tetrate

THE ENTERPRISE SERVICE MESH COMPANY

Devarajan Ramaswamy
Engineer

Nizam Uddin
Engineer

Scaling To 1M RPS With Multi Cluster Istio

IstioCon

#IstioCon

AGENDA

1 REQUIREMENTS: HIGH THROUGHPUT AT SCALE

2 SETUP & TOPOLOGY

3 TOOLS USED

4 EXERCISE: MINIMALISTIC CASE ~1000 RPS

5 EXERCISE: STAGE 2 LOAD >5000 RPS

6 EXERCISE: STAGE 3 LOAD >25,000 RPS

7 EXERCISE: FULL SCALE LOAD >500,000 RPS

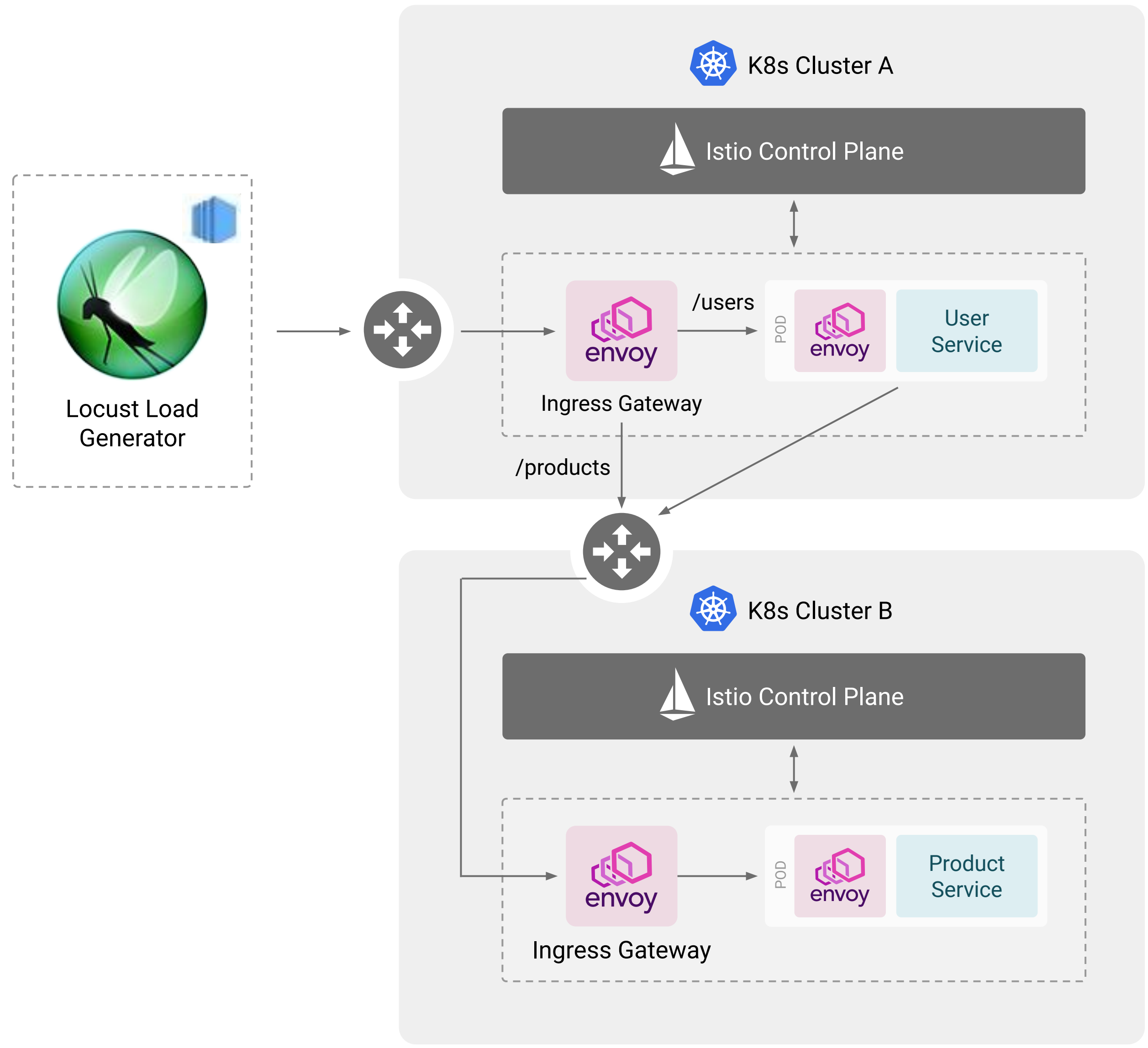
Requirements

HIGH THROUGHPUT AT SCALE

- **Context**
    - Users send http/1 requests and connections are left open.
    - Traffic increase is effected through increasing number of users connected.
    - System should auto scale as throughput goes up
- **Expectation**
    - System should be capable of handling 500k RPS
    - Latency should remain within a small band with the tail (p99) less than 500% of median response time for the worst case
    - System should show ability to handle traffic up to 1m RPS (this maps to ~2.5m users)

SETUP & TOPOLOGY

- Two kubernetes clusters with Istio 1.9.8 installed and application services deployed as part of mesh.
- Locust server on VMs (master/slave mode) that generates connection oriented http traffic load.



TOOLS USED

APPLICATION

A Go based application

The Application consisting of ‘user’ and ‘product’ microservices have APIs to simulate the typical create, get and getAll methods.

