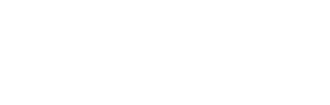


# A Kubernetes crime story

Tamás Kornai

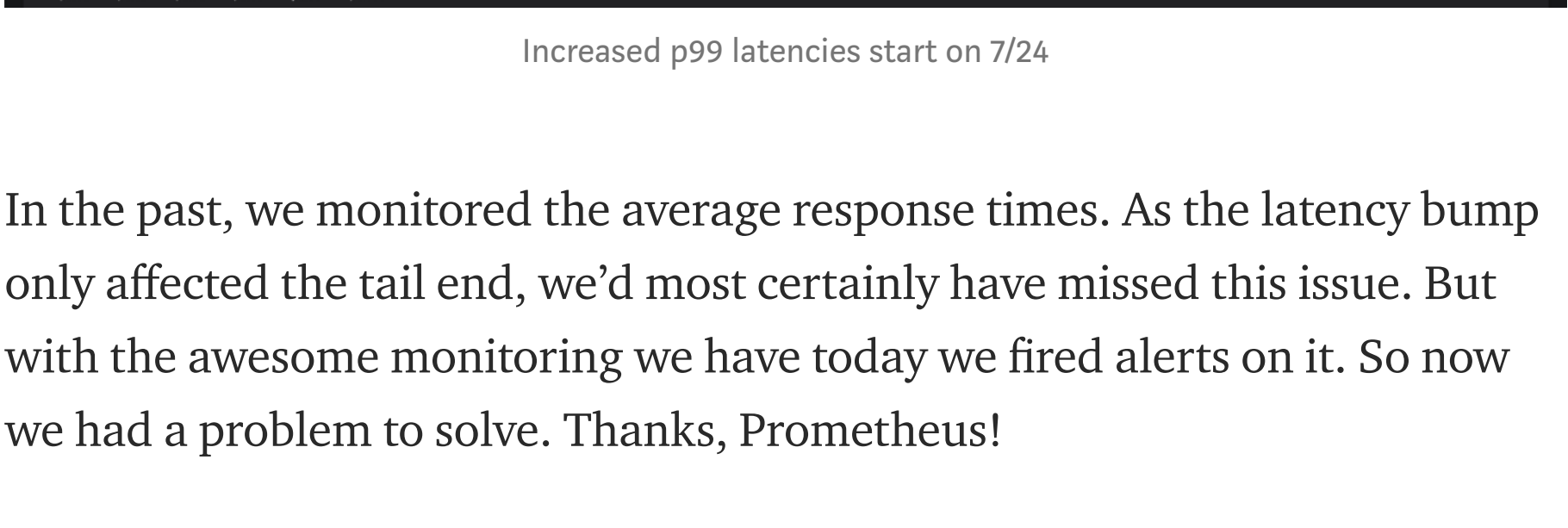
Follow

Oct 3, 2019 · 5 min read



## The crime

The problem with having a great monitoring system is that you notice problems you couldn't notice before. This is exactly what happened to us when we migrated our Nginx based reverse proxy to Kubernetes: we noticed that tail latency (p99) got somewhat higher for a few of our services.

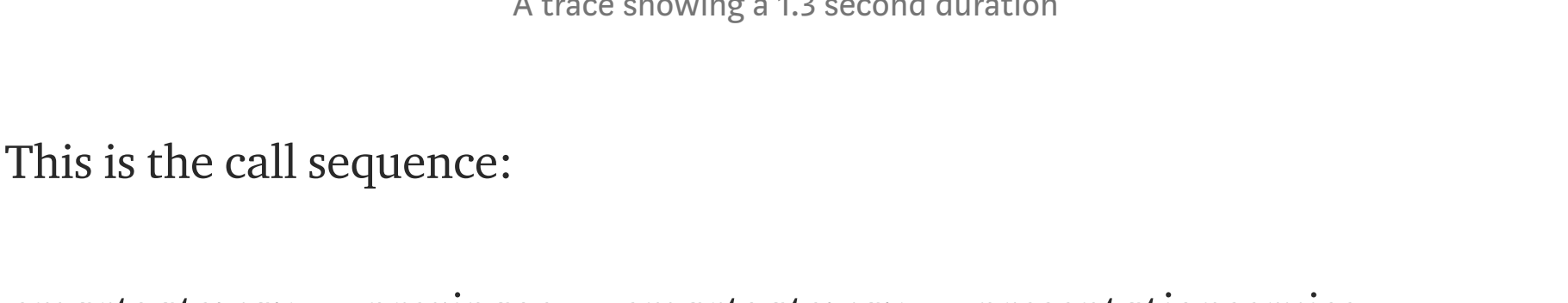


In the past, we monitored the average response times. As the latency bump only affected the tail end, we'd most certainly have missed this issue. But with the awesome monitoring we have today we fired alerts on it. So now we had a problem to solve. Thanks, Prometheus!

