ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20230207000330
Original scientific paper

An Approach for Fast Fault Detection in Virtual Network

Yan WANG*, Ruiming FANG

Abstract: The diversity of applications in cloud computing and the dynamic nature of environment deployment makes virtual machines, containers, and distributed software systems to often have various software failures, which make it impossible to provide external services normally. Whether it is cloud management or distributed application itself, it takes a few seconds to find the fault of protocol class detection methods on the management or control surfaces of distributed applications, hundreds of milliseconds to find the fault of protocol class detection methods based on user interfaces, and the main time from the failure to recovery of distributed software systems is spent in detecting the fault. Therefore, timely discovery of faults (virtual machines, containers, software) is the key to subsequent fault diagnosis, isolation and recovery. Considering the network connection of virtual machines/containers in cloud infrastructure, more and more intelligent virtual network cards are used to connect virtual network elements (Virtual Router or Virtual Switch). This paper studies a fault detection mechanism of virtual machines, containers and distributed software based on the message driven mode of virtual network elements. Taking advantage of the VIRTIO message queue memory sharing feature between the front-end and back-end in the virtual network card of the virtualization network element and the virtual machine or container it detects in the same server in the cloud network, when the virtualization network element sends packets to the virtual machine or container, quickly check whether the message on the queue header of the previously sent VIRTIO message has been received and processed. If it has not been received and processed beyond a certain time threshold, it indicates that the virtual machine, the container and distributed software have failed. The method in this paper can significantly improve the fault detection performance of virtual machine/container/distributed application (from the second pole to the millisecond level) for a large number of business message scenarios, and provide faster fault detection for the rapid convergence of virtual network traffic, migration of computing nodes, and high availability of distributed applications.

Keywords: cloud computing; fast perception; fault detection; virtual network

1 INTRODUCTION

Cloud computing resources are provided to users in the form of on-demand services by using virtual machines or containers [1]. A large number of distributed applications hosted in virtual machines or containers usually rely on cloud computing platforms to deploy distributed software systems to provide online services [2].

However, due to the diversity of applications and the dynamic deployment environment, various software failures (e.g. virtual machines or containers, operating system kernels, network cards and user plane drivers, application message sending and receiving) often occur in virtual machines, containers and distributed software systems, which make it impossible to affect the normal operation of external services [3]. Tellme Networks reports that 75% of the failure recovery time of distributed software systems is used for fault detection and 18% for fault diagnosis [4]. Early fault detection can mitigate or prevent 65% of failures [5]. Therefore, timely fault detection (virtual machines, containers, software) is the key to subsequent fault diagnosis, protection, isolation and recovery.

For virtual machine and container faults, the connectivity heartbeat of cloud management can be used for fault discovery [6, 7]. For distributed applications, the application probe of cloud management can be used for fault discovery [8, 9] or the distributed application can implement its own fault detection mechanism or among different distributed applications [10, 11]. Whether it is cloud management or distributed application itself, it usually takes several seconds to discover the fault of the protocol detection method of the management plane or control plane of the distributed application [12] (for example, PING or HTTPS, at least three consecutive times per 1 s), and hundreds of milliseconds to discover the fault of the protocol detection method based on the user plane [13] (for example, BFD, at least three consecutive times per 100 ms). If you try to reduce the packet interval to find faults more quickly, you will be constrained to support more detection objects due to limited computing and processing capacity when the control plane or user plane detection protocol is a computing resource sharing process in the system.

Therefore, it is a challenge for the current cloud computing fault detection mechanism to avoid the complexity of fault detection of distributed applications themselves, achieve faster fault detection (from seconds to tens of milliseconds) of virtual machines, containers and distributed software with fewer computing resources, and simultaneously meet a large number of detection objects deployed in the server (hundreds of virtual machines, containers and distributed applications).

Considering that cloud infrastructure is built by interconnecting large-scale virtualized data centers [14], with the maturity of server smart network cards, virtual machines/containers in the data center are more and more connected by virtual network cards (vNICs) to distributed virtualized network elements as DVR (Distributed Virtual Routers) or DVS (Distributed Virtual Switches) [15-19], and the functions of user and control surfaces on the cpu of the virtualized network elements are uninstalled from the smart network cards [16-18, 26-29].