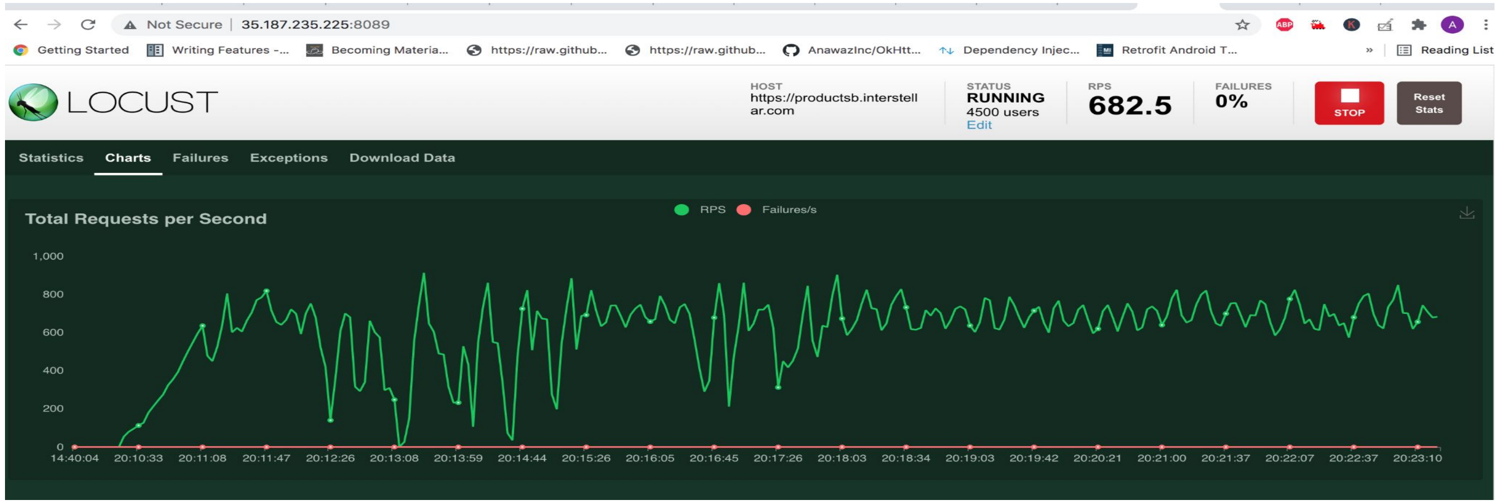
LOCUST

An open source load testing tool, define user behaviour with Python code, and swarm your system with millions of simultaneous users.

LOAD GENERATOR

EXERCISE: MINIMALISTIC CASE ~1000 RPS

Idea here is to have a single pod for all the services and test the limits of the system in terms of throughput and latency.As we increased the users to around 4000 (~700 RPS), we noticed the latency started to increase and had significant variations over time



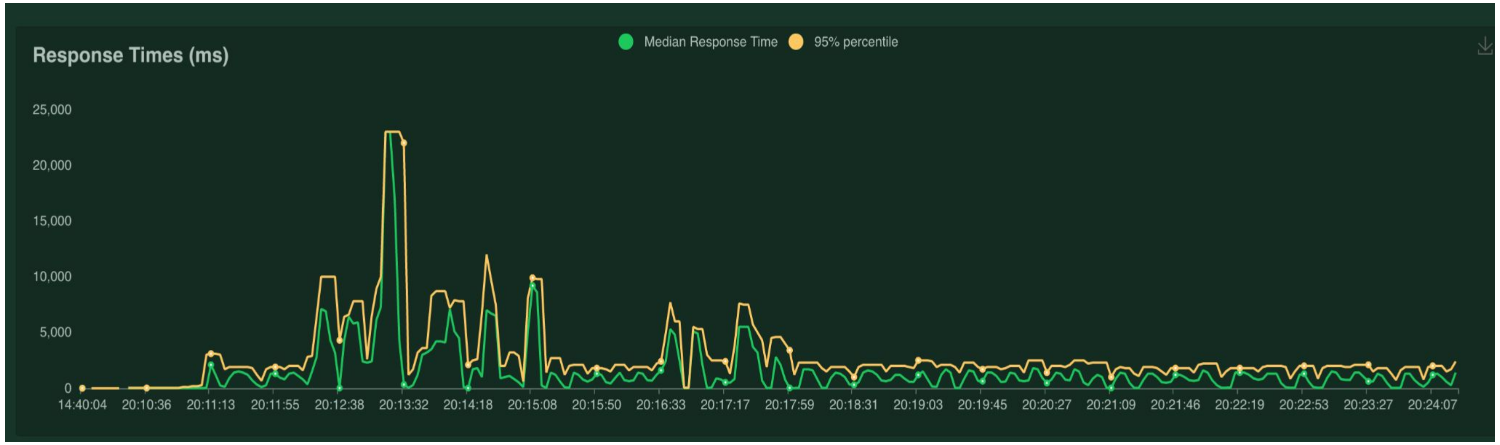
Not Secure | 35.187.235.225:8089

HOST
https://productsb.interstellar.com

STATUS
RUNNING
4500 users
Edit

RPS
682.5

FAILURES
0%



tetrate

EXERCISE: MINIMALISTIC CASE ~1000 RPS

A few things we could do quickly to steady the graph and increase the throughput

Tune Sidecar Concurrency

Applications containers were thin. Increasing sidecar concurrency improved the throughput. After experimenting a little, we settled with a concurrency of '4'

Reduce Trace Sampling Rate

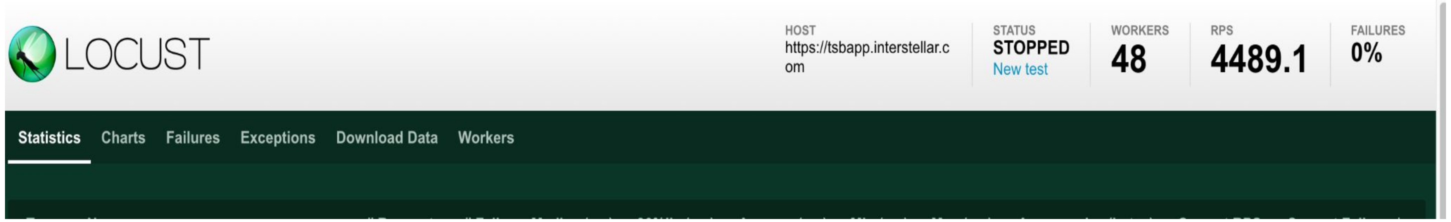
The installation default 1% sampling rate was an overkill. Considering the high traffic rate requirement of the exercise, we reduced it to 0.01%

Upgrade to HTTP2

Upgrading the traffic from HTTP/1 to HTTP/2 helped us multiplexing several requests over the same connection and avoided the connection creation overheads.

EXERCISE:
STAGE 2 LOAD 5000 RPS

As we increased the throughput, pods were getting auto scaled serving the increased traffic. However, as we progressed, we started seeing the response time graph becoming jagged with wild variations.