## The investigation

Higher tail latencies were visible for services that do many sequential service-to-service calls. We could reproduce the issue with stress testing, but we had absolutely no clue for what might have caused it. The only change we did was that we started to run the reverse proxy on shared Kubernetes worker nodes instead of dedicated ElasticBeanstalk machines. They were the same containers in similar private VPC subnets consuming about the same amount of resources.

So what can you do in a case like this? Our idea was to extend our investigation toolkit and introduce tracing so we can better understand why long requests take long. The easiest way seemed to be to integrate with AWS X-Ray, and we were not disappointed: after about two days we had traces flowing in.

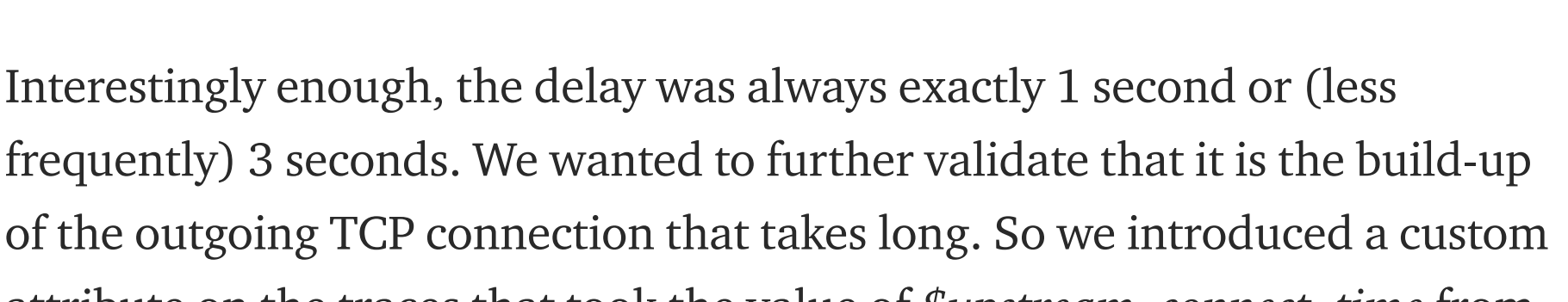


This is the call sequence:

```
smartgateway → prezipage → smartgateway → presentationservice
```

Smartgateway is the above mentioned reverse proxy. The first request on smartgateway is a user request, whereas the second is a service-to-service call from prezipage to presentationservice. In our current architecture service-to-service remote calls are also passing through the proxy.

For some reason, it took a full second for the remote call to get from smartgateway to presentationservice. Once the request appeared on presentationservice it only took 12.8 ms to serve it:



Interestingly enough, the delay was always exactly 1 second or (less frequently) 3 seconds. We wanted to further validate that it is the build-up of the outgoing TCP connection that takes long. So we introduced a custom attribute on the traces that took the value of `$upstream_connect_time` from [http://nginx.org/en/docs/http/nginx\\_http\\_upstream\\_module.html#variables](http://nginx.org/en/docs/http/nginx_http_upstream_module.html#variables). As it turned out, the assumption was valid, we could observe the expected upstream connection time delays.

## The offender

At this point, we felt that we are onto something and turned to google for more pointers. Luckily enough, the search term “kubernetes slow outgoing connections” returns this great article as the first result: <https://tech.xing.com/a-reason-for-unexplained-connection-timeouts-on-kubernetes-docker-abd041cf7e02>

In this post, the author explains how they tracked down a timeout issue with the exact same characteristics. It's a great read with crystal clear explanations, so you should go and read it. We will wait here until you do.

## The verdict

So welcome back!

I trust you have read the article, but let me just give a very short and high-level summary. When a pod has to connect to an external service (by external I mean a service with a public IP), the outgoing packet will need to go through a source network address translation (SNAT) operation. By default, this is happening on the worker node, configured by iptables rules. There is a race condition in the underlying Netfilter module though. Sometimes SYN packages are dropped when we try to insert new connection tuples into the conntrack table, leading to 1 or 3 seconds connection timeouts. (If the previous few sentences did not make sense, then reading the referenced article one more time might be a good idea.)

So let's discuss how our problem was different.

