Closing the Network Diagnostics Gap with Vigil

Behnaz Arzani*, Selim Ciraci †, Luiz Chamon*, Yibo Zhu‡, Hongqiang Liu‡
, Jitu Padhye‡, Geoff Outhred†, Boon Thau Loo*
* University of Pennsylvania, † Microsoft, ‡ Microsoft Research

CCS CONCEPTS

•Networks → Transport protocols; Error detection and error correction; *Network performance analysis; Network measurement;*

KEYWORDS

Data Centers, TCP, Network Diagnosis, Tomography

1 INTRODUCTION

Vigil started with an ambitious goal: For every TCP retransmission in our data centers, we wanted to pinpoint the network link that caused the packet drop that triggered the retransmission with negligible diagnostic overhead or changes to the networking infrastructure.

This goal may sound like an overkill—after all, TCP is supposed to be able to deal with a few packet losses. Packet losses might occur due to simple congestion instead of network equipment failures. Even network failures might be transient. Above all, there is a danger of drowning in a sea of data without generating any actionable intelligence.

These objections are valid, but so is the need to diagnose TCP "failures" which can result in severe problems for applications. For example, in our data centers, VM images are stored in a storage service. Even a small network outage can cause the host kernel to "panic" and reboot the VM. In fact, 17% of VM reboots in our data centers are caused by network issues and in over 70%, no monitoring systems was able to pinpoint the link(s) that caused the problem.

Since VM reboots directly affect the end customer, we place very high value on understanding their root causes. Any persistent pattern in such transient failures is a cause for concern and is potentially actionable. An example of such failure is silent packet drops [1]. Such problems are nearly impossible to detect with traditional monitoring systems (e.g., SNMP counters). If a switch is experiencing such problems, we may want to reboot or replace it. Such interventions are "costly" in that they affect a large number of VMs. Therefore, we need a system to correctly assign the blame in face of such transient failures.

None of the existing systems meet the ambitious goal we have set for ourselves. Pingmesh [1] sends periodic probes to detect link failures and can therefore leave "gaps" in coverage, as it must manage the overhead of probing. Also, since it uses out-of-band probes, it cannot detect failures that affect only in-band data. NetPoirot [2] identifies the network as a likely cause of performance issues, but cannot find the specific device that causes the problem. Roy et. al. [3] report a system that monitors all paths in the network for possible link failures. Their system requires modifications to routers and assumes a specific topology. Everflow [4] cannot be directly used to pinpoint the location of packet drop, since it would require capturing all traffic, which is not scalable.

We propose Vigil, a simple, lightweight, always-on monitoring tool. Vigil records the path of TCP connections that suffer from retransmission and assigns a proportional "blame" to each link on the path. It then provides a *ranking* of the links that represents their relative packet drop rates. Using this ranking, it can find the most likely cause of packet drops on each TCP connection.

Vigil does not require any changes to the existing networking infrastructure nor to the client software—the monitoring agent is an independent entity sitting on the side. Vigil detects in-band failures and is hence more useful than tools such as Pingmesh [1]. Vigil continues to perform well in the presence of noise. Finally, Vigil's overhead is negligible.

While the high-level design of Vigil is deceptively simple, the practical challenges of making Vigil work are non-trivial. For example, its path discovery is based on a traceroute-like approach. Due to the use of ECMP, traceroute packets have to be carefully crafted to ensure that they follow the same path as the TCP connection. Also, we needed to ensure that we do not overwhelm the routers along the path with traceroute packets (traceroute responses are handled by control-plane CPUs of the routers, which are quite puny). To this end, we need to do careful calculations to ensure that our sampling strikes the right balance between the need for accuracy and the overhead on the switches. On the theoretical side, we are able to show that Vigil's simple blame assignment scheme is highly accurate even in the presence of noise.

2 PROBLEM AND CHALLENGES

The goal of Vigil is to identify the cause of TCP retransmissions with high probability.

The design of Vigil is driven by two practical requirements: (i) it should scale to data center size networks and (ii) it should be deployable in a running data center with as little change to the infrastructure as possible.

