Labeled Network Stack: A Co-designed Stack for Low Tail-Latency and High Concurrency in Datacenter Services

Wenli Zhang, Ke Liu, Hui Song, Lan Yu, Mingyu Chen

Labeled Network Stack: A Co-Designed Stack for Low Tail-Latency and High Concurrency in Datacenter Services

Wenli Zhang[1], Ke Liu, Hui Song, Lan Yu, and Minyu Chen[1,2]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Kexueyuan South Road 6, Haidian District, Beijing, P.R.China,
{zhangwl,liuke,songhui,yulan,cmy}@ict.ac.cn,
WWW home page: http://acs.ict.ac.cn/network/index-eng.html
[2] University of Chinese Academy of Sciences, Yuquan Road 19(A), Shijingshan District, Beijing, P.R.China

Abstract. Many Internet, mobile Internet, and IoT services require both low tail-latency and high concurrency in datacenters. The current protocol stack design pays more attention to throughput and average performance, considering little on tail latency and priority. We address this question by proposing a hardware-software co-designed Labeled Network Stack (LNS) for future datacenters. The key innovation is a payload labeling mechanism that distinguishes data packets in a TCP link across the full network stack, including the application, the TCP/IP and the Ethernet layer. This design enables prioritized data packets processing and forwarding along the full data path, to reduce the tail latency of critical requests. We built a prototype datacenter server to evaluate the LNS design against a standard Linux kernel stack and the mTCP research, using IoT kernel benchmark MCC. Experiment results show that the LNS design can provide an order of magnitude improvement on tail latency and concurrency.

Keywords: tail latency, high concurrent server, priority, label, network stack

1 Introduction

For the new generation of cloud computing server applications such as mobile Internet and IoT, with characteristics of high concurrency and low latency constraints, the behavior, motivation and access time of concurrent clients are all uncertain [1], so the unconscious resource competition from massive concurrent requests will lead to fluctuations in service latency. Google put forward the data center "Tail Latency" issue [2], usually measured with the 99^{th} percentile latency. UC Berkeley also highlighted the long tail problem in web applications [3]. One main goal of their FireBox project [4] is to cope with tail latency for 2020 cloud computing systems.

Long tail latency will seriously affect Quality of Service (QoS). So, to guarantee the tail latency not exaggerated too much, the utilization rate of data center resources for online services is usually not very high, i.e., the CPU utilization rate is generally <30%. Besides, the long TCP connection has become the mainstream for cloud client/server communication now, i.e. MQTT. Long connection is used to support many mobile clients, for the server to locate the client (i.e. many IoT devices only have internal IPs) and to reduce the overhead of multiple authentications, however, it would bring about high concurrency problem, and make the tail latency more serious, because of long-term resource occupation.

How to control tail latency has become an important direction for cloud computing and big data research in recent years. User-mode TCP/IP and NIC offload are mainly involved recently. While Jialin Li et al. [7] found that application, hardware and operating system may all cause tail-delayed response. However, many studies overlook one important factor. Different requests have different delay requirements, while the current works improve delay commonly without differentiate the requests. Traditional layered network stacks only provide priority in the flow granularity coarsely at some layers, and lack of finer grained priority control mechanisms.

Therefore, we proposed a label-based network stack, using payload labeling and codesign across layers to support full-path data-sensing and prioritization. Then, we design and implement a prototype. Test results showed that the LNS got an order of magnitude improvement on tail latency and concurrency over the mainstream systems. Our two main contributions are as follows.

1) **Labeled Network Stack (LNS)** to achieve full-data-path QoS guarantee. The LNS is to support distinguishing, isolation and prioritizing in packet granularity across the full data path through payload labeling. It is different from the traditional flow level control method that only based on predefined protocol header. So, the LNS can do more efficiently to get both high concurrency and low tail latency.

2) **Prototype of LNS** to show an order of magnitude improvement on tail latency and concurrency over the mainstream. Based on LNS idea and standard X86 Linux server, we did hardware-software co-design on NIC, TCP/IP stack, server framework layers and formed the first testbed for LNS. Tests show that significant improvement. Besides IoT and mobile Microservices, our server fits into application with features on long connection, high concurrency and user experience requirement widely.

2 Labeled Network Stack

We first discuss the motivation on LNS, then address our design.

2.1 Why LNS?

Some requests in the data flow are naturally of priority requirement for many cloud services. For example, order operation should be more sensitive on latency than browsing in electronic commerce. However, traditional method lacks distinguishing, isolation and prioritizing abilities for packets. This inevitably causes unnecessary waiting delay for critical services, especially in the case of resource reuse contention.

Without a doubt, there are many labels in traditional networks, such as ECN congestion label and DSCP differentiated service field, but no label works through the whole packet processing after the header peeled off. So, when without priority policy, data packet processing would use system resources in unordered state, which inevitably leads to high entropy [5]. While the current mainstream NICs, TCP/IP stacks, and server frameworks all do not support priority for the full data path.

Then, we put forward the idea to prioritize some requests in network flow according to application features and do hardware-software co-design optimization in full stack, which finally form the LNS.

2.2 LNS Idea

The main idea is that 1) **payload labeling mechanism**. When to send a packet, the sever framework will attach a label in front of the payload according to application requirement. For example, to control overhead 1 bit is added as priority in our experiment, where 0x1 stands for high priority and 0x0 stands for the low. When the encrypted packet arrives at the receiver, it should be decrypted before protocol analysis, and put to the right priority queue by label identifying. Finally, it accesses database with labeled RPC. For simplicity, the priority in this work is randomly labeled by programmers in flow generator, while later the label can be attached based on application characteristics automatically; 2) **multi-path priority in packet**, including queue partition in the intelligent NIC, multi-queue zero-copy driver, custom user-mode TCP/IP stack and other layers. It provides a mechanism to avoid frequent blockages caused by a single queue and prioritize in packet granularity. 3) **tail latency QoS scheduling**. As shown in Fig. 1, through payload labeling and priority scheduling in all layers, it can reduce the latency of critical operations more accurately and improve both overall service efficiency and user experience. In LNS we coordinate well with the design idea on distinguishing, isolate and prioritizing. To focus, we only research tail latency in single node in this paper.