LOCUST

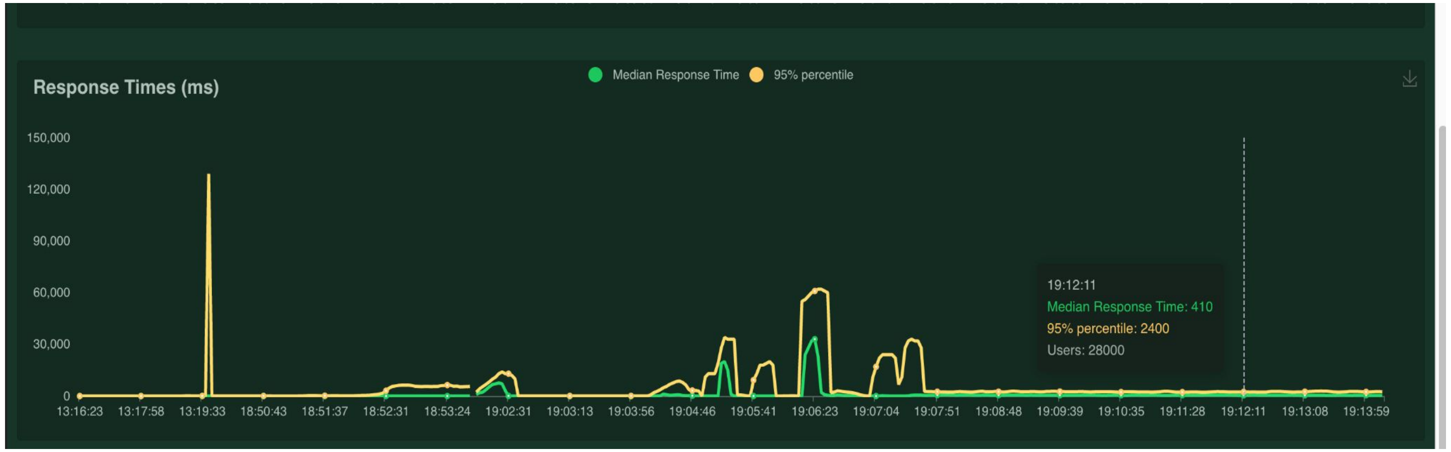
HOST https://tsbapp.interstellar.com

STATUS STOPPED

WORKERS 48

RPS 4489.1

FAILURES 0%



Response Times (ms)

19:12:11
Median Response Time: 410
95% percentile: 2400
Users: 28000

EXERCISE:
STAGE 2 LOAD 5000 RPS

A quick analysis revealed the following:

1. Pods autoscaling was not in synchrony among different deployments in the traffic path
2. Pod clustering on nodes
3. ControlPlane pods on dataplane nodes

Resource Optimization

Optimize resource requests, limits and HPA settings

Even Pods distribution

Set preferred pod anti affinity to uniformly distribute pods across all the nodes

Segregate NodePools

Separate out node-pools for Control Plane and Data Plane traffic

Load Balancing Options

With high degree of heterogeneous requests, round_robin does not yield the desired results. Rather least_request that has knowledge of pending requests on a connection serves better

ROUND_ROBIN

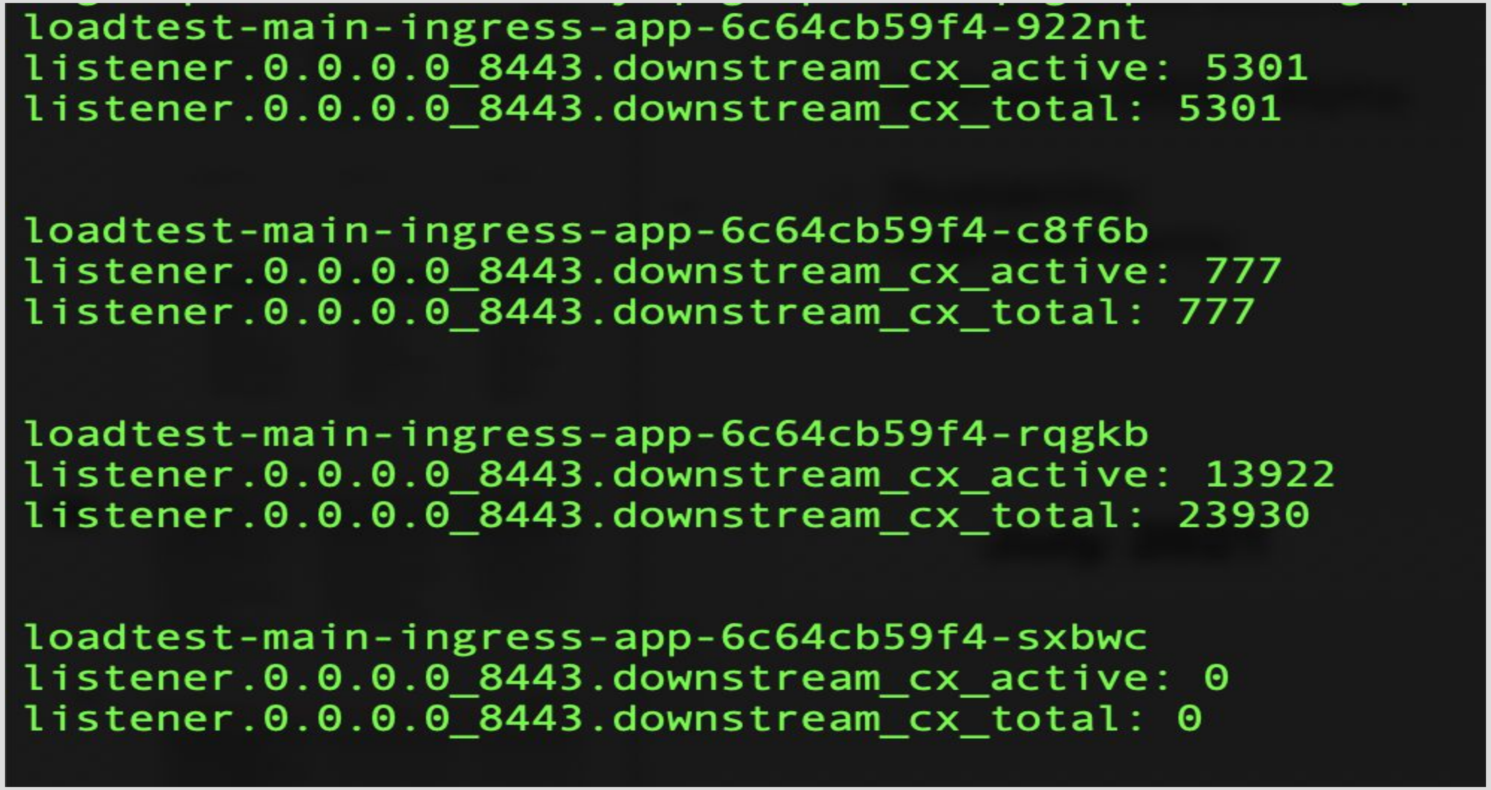
```
# Destination Rule
spec:
 host: example.svc
 trafficPolicy:
   loadBalancer:
     simple: ROUND_ROBIN
```

LEAST_REQUEST

```
# Destination Rule
spec:
 host: user.svc
 trafficPolicy:
   loadBalancer:
     simple: LEAST_CONN

# Envoy filter
patch:
     operation: MERGE
     value:
       least_request_lb_config:
         choice_count: 10
```