This paper studies a fault detection mechanism for virtual machines, containers and distributed software based on the message driven mode of virtualized network elements. The main contribution is to propose solutions to the challenges faced by the above cloud computing fault detection mechanism. The main idea is to use the virtio message queue memory sharing feature between the front end and back end of the virtualized network element in the same server in the cloud network and the vNIC network card of the virtual machine or container it detects. The virtualized network element no longer detects virtual machine, container and distributed software failures through heartbeat messages, but quickly detects failures through software. It is found that when the virtualized network element sends packets to the virtual machine or

container, quickly check whether the message on the queue header of the previously sent virtio message has been received and processed by the virtual machine or container. If the message has not been received and processed beyond a certain time threshold, it indicates that the virtual machine, container and distributed software have failed.

2 RESEARCH METHOD

2.1 Technical Components

2.1.1 Vhostuse

As shown in Fig. 1, the path of a complete packet from the virtual machine to the physical machine is: virtual machine-virtual network card-virtualization layer-kernel bridge-physical network card. The fully virtualized network card will cause more IO and MMIO operations every time it sends and receives packets in the virtual machine, which makes the virtual machine trap frequently, and finally leads to poor network performance.

In order to solve the above performance problems of full virtualization, the industry has proposed virtio technology. The virtio driver in the virtual machine and the back-end driver vhostuser on the host simply use the virtqueue shared queue to exchange data [19]. The path of data packets from the virtual machine to the physical machine is simplified as: virtual machine-virtual network card-kernel network bridge-physical network card. The front-end virtio driver is used in VM/C, and the back-end vhostuser driver is used in the user state bridge. The vhostuser (user mode driver) combined with front-end virtio driver eliminates the complex IO operation during virtual network card simulation, and can greatly improve the performance of virtual network.

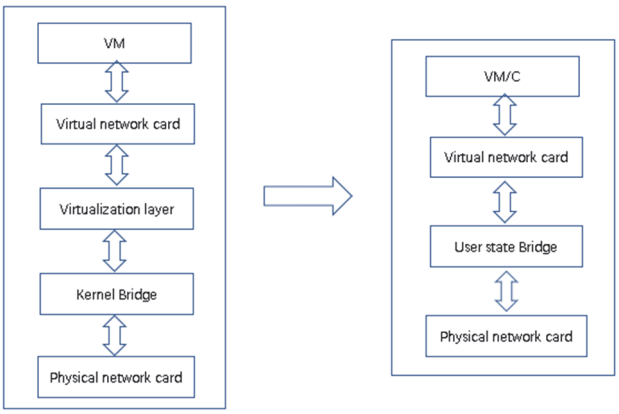

Figure 1 Data path evolution in virtual network

2.1.2 Virtqueue

As shown in Fig. 2, the virtual queue is the actual data link connecting the virtio front-end driver and the host vhostuser back-end driver in the guest operating system [20]. The network device has two virtqueues, one for sending data packets and one for receiving data packets.

Each virtual queue is mainly composed of three parts:

- Descriptor table is used to store some associated descriptors. Each descriptor is a description of buffer, including an address/length pair.
- Available rings (available rings) are used on the guest side to indicate which descriptor chains are currently available.
- The used ring (used ring) is used to indicate that the host side indicates that those descriptors have been used.

The descriptor list points to the actual data to be transmitted, and the two ring tables point to the descriptor list, marking the processing progress of the descriptors in the descriptor list by the front-end and back-end drivers. When virtqueue is used to receive messages, the front end adds the blank descriptor block to the available ring and provides it to the back end to receive messages. After receiving messages, the back end will put them into the used ring. When virtqueue is used to send messages, the front-end driver adds the message descriptor to be sent to the available ring for processing by the back-end. After processing, the back-end will put it into the used ring, and the front-end will release the message descriptor and reload it into the available ring.