3 Implementation

Based on the LNS idea, we developed a prototype to achieve label identification and scheduling across process stages, including customized NIC Sando, mTCP-based user-mode protocol stack and epoll-based event driven server framework with priority enhancement.

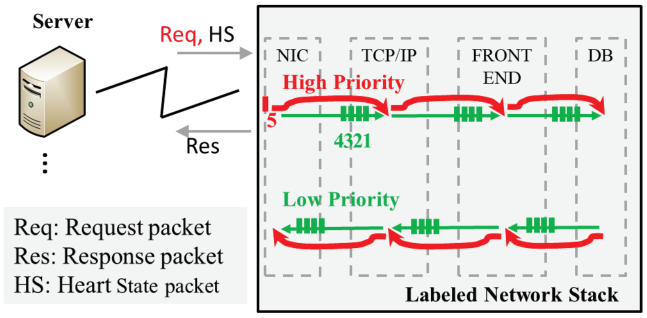

Fig. 1. Main idea of LNS. Its features include full-path payload labeling priority in packet granularity and high priority taking precedence. Assumed the 5th arriving packet is of high priority, it can be served first before the other four in queue

4 Evaluation

We answer a question in this section that: can we have order of magnitude improvement on tail latency and concurrency over the mainstream? In Section 3.2, experiment results show that the LNS design can provide that high performance.

4.1 Experiment setup

We used mainstream X86 servers to build the test system. It uses the flow generator MCC [8] to simulate long connections of massive IoT devices with sever, and the load balancer distributes the data to 4 servers for related processing (limited by the amount of Sando card). The monitor[8] adopts the full-traffic accurate measurement to get server-side latency in nanosecond (once timing itself costs about 2.7 nanoseconds by executing clock_gettime). To facilitate the distinction, when the LNS is running on the process node, it is briefly as LNS with Sando for short. A target system is abbreviated as e1000-Linux, which runs the X86 standard hardware and software, then alternative the kernel stack as the user-level protocol stack mTCP [6] to form the second target system shortly as e1000-mTCP.

4.2 High concurrency and low tail latency

As shown in Fig. 2, while ranging the 99^{th} percentile latency from 1 to 60 milliseconds, the LNS with Sando system can reach up 10 million connections, which increases an order of magnitude compared with e1000-Linux system, and at least 1 times more than e1000-mTCP, with 4 nodes in the test system.

5 Conclusion

This paper proposes a Labeled Network Stack (LNS) technique for future datacenters, to provide order of magnitude improvement on tail latency and con-

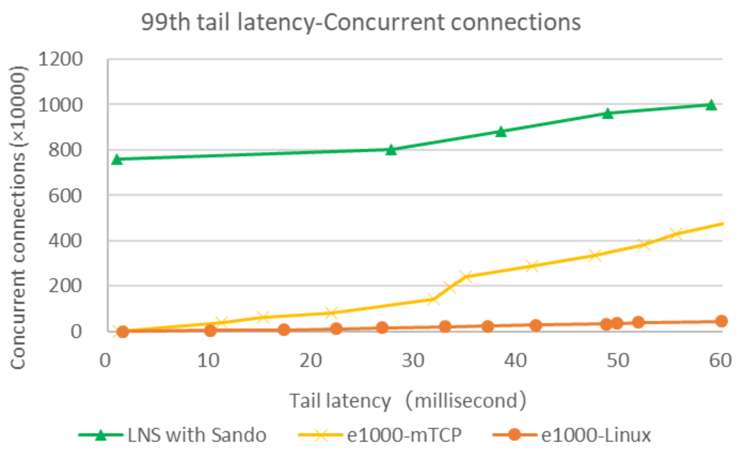

Fig. 2. 99-percentile tail latency - Concurrent connection

currency. Its main innovation is a labeling mechanism that labels the payload (packet body) rather than the packet header. Such payload labeling enables distinguishing different payload requests in the same flow across the full network stack and facilitates prioritized data packets processing and forwarding along the full data path, including driver, protocol stack and service software, down to the network interface card (NIC) hardware. Evaluation results on a prototype system show that the hardware-software co-designed LNS technique with label-driven prioritization has advantages of both Low tail latency and High concurrency.

References

1. Y. Zhang, et.al. Treadmill: Attributing the source of tail latency through precise load testing and statistical inference. ISCA-43 (2016).
2. Dean J, et.al. The tail at scale. Communications of the ACM. 56(2), 74-80 (2013).
3. Zats D, et.al. DeTail: Reducing the flow completion time tail in datacenter networks. ACM SIGCOMM Comput. Commun. Rev. 42, 139-150 (2012).
4. K. Asanovi'c. FireBox: A Hardware Building Block for2020Warehouse-ScaleComputers. FAST2014.
5. Zhiwei Xu, et.al. Low-entropy Cloud Computing Systems. ASCIENTIA SINICA Informationis; doi: 10.1360/N112017-00069.
6. E. Jeong, et.al.: mTCP: A Highly Scalable User-level TCP Stack for Multicore Systems. In Proceedings of NSDI '14 (2014).
7. Li J, et.al.: Tales of the Tail: Hardware, OS, and Application-level Sources of Tail Latency. symposium on cloud computing. 1-14 (2014).
8. http://acs.ict.ac.cn/network/MCC.html or HCMonitor.html.