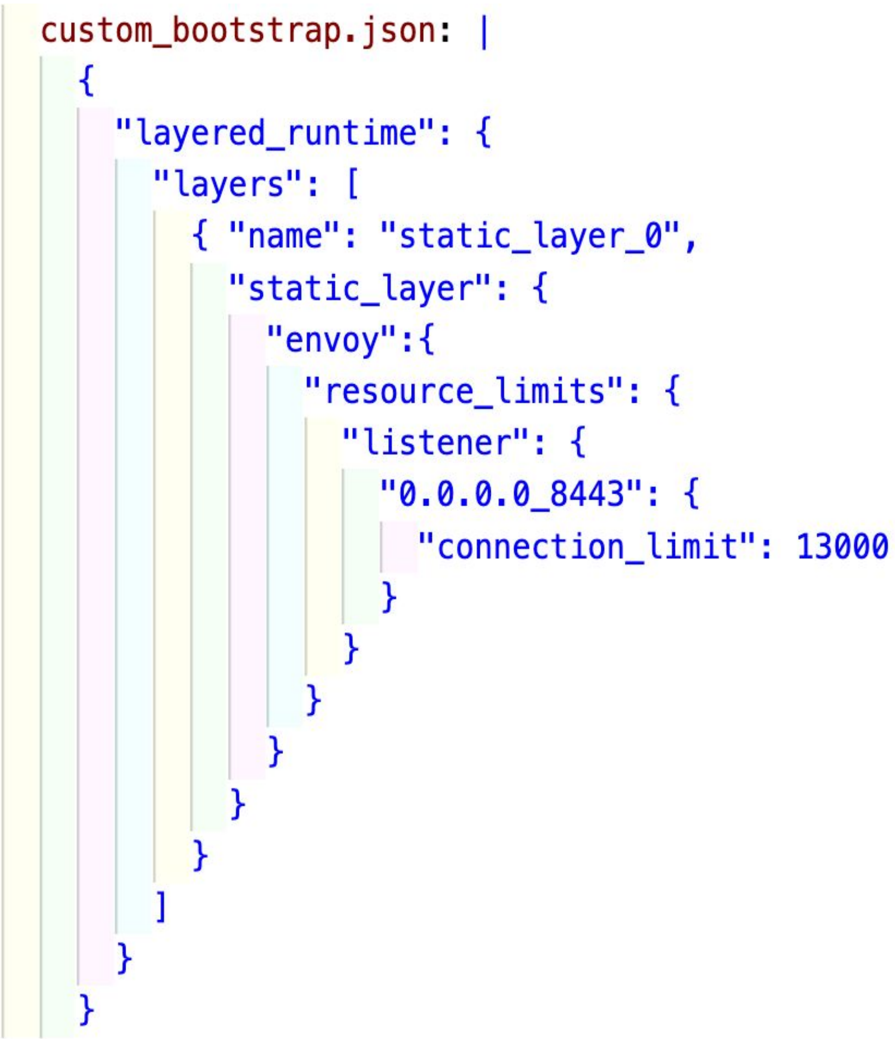
Downstream Connection Pile Up

As new connections from Locust client were constantly being added round robin among the ingress pods, the older the pod, the more the connections and the traffic it served. This caused a huge tail from overloaded ingress pods

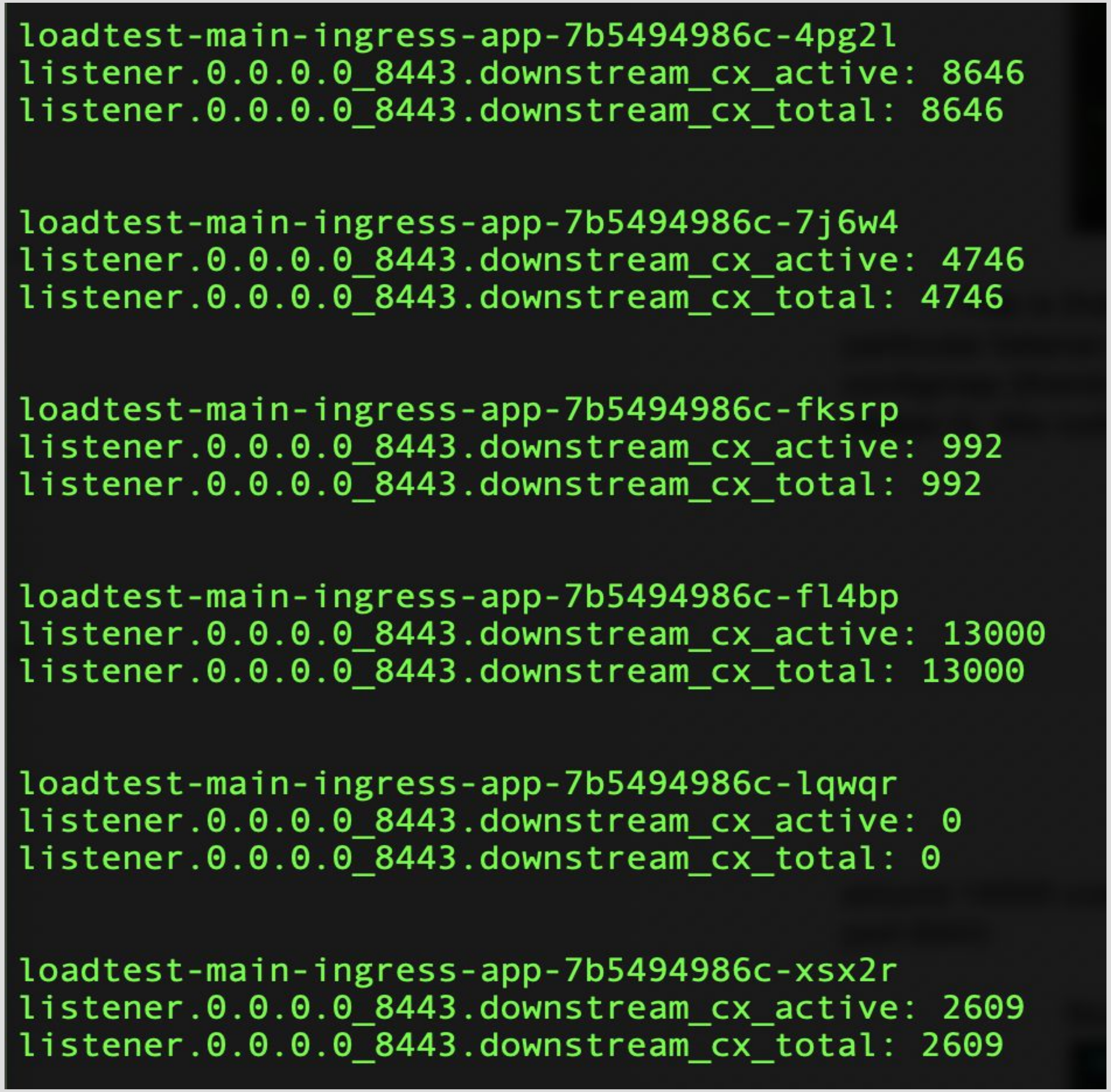


```
loadtest-main-ingress-app-6c64cb59f4-922nt
listener.0.0.0.0_8443.downstream_cx_active: 5301
listener.0.0.0.0_8443.downstream_cx_total: 5301


loadtest-main-ingress-app-6c64cb59f4-c8f6b
listener.0.0.0.0_8443.downstream_cx_active: 777
listener.0.0.0.0_8443.downstream_cx_total: 777


loadtest-main-ingress-app-6c64cb59f4-rqgkb
listener.0.0.0.0_8443.downstream_cx_active: 13922
listener.0.0.0.0_8443.downstream_cx_total: 23930


loadtest-main-ingress-app-6c64cb59f4-sxbwc
listener.0.0.0.0_8443.downstream_cx_active: 0
listener.0.0.0.0_8443.downstream_cx_total: 0
```

