scale system problems by mining console logs. In Proceedings of ACM SOSP, 2009.

[27] X. Zhao, Y. Zhang, D. Lion, M. F. Ullah, Y. Luo, D. Yuan, and M. Stumm. lprof: A non-intrusive request flow profiler for distributed systems. In Proceedings of USENIX OSDI, 2014.

[28] Luca Deri. [n. d.]. Monitoring Containerised Application Environments with eBPF. ([n. d.]).

[29] Luca Deri, Samuele Sabella, and Simone Mainardi. 2019. Combining System Visibility and Security Using eBPF. In Proceedings of the Third Italian Conference on Cyber Security, Pisa, Italy, February 13-15, 2019 (CEUR Workshop Proceedings), Pierpaolo Degano and Roberto Zunino (Eds.), Vol. 2315.

[30] Hao Zhou, Ming Chen, Qian Lin, Yong Wang, Xiaobin She, Sifan Liu,Rui Gu, Beng Chin Ooi, and Junfeng Yang. 2018. Overload Control for Scaling WeChat Microservices. Proceedings of the ACM Symp osium on Cloud Comp uting - SoCC '18 (2018).

[31] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In Proceedings of the Twenty-Fourth International Conference on Architectural Sup p ort for Programming Languages and Op erating Systems (ASPLOS '19). Association for Computing Machinery, New York,NY, USA, 107–120.

[32] Santana, Matheus et al. "Transparent tracing of microservice-based applications." Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (2019): n. pag.

[33] Ardelean, D. et al. "Performance Analysis of Cloud Applications." NSDI (2018).

[34] Vieira, M. et al. "Fast Packet Processing with eBPF and XDP." ACM Computing Surveys (CSUR) 53 (2020): 1 - 36.

[35] Deri, L. et al. "Combining System Visibility and Security Using eBPF." ITASEC (2019).