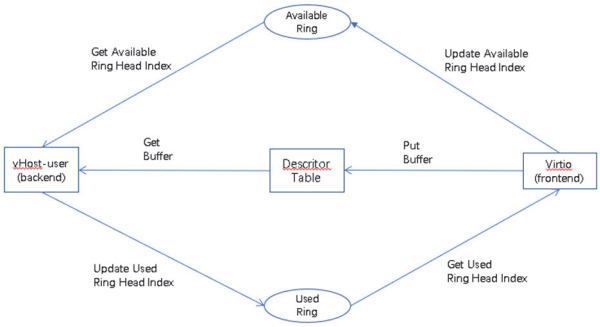

Figure 2 Composition of virtqueue virtual queue

2.2 TECHNICAL SCHEME

This paper studies a fault detection method based on virtqueue virtual queue state, which meets the fault detection of virtual queue back-end driver to front-end driver. The vhostuser backend first judges the virtqueue virtual queue front-end driver failure by checking whether there is no change in the available ring head index position during continuous multiple message processing, and then further associates the front and rear end object states through the virtqueue virtual queue driver state.

2.2.1 Step 1: Identify Virtqueue Virtual Queue Front end Fault Status

As shown in Fig. 3, the back-end vhostuser driver can identify the front-end fault process of virtqueue virtual queue:

(1) The front-end virtio driver uses the used ring element corresponding to the back-end received message to index to the desc descriptor to obtain buffer data.

(2) The front end virtio then releases the message buffer, adds the new blank message buffer to the descriptor table, adds the available ring element pointing to the descriptor, and updates the available ring head index position.

(3) The vhostuser driver receives the message from the outside, obtains the current available ring head index position, uses a certain number of available ring elements not exceeding this position, fills the buffer data and stores the buffer index in DESC, and further stores the desc-index in the new used ring element, that is, the used ring element to be read by the front end in (1).

(4) The vHostuser driver can be selected according to the granularity of virtual machine, container or distributed application. Each time the message is processed, judge whether the index position of the available ring head changes. It is positive under normal circumstances, as the back end consumes the available ring element, the front end will update the available ring head index position in time to supplement the available ring element. Therefore, if the vhostuser driver continuously receives and processes n messages, and the available ring head index position remains unchanged, the vhostuser driver can judge the virtqueue virtual queue front end failure.

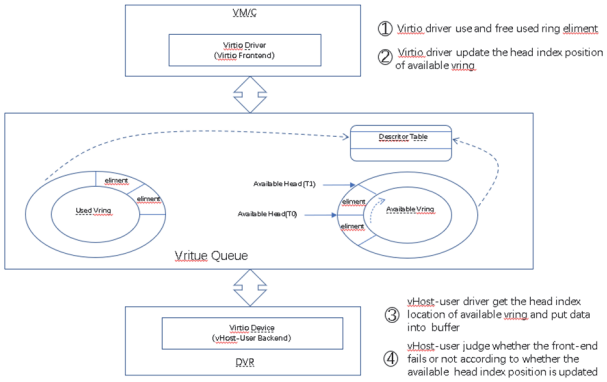

Figure 3 Virtqueue virtual queue front end fault detection method

2.2.2 Step 2: Associated Front-end VM / C Fault Status

As shown in Fig. 4, the front-end fault model causing virtqueue read processing packet failure is as follows:

(1) In case of APP packet processing failure in the virtual or container, the app will no longer receive messages from the protocol stack or dpdk based on the virtio driver, the protocol stack or dpdk will not receive messages from the virtio driver, and the virtio driver will no longer read processing packets from the virtqueue and update the available ring.

(2) When virtio driver fails or resets, virtio driver in virtual or container will no longer read processing packets from virtqueue and update available ring.

(3) If the guestos fails or resets, the virtio driver installed on the guestos in the virtual machine will no longer read the processing package from the virtqueue and update the available ring.

(4) When the qemu fails or resets, the virtio driver in the virtual machine will no longer read processing packets from the virtqueue and update the available ring.

It can be seen from the above that by identifying the front-end fault status of virtqueue virtual queue, the effect is the same as that of the back-end, and the heartbeat is used to distinguish the front-end

If there is a fault, you can get whether the receiving and contracting path of the VM/C front end is faulty, and then determine whether the front end is faulty.