Limit Downstream Connections

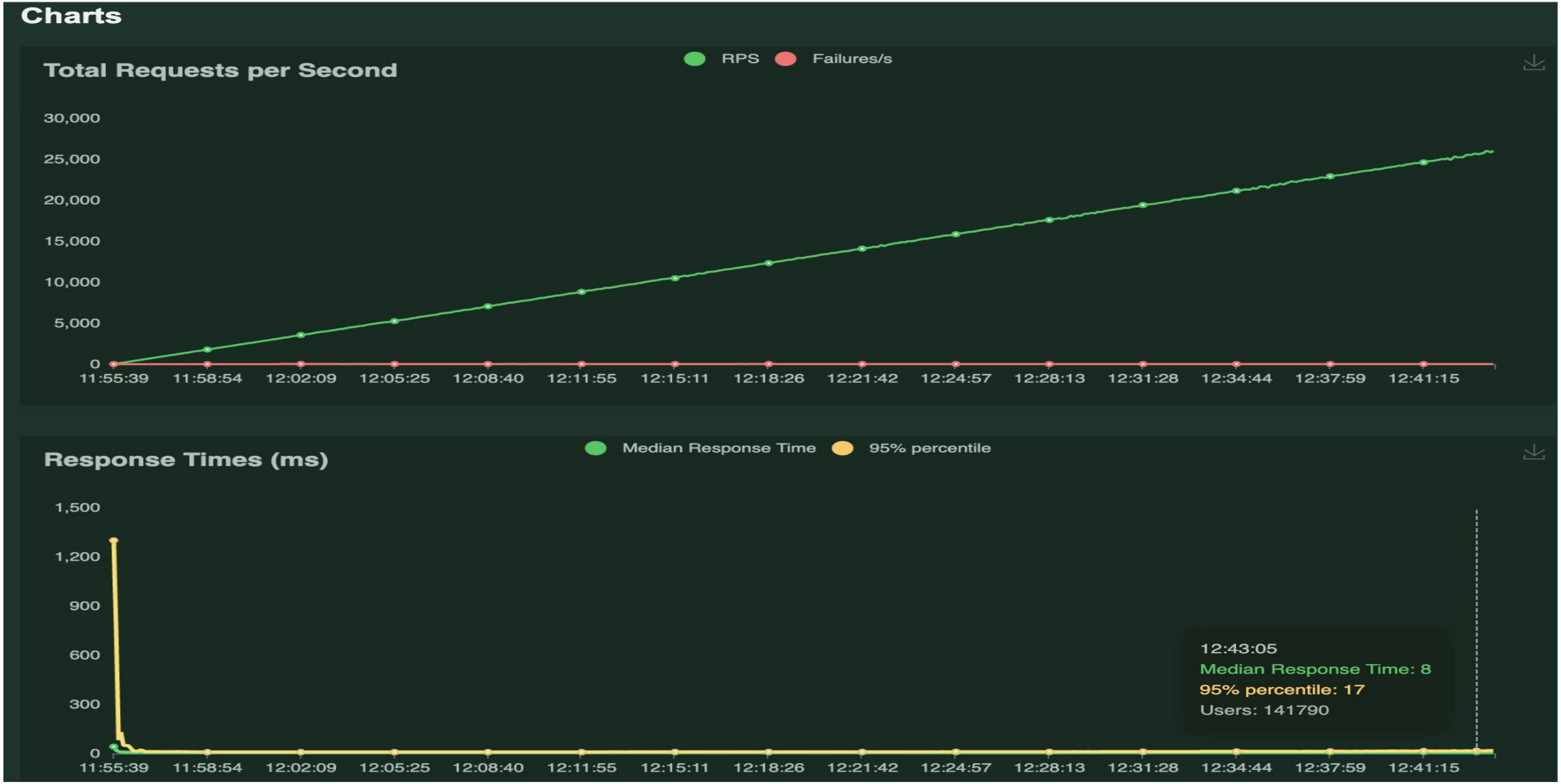


```
custom_bootstrap.json: |
  {
    "layered_runtime": {
      "layers": [
        { "name": "static_layer_0",
          "static_layer": {
            "envoy":{
              "resource_limits": {
                "listener": {
                  "0.0.0.0_8443": {
                    "connection_limit": 13000
                  }
                }
              }
            }
          }
        }
      ]
    }
  }
```



```
loadtest-main-ingress-app-7b5494986c-4pg2l
listener.0.0.0.0_8443.downstream_cx_active: 8646
listener.0.0.0.0_8443.downstream_cx_total: 8646


loadtest-main-ingress-app-7b5494986c-7j6w4
listener.0.0.0.0_8443.downstream_cx_active: 4746
listener.0.0.0.0_8443.downstream_cx_total: 4746


loadtest-main-ingress-app-7b5494986c-fksrp
listener.0.0.0.0_8443.downstream_cx_active: 992
listener.0.0.0.0_8443.downstream_cx_total: 992


loadtest-main-ingress-app-7b5494986c-fl4bp
listener.0.0.0.0_8443.downstream_cx_active: 13000
listener.0.0.0.0_8443.downstream_cx_total: 13000


loadtest-main-ingress-app-7b5494986c-lqwqr
listener.0.0.0.0_8443.downstream_cx_active: 0
listener.0.0.0.0_8443.downstream_cx_total: 0


loadtest-main-ingress-app-7b5494986c-xsx2r
listener.0.0.0.0_8443.downstream_cx_active: 2609
listener.0.0.0.0_8443.downstream_cx_total: 2609
```