We are using AWS EKS, so the networking stack is different from what is described in the article. The proposed solution to change the port allocation algorithm to fully-random is unfortunately not yet available on EKS. The [amazon-vpc-cni-k8s](#) plugin does support that setting:

```
AWS_VPC_K8S_CNI_RANDOMIZESNAT=prng
```

But unfortunately, the iptables version that comes with the official AMI does not. Luckily there is another option that we can use:

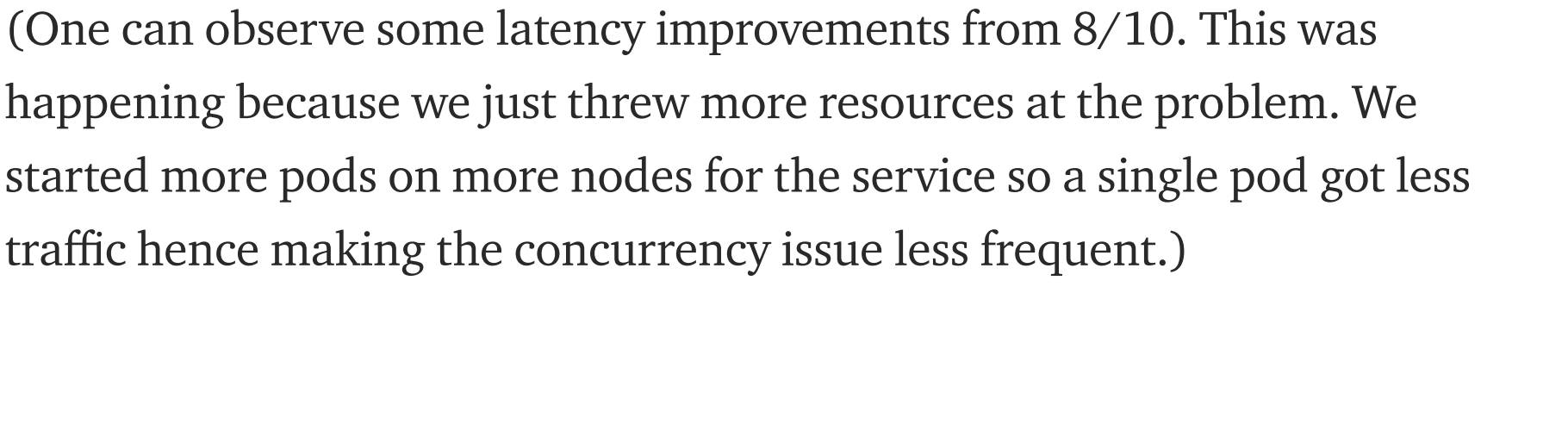
```
AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

With that option, we can turn off SNAT altogether on the node. You can read the slightly misleading documentation [here](#).

Why is it misleading? There is an open PR explaining it: <https://github.com/awsdocs/amazon-eks-user-guide/pull/53>

## The redemption

After we applied the `AWS_VPC_K8S_CNI_EXTERNALSNAT` environment variable on 8/15, latency numbers returned to the baseline that we had before the reverse proxy migration.








(One can observe some latency improvements from 8/10. This was happening because we just threw more resources at the problem. We started more pods on more nodes for the service so a single pod got less traffic hence making the concurrency issue less frequent.)


## The moral of the story

Observability is key when debugging a complex system, like Kubernetes. By adding tracing we could better understand the characteristics of slow requests. With that information at hand, we got very close to the root of the problem: we just needed a single lucky search query and a great article explaining what we are staring at. For the actual solution, we still had to wrap our head around how SNAT works, dive deep into Amazon Kubernetes (EKS) internals and understand the role of what the `amazon-vpc-cni-k8s` plugin plays. Hopefully, with the solution provided above and with the yet-to-be-improved documentation, we can help some fellow engineers on their Kubernetes journey.

KubernetesNatTracingEngineeringDebugging

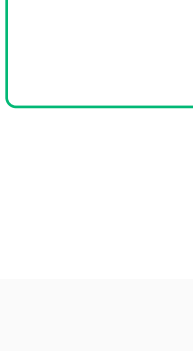
121 claps



WRITTEN BY

Tamás Kornai

Follow

Prezi Engineering

The things we learn as we build our products

Follow

Write the first response

### More From Medium

- When building an app, what should you be logging?

Lucas Jellema in JavaScript In Plain English

Debug using Breakpoints in Xcode

Steven Curtis

Fantastic Probes And How To Configure Them — A Kubernetes Story

Ricardo A. in The Startup

Vertical Pod Autoscaler deep dive, limitations and real-world examples

Daniel Megyesi in Infrastructure adventures
- Kubernetes Container Resource Requirements — Part 2: CPU

Will Tomlin in Expedia Group Technology

The Chase After a Sometimes-Bug

Lea Cohen in The Startup

Kubernetes Garbage Collection

Bharat

Garbage Collection in Kubernetes

Abhishek Sharma