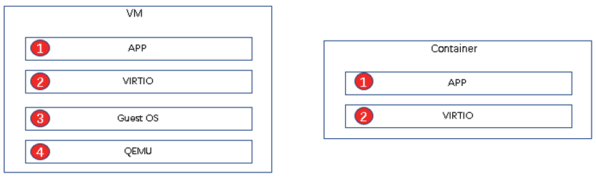

Figure 4 Front end VM / C fault model

2.2.3 Step 3: Port Status of Associated Backend Virtio Devices

As shown in Fig. 5, the back-end vhostuser driver further links the virtio device port status down on the DVR by judging the front-end failure of the virtqueue virtual queue, so as to trigger the traffic to switch to other virtio device ports of the local DVR or the virtio device port of the remote DVR.

(1) Vhostuser detects a virtio queue front-end failure and links the virtio device port down.

(2) DVR routing convergence switches the traffic to the virtio device port of other ups.

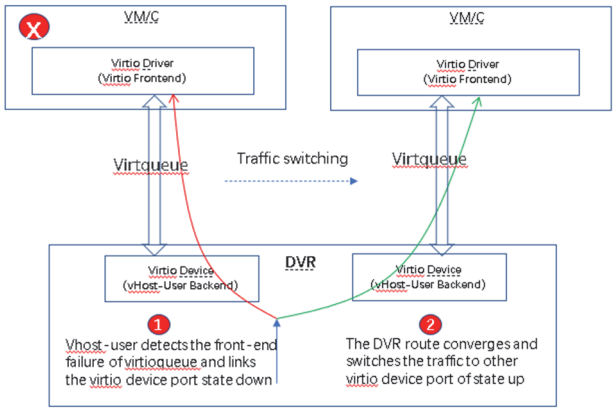

Figure 5 Back end virtio device port traffic switching

2.3 Establishment of Verification Environment

As shown in Fig. 6, build the above traffic convergence performance test topology:

(1) A bare metal DVR is deployed on host1 for fault detection and traffic switching of VM/C. The bare metal DVR is extended by the vrouter software of the open-source tungsten fabric project [21, 24, 25] which support distributed routing and switching units. Two sets of VM/C are installed on host1 at the same time, and its applications are simulated by the vrouter software deployed in two VM/C with same service ip.

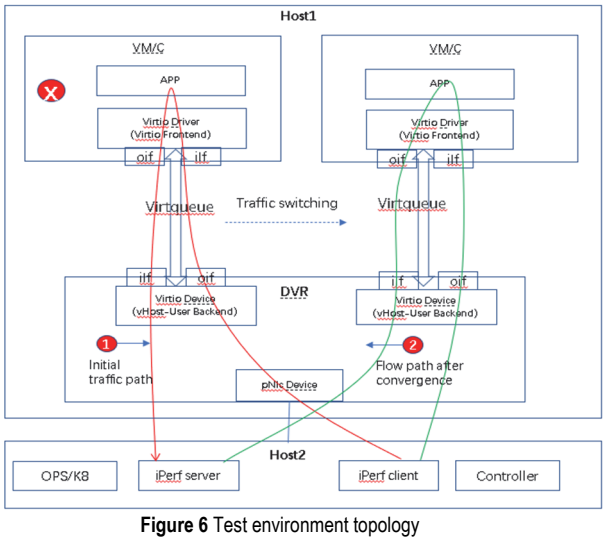

Figure 6 Test environment topology

(2) Iperf server and iperf client are installed on host2 to test streaming with destination of service ip. OPS/K8 management system is installed on host2 to deploy VM/C.

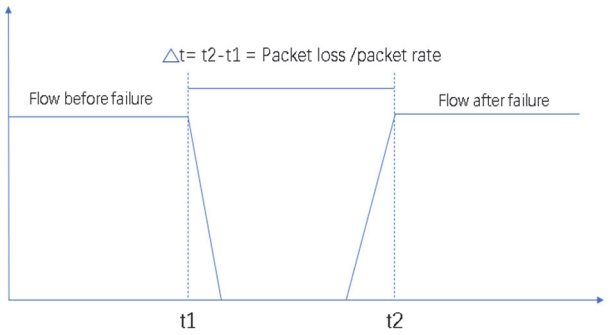

Figure 7 Flow convergence performance test method

The end-to-end flow convergence performance test methods for different fault detection methods are as follows:

(1) DVR initially sets high priority to the upper left VM/C static route and low priority to the upper right VM/C static route, so as to meet the initial traffic to the upper left VM/C. When the upper left VM/C fails, DVR switches the traffic to the upper VM/C.