Result with 1-25,000 RPS



12:43:05
Median Response Time: 8
95% percentile: 17
Users: 141790

EXERCISE: STAGE 3 LOAD > 25000 RPS

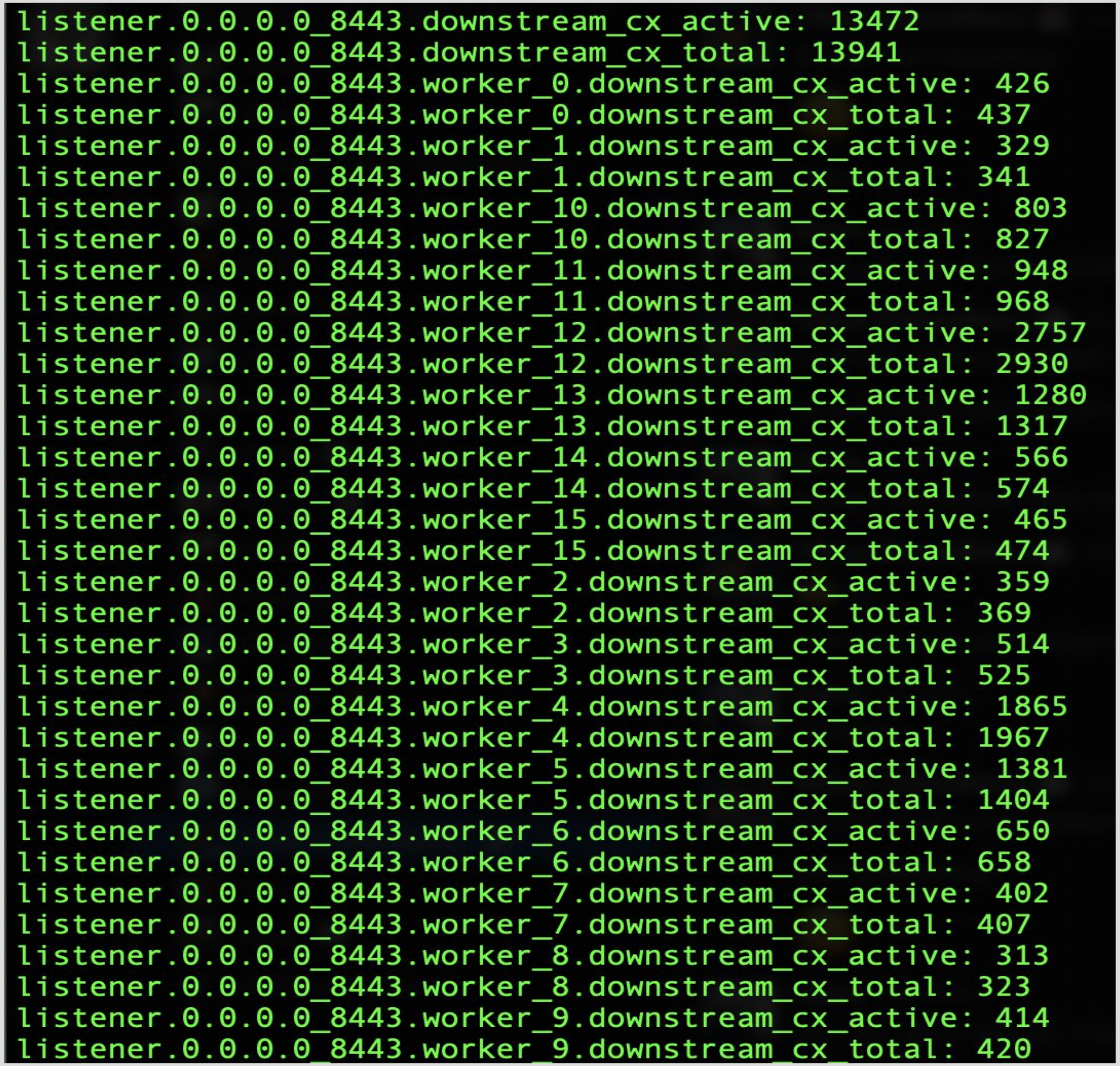
1. More app and ingress gateway pods getting scheduled on the same node at higher traffic rates as HPA kicks in
2. Default concurrency of gateway pods (number of cores) made little sense

Limiting Ingress gateways concurrency

As the number of cores is limited, the net increase in the number of ingress gateway pods' threads had the potential to add to the tail. In our case, we set the gateway concurrency to 6 threads. This concurrency number would vary on a case by case basis

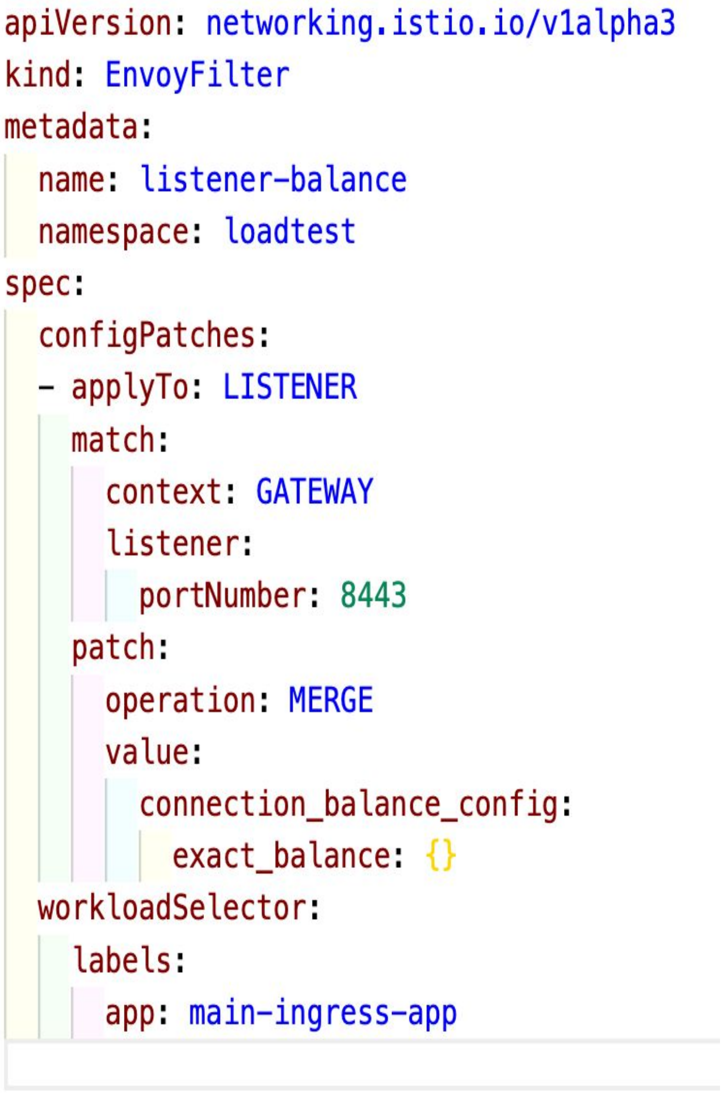
Connection Balance across Worker Threads

Uneven downstream connection distribution not only makes some threads to bear more load, but also resulted in skewed distribution of requests over the upgraded HTTP2 connections.