There are a number of ways to identify the cause of packet drops. One can continuously monitor switch counters. These are inherently unreliable [5] and monitoring thousands of switches in a data center at a fine time granularity is not scalable. Having to correlate this data with each TCP retransmission *significantly* exacerbates this problem. One can use a system like PingMesh that sends probe packets to monitor link status. Such systems suffer from a trade-off: sending too many probes creates unacceptable overhead whereas reducing the probing rate leaves temporal and spatial gaps in coverage. More importantly, the probe traffic does not capture what the end user

SIGCOMM Posters and Demos '17, Los Angeles, CA, USA
 978-1-4503-5057-0/17/08...$15.00
DOI: 10.1145/3123878.3131979

and the TCP connections see. Thus, we choose the third alternative, which is to use data traffic itself as probe traffic [3]. Using data traffic has the advantage that the system introduces little or no monitoring overhead.

As one might expect, almost all traffic in our data centers is TCP traffic. One way to monitor this type of traffic is to use a system like Everflow. Everflow inserts a special tag in every packet and has the switches mirror tagged packets to special collection servers. Thus, if a tagged packet is dropped, we can easily determine the link on which it happened. Unfortunately, there is no way to know in advance which packet is going to be dropped, so we would have to tag and mirror every single TCP packet. This is clearly infeasible. We could tag only a fraction of packets, but doing so would incur in another sampling rate trade-off.

Hence, it follows that we must rely on some form of network tomography [6, 7, 8]. We can take advantage of the fact that TCP is a reliable delivery protocol so that any packet loss results in retransmission[1],which can be easily detected. If we knew the path of every TCP connection, we can set up a standard optimization problem to determine which link may have dropped the packet. A straightforward set cover optimization formulation that attempts to minimize the number of "blamed" links will correctly identify the cause of drops[2].

Still, there are two issues with this approach: (i) the optimization problem is known to be NP-hard [10] and solving it on the data-center scale is not feasible; (ii) tracking the path of every single TCP connection in the data center is not scalable in our setting.

One can use alternative solutions such as using Everflow to track the path of SYN packets or use a system like the one described in [3]. However, both these schemes rely on making changes to the switches. The only way to capture the path of a TCP connection without making any special infrastructure support is to run something like a traceroute. However, traceroute relies on getting ICMP TTL exceeded messages back from the switches. These messages are generated by the control plane, i.e., the switch CPU, not the ASIC that drives the dataplane. To avoid overloading the CPU, our datacenter administrators have capped the rate of ICMP responses to 100 per second. This severely limits the number of TCP connections we can track.

Given these limitations, what can we do? We have shown that the answer, for data center networks, is deceptively simple. We have showm that if (a) we track the path only of those TCP connections that have suffered retransmissions, (b) assign each link on the path of such a connection a vote of $1/h$, where h is the path length, and (c) sum up the votes during a given period, then the top-voted links are almost always the ones that are dropping packets! Unlike the solution of the optimization problem, our scheme is able to provide a *ranking* of the links in term of their drop rates, i.e. if link A has a higher total vote than B, it is also dropping more packets (with high probability). This allows Vigil to find the link most likely responsible for each connection's packet drops. We will show a brief description of Vigil's building blocks next, however, details of our findings are omitted due to space limitations.

[1]False retransmissions are rare and we handle them.

[2]Examples of such an optimization problem can be found in [9]

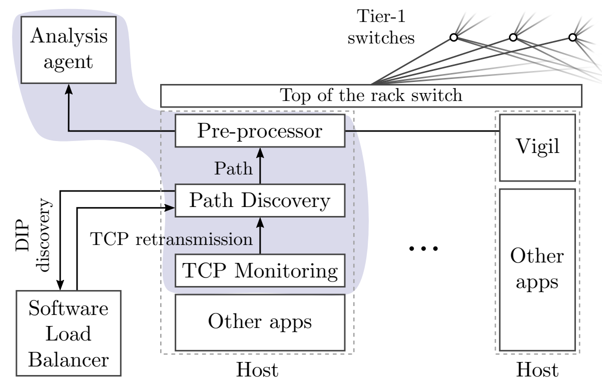

Figure 1: Overview of Vigil architecture

3 DESIGN OVERVIEW

Figure 1 shows the overall architecture of Vigil. Within our data centers, Vigil is deployed side-by-side other applications on each end-host as a user-level process running in the host OS. Vigil consists of three agents:

The *TCP monitoring agent* detects potential retransmissions at each end-host. The TCP monitoring agent is deployed on all end hosts in the data center.

Upon a retransmission, the TCP monitoring agent then triggers the *path discovery agent* to identify the set of links along this connection. The path discovery agent uses a modified version of traceroute to discover this path to the destination IP (DIP).

At each end-host, at regular intervals, a voting based scheme is carried out based on the reported paths of connections that suffered retransmission within the epoch. The results of the vote is sent to a centralized analysis agent to identify the top-voted links across the entire data center.

Overall, the Vigil implementation consists of 6000 lines of C code at the end-host. The analysis engine contains an additional several hundred lines of code. Its memory usage never goes beyond 6 KB on any of our production hosts and its CPU utilization is minimal (+1% overhead for each core). The bandwidth utilization of Vigil is also minimal (maximum of 200 KBps per host).

4 ONGOING WORK

We are currently evaluating a prototype of Vigil on a large scale production datacenter. There are various challenges that such a deployment would need to overcome. These include: handling NATs and software load balancers, avoiding false positives due to high traffic skews, as well as potential noise caused by ACK drops on the reverse path.

We are also working on a mathematical model of Vigil that allows us to quantify its worse case accuracy.

REFERENCES

[1] Guo, C., Yuan, L., Xiang, D., Dang, Y., Huang, R., Maltz, D., Liu, Z., Wang, V., Pang, B., Chen, H., et al. Pingmesh: A large-scale system for data center network latency measurement and analysis. In *ACM SIGCOMM* (2015), pp. 139–152.

[2] Arzani, B., Ciraci, S., Loo, B. T., Schuster, A., Outhred, G., et al. Taking the blame game out of data centers operations with NetPoirot. In *ACM SIGCOMM* (2016), pp. 440–453.

[3] Roy, A., Bagga, J., Zeng, H., and Sneoren, A. Passive realtime datacenter fault detection. In *ACM NSDI* (2017).

[4] Zhu, Y., Kang, N., Cao, J., Greenberg, A., Lu, G., Mahajan, R., Maltz, D., Yuan, L., Zhang, M., Zhao, B. Y., et al. Packet-level telemetry in large datacenter networks. In *ACM SIGCOMM* (2015), pp. 479–491.

[5] Wu, X., Turner, D., Chen, C.-C., Maltz, D. A., Yang, X., Yuan, L., and Zhang, M. NetPilot: Automating datacenter network failure mitigation. *ACM SIGCOMM Computer Communication Review 42*, 4 (2012), 419–430.

[6] Zhang, Y., Roughan, M., Willinger, W., and Qiu, L. Spatio-temporal compressive sensing and internet traffic matrices. *ACM SIGCOMM Computer Communication Review 39*, 4 (2009), 267–278.

[7] Ma, L., He, T., Swami, A., Towsley, D., Leung, K. K., and Lowe, J. Node failure localization via network tomography. In *ACM SIGCOMM IMC* (2014), pp. 195–208.

[8] Liu, C., He, T., Swami, A., Towsley, D., Salonidis, T., and Leung, K. K. Measurement design framework for network tomography using fisher information. *ITA AFM* (2013).

[9] Mysore, R. N., Mahajan, R., Vahdat, A., and Varghese, G. Gestalt: Fast, unified fault localization for networked systems. In *USENIX ATC* (2014), pp. 255–267.

[10] Bertsimas, D., and Tsitsiklis, J. N. *Introduction to linear optimization*. Athena Scientific, 1997.