(2) During the performance test, the iperf client on host2 sends constant speed traffic, initially reaches the VM/C in port on the left in the figure through the DVR out port, and then reaches the DVR in port from the VM/C out port. The DVR further forwards the message to the iperf server on host2.

(3) The front end deploys virtual machines and containers in turn, sets different front-end fault detection methods in turn on the DVR, and records the end-to-end traffic convergence performance data respectively. As shown in Fig. 7, the packet loss time of traffic switching can be calculated by dividing the packet loss number by the packet rate [22, 23].

3 RESULTS

As shown in Tab. 1, set the parameters of different fault detection methods required for performance test [14]:

Table 1 Fault detection method parameters

	virtqueue	ping	bfd
Packet interval / ms	1/pps	1000	200
Max. times	5	5	5

Flow from iperf client to iperf server at different flow rates (4 KB UDP). Conduct flow convergence test in case of VM/C failure for different detection methods, and the flow convergence results can be obtained as shown in Tab. 2 and Fig. 8.

Table 2 Flow convergence performance data (udp)

	Packet rate / pps	1	10	100	1000	10000
Convergence / ms	virtqueue	5098	587	141	96	89
	ping	5099	5088	5084	5092	5090
	bfd	1089	1090	1095	1084	1092

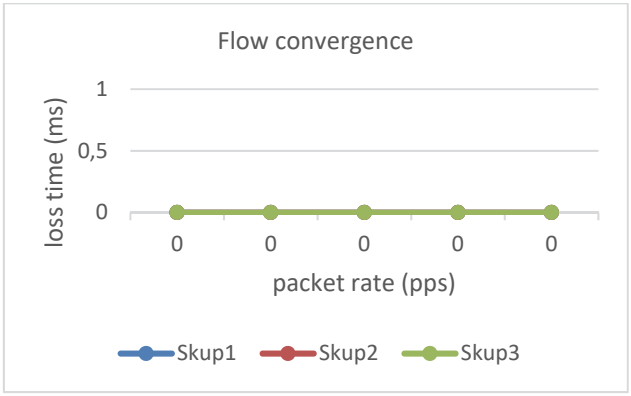

Figure 8 Flow convergence performance test results (udp)

Flow from iperf client to iperf server at different flow rates (4 KB TCP). Conduct flow convergence test in case of VM/C failure for different detection methods, and the flow convergence results can be obtained as shown in Tab. 3 and Fig. 9.

Table 3 Flow convergence performance data (tcp)

	Packet rate / pps	1	10	100	1000	10000
Convergence / ms	virtqueue	5088	592	135	92	94
	ping	5085	5069	5089	5072	5081
	bfd	1046	1058	1069	1092	1077

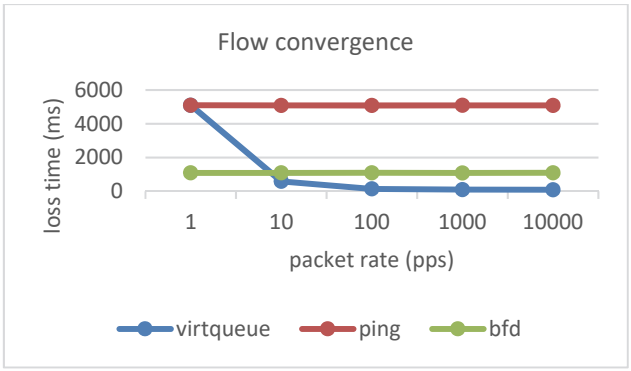

Figure 9 Flow convergence performance test results (tcp)

It can be seen from Tab. 2 and Tab. 3 and Fig. 9 that the front-end fault detection method based on virtqueue is significantly better than the BFD and Ping detection methods in terms of traffic switching convergence time when the contracting rate increases (more than 100 pps). The packet loss time mainly comes from the traffic switching convergence time of DVR (about 90 ms), the application of UDP and TCP stream reconstruction time is comparable and less than 1ms is negligible, and the fault detection time is the contracting interval of 5 packets (less than 50 ms). However, when the contracting rate decreases (less than 8 pps), the packet loss time will exceed the BFD detection method. When it further decreases (less than 1 pps), the packet loss time will exceed the Ping detection method. In extreme cases, when the contracting rate is zero, the virtqueue method cannot detect the front-end fault.