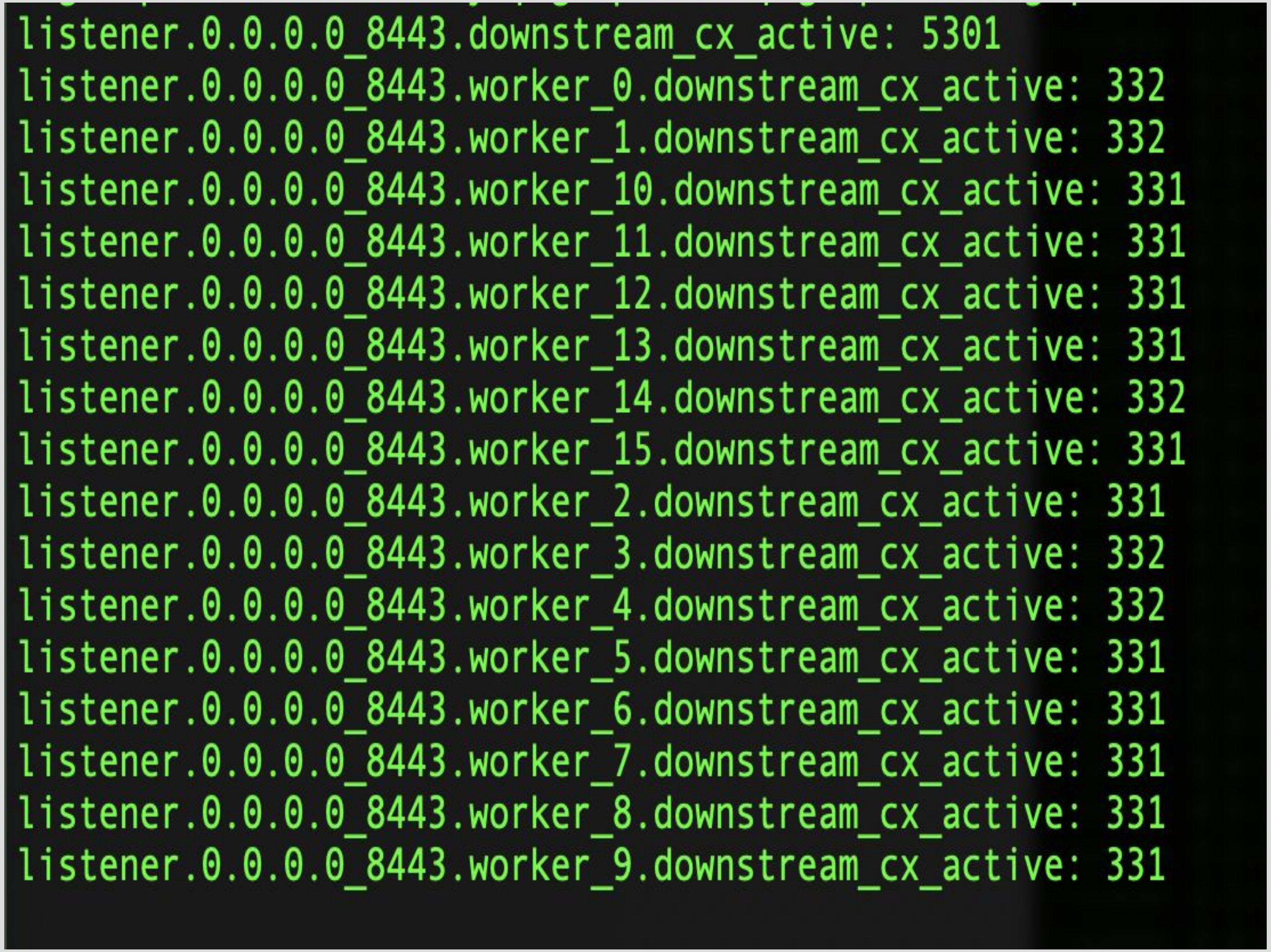


```
listener.0.0.0.0_8443.downstream_cx_active: 13472
listener.0.0.0.0_8443.downstream_cx_total: 13941
listener.0.0.0.0_8443.worker_0.downstream_cx_active: 426
listener.0.0.0.0_8443.worker_0.downstream_cx_total: 437
listener.0.0.0.0_8443.worker_1.downstream_cx_active: 329
listener.0.0.0.0_8443.worker_1.downstream_cx_total: 341
listener.0.0.0.0_8443.worker_10.downstream_cx_active: 803
listener.0.0.0.0_8443.worker_10.downstream_cx_total: 827
listener.0.0.0.0_8443.worker_11.downstream_cx_active: 948
listener.0.0.0.0_8443.worker_11.downstream_cx_total: 968
listener.0.0.0.0_8443.worker_12.downstream_cx_active: 2757
listener.0.0.0.0_8443.worker_12.downstream_cx_total: 2930
listener.0.0.0.0_8443.worker_13.downstream_cx_active: 1280
listener.0.0.0.0_8443.worker_13.downstream_cx_total: 1317
listener.0.0.0.0_8443.worker_14.downstream_cx_active: 566
listener.0.0.0.0_8443.worker_14.downstream_cx_total: 574
listener.0.0.0.0_8443.worker_15.downstream_cx_active: 465
listener.0.0.0.0_8443.worker_15.downstream_cx_total: 474
listener.0.0.0.0_8443.worker_2.downstream_cx_active: 359
listener.0.0.0.0_8443.worker_2.downstream_cx_total: 369
listener.0.0.0.0_8443.worker_3.downstream_cx_active: 514
listener.0.0.0.0_8443.worker_3.downstream_cx_total: 525
listener.0.0.0.0_8443.worker_4.downstream_cx_active: 1865
listener.0.0.0.0_8443.worker_4.downstream_cx_total: 1967
listener.0.0.0.0_8443.worker_5.downstream_cx_active: 1381
listener.0.0.0.0_8443.worker_5.downstream_cx_total: 1404
listener.0.0.0.0_8443.worker_6.downstream_cx_active: 650
listener.0.0.0.0_8443.worker_6.downstream_cx_total: 658
listener.0.0.0.0_8443.worker_7.downstream_cx_active: 402
listener.0.0.0.0_8443.worker_7.downstream_cx_total: 407
listener.0.0.0.0_8443.worker_8.downstream_cx_active: 313
listener.0.0.0.0_8443.worker_8.downstream_cx_total: 323
listener.0.0.0.0_8443.worker_9.downstream_cx_active: 414
listener.0.0.0.0_8443.worker_9.downstream_cx_total: 420
```