4 CONCLUSION

It can be concluded from the above research that the virtqueue front-end state detection method of the virtualization network element message driven mode studied in this paper takes into account the low computing

resource overhead and high fault detection performance. Compared to current fault detection methods of various heartbeat methods, this method can significantly improve the fault detection performance of virtual machines, containers, and distributed applications (from seconds to tens of milliseconds) for a large number of packet throughput scenarios without occupying additional computational resources.

It can also be concluded from the above research results that the virtqueue front-end fault detection method studied in this paper is mainly applicable to scenarios where the front-end packet receiving rate is high (more than 100 pps), which can quickly detect virtual machine/container faults within 50ms. It is not suitable for the kernel TAP port, and it is recommended to independently deploy VMs/containers for applications, typically container deployment scenarios. When multiple applications deploy VMs together, although the application failure can be detected through the flow classification detection by superimposing advanced ACL on the interface, at this time, the virtqueue front-end state detection based on each packet can quickly detect front-end faults. Low packet rate scenarios (less than 8 pps) are not suitable for longer detection time. At this time, the number of packets lost is very small, so it is not meaningful to perform fast detection. You can still use some detection methods based on slow heartbeat. Therefore, in the actual application scenario, the front-end status detection of virtqueue can be combined with some of the slow detection methods described above. When multiple detection methods coexist, the front-end is considered to have failed when any one of the back-end detection methods first detects the failure, which can provide faster fault discovery for the rapid convergence of traffic in the work scenario of a large number of packets of virtualized network services, migration of computing nodes, and high availability of distributed applications.

Acknowledgements

This work is supported by Fund Projects (Ec202301, Title: Research on impact dynamics of cam drive system of high-end printing equipment based on DIC measurement).

5 REFERENCES

[1] Barroso, L., Dean, J., & Holzle, U. (2003). Web search for a planet: The google cluster architecture. *IEEE Micro*, *23*(2), 22-28. https://doi.org/10.1109/MM.2003.1196112

[2] Tao, W., Fengchun, L., Liya, W., Yanfeng, J., & Lei, W. (2017). Research on fault detection technology of distributed software system based on statistical monitoring in cloud environment. *Journal of Computer Science*, (02), 397-413. https://doi.org/10.1109/ACCESS.2019.2962176

[3] Jiang, G., Haifeng, C., & Kenji, Y. (2006). Modelling and Tracking of Transaction Flow Dynamics for Fault Detection in Complex Systems. *IEEE transactions on dependable and secure computing*, (4). https://doi.org/10.1109/TDSC.2006.52

[4] Birke, R., Ioana, G., Lydia, Y. C., Dorothea, W., & Ton, E. (2014). Failure Analysis of Virtual and Physical Machines: Patterns, Causes and Characteristics. *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks,* 1-12. https://doi.org/10.1109/DSN.2014.18

[5] David, O., et.al. (2002). Why Do Internet Services Fail, and What Can Be Done About It? *Usenix symposium on internet technologies and systems*, 1-16.

[6] Openstack. High Availability for Virtual Machines.

[7] Kubernetes. High Availability (HA).

[8] Absa, S., Shajulin, B., & Anto Kumar, R. P. (2019). Monitoring IaaS using various cloud monitors.*Cluster Computing*, (5). https://doi.org/10.1007/s10586-017-1657-y

[9] Prasad, V. K. & Bhavsar, M. D. (2020). Monitoring IaaS Cloud for Healthcare Systems: Healthcare Information Management and Cloud Resources Utilization. *Int. J. E Health Medical Commun*, *11*, 54-70. https://doi.org/10.4018/IJEHMC.2020070104

[10] Federici, M., Carlo, G., & Bruno, L. M. (2014). HAVmS: Highly Available Virtual Machine Computer System Fault Tolerant with Automatic Failback and Close to Zero Downtime. *Computer Science.* https://doi.org/10.14311/APP.2014.01.0278

[11] Binh, N., Tian, Z., Bozidar, R., Ryan, S., Thomas, K., Jakub, K., & Van der Merwe, J. (2018). ECHO: A Reliable Distributed Cellular Core Network for Hyper-scale Public Clouds. *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18). Association for Computing Machinery*, 163-178. https://doi.org/10.1145/3241539.3241564

[12] Amazon. Health checks for your target groups.

[13] Amazon. How do I enable BFD for my Direct Connect?

[14] Ravi, J., Vincenzo, P., & Marco, S. (2013). Fault Tolerance Management in Cloud Computing: A System-Level Perspective. *IEEE Systems Journal*, *7*(2), 288-297. https://doi.org/10.1109/JSYST.2012.2221934

[15] Phillipa, G., Navendu, J., & Nachiappan, N. (2011). Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. *Computer Communication Review: A Quarterly Publication of the Special Interest Group on Data Communication*, *41*(4), 350-361. https://doi.org/10.1145/2043164.2018477

[16] Wang, Y., Xiong, W., Zhuobin, H., Wei, L., & Shizhong, X. (2013). Joint optimization of dynamic resource allocation and packet scheduling for virtual switches in cognitive internet of vehicles. *EURASIP Journal on Advances in Signal Processing*, *2022*(1). https://doi.org/10.1186/s13634-022-00862-7

[17] Thomas, B., Bruno, B., Guillaume, A. G., & Vincent, J. (2018). An Accurate and Efficient Modeling Framework for the Performance Evaluation of DPDK-Based Virtual Switches. *IEEE Trans. Network and Service Management*, *15*(4). https://doi.org/10.1109/TNSM.2018.2874476

[18] Junichi, N. & Norihiko, S. (2016). Efficient switch clustering for distributed controllers of OpenFlow network with bi-connectivity. *Computer Networks*, *96.* https://doi.org/10.1016/j.comnet.2015.10.017

[19] Openvswitch. DPDK vHost User Ports.Available:

[20] Oasis. Virtual I/O Device (VIRTIO) Version 1.0.

[21] Opencontrail. Contrail vRouter.

[22] Gurumurthy, S., et. al. (2022). Hybrid pigeon inspired optimizer-gray wolf optimization for network intrusion detection. *Journal of System And Management Sciences*, *12*(4), 383-397.

[23] Mohammad, B. R., Abdelsalam, A. A., & Ali, K. A. (2022). Wireless sensor network performance enhancement using software defined networking principle. *Journal of System and Management Sciences*, *12*(2), 221-235.

[24] Ahranjani, L., Saen, R., & Gholenji, I. (2021). Estimating capacity utilization in network dea in the presence of undesirable outputs. *Economic Computation and Economic Cybernetics Studies and Research*, *55*(4), 227-240. https://doi.org/10.24818/18423264/55.4.21.15

[25] Xu, W., Sun, H. Y., Awaga, A. L., Yan, Y., & Cui, Y. J. (2022). Optimization approaches for solving production

scheduling problem: a brief overview and a case study for hybrid flow shop using genetic algorithms. *Advances in Production Engineering & Management*, 17(1), 45-56, https://doi.org/10.14743/apem2022.1.420
[26] Andy, P., et.al. (2022). Nvidia tests: DPUs can cutpower needed by servers. *Network World (Online).*
[27] Yan, Y., Arash, F. B., Reza, N., & Dimitra, S. (2020). P4-enabled Smart NIC: Enabling Sliceable and Service-Driven Optical Data Centres. *Journal of Lightwave Technology*, *38*(9). https://doi.org/10.1109/JLT.2020.2966517
[28] Andy, P. et.al. (2019). Mellanox introduces SmartNICs to eliminate network load on CPUs. *Network World (Online).*
[29] Gao, P., Yang, X., & Chao, H. J. (2021). OVS-CAB: Efficient rule-caching for Open vSwitch hardware offloading. *Computer Networks*, *188*. https://doi.org/10.1016/j.comnet.2021.107844

Contact information:

Yan WANG, Associate Professor
(Corresponding author)
Beijing Institute of Graphic Communication,
No. 1, Xinghua Street (Section 2), Daxing District, Beijing, China
E-mail: wangyanzi@bigc.edu.cn

Ruiming FANG, Associate Professor
Beijing Institute of Graphic Communication,
No. 1, Xinghua Street (Section 2), Daxing District, Beijing, China
E-mail: frmluck@bigc.edu.cn.