Connection Balance across Worker Threads

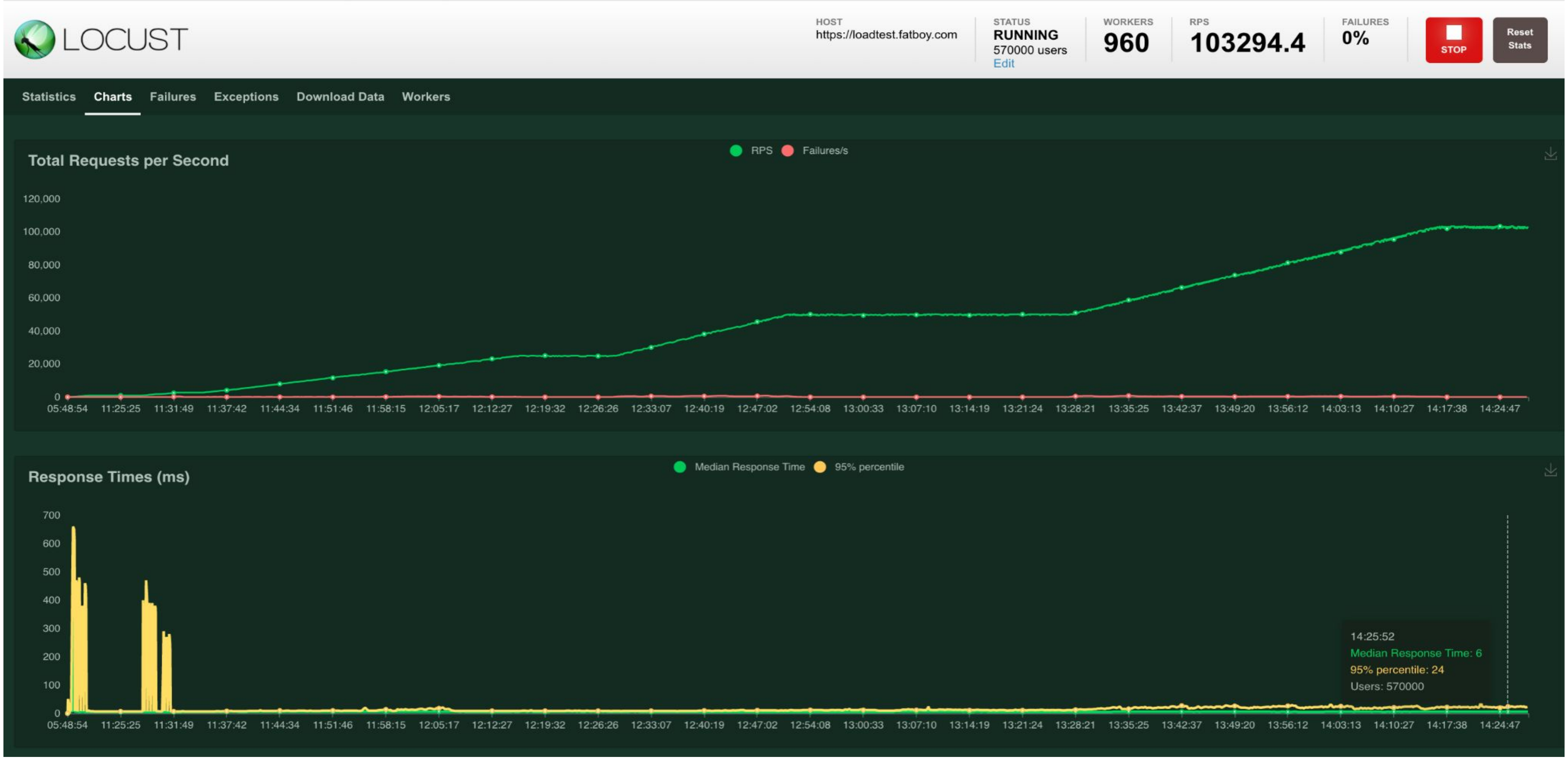


```yaml
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: listener-balance
  namespace: loadtest
spec:
  configPatches:
  - applyTo: LISTENER
    match:
      context: GATEWAY
      listener:
        portNumber: 8443
    patch:
      operation: MERGE
      value:
        connection_balance_config:
          exact_balance: {}
  workloadSelector:
    labels:
      app: main-ingress-app
```



```
listener.0.0.0.0_8443.downstream_cx_active: 5301
listener.0.0.0.0_8443.worker_0.downstream_cx_active: 332
listener.0.0.0.0_8443.worker_1.downstream_cx_active: 332
listener.0.0.0.0_8443.worker_10.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_11.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_12.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_13.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_14.downstream_cx_active: 332
listener.0.0.0.0_8443.worker_15.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_2.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_3.downstream_cx_active: 332
listener.0.0.0.0_8443.worker_4.downstream_cx_active: 332
listener.0.0.0.0_8443.worker_5.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_6.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_7.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_8.downstream_cx_active: 331
listener.0.0.0.0_8443.worker_9.downstream_cx_active: 331
```

Result with >100,000 RPS



HOST
https://loadtest.fatboy.com

STATUS
RUNNING
570000 users

WORKERS
960

RPS
103294.4

FAILURES
0%

Total Requests per Second

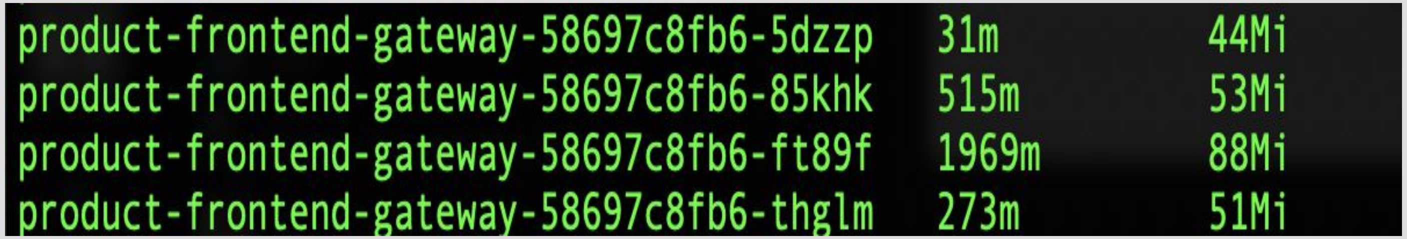
Response Times (ms)

14:25:52
Median Response Time: 6
95% percentile: 24
Users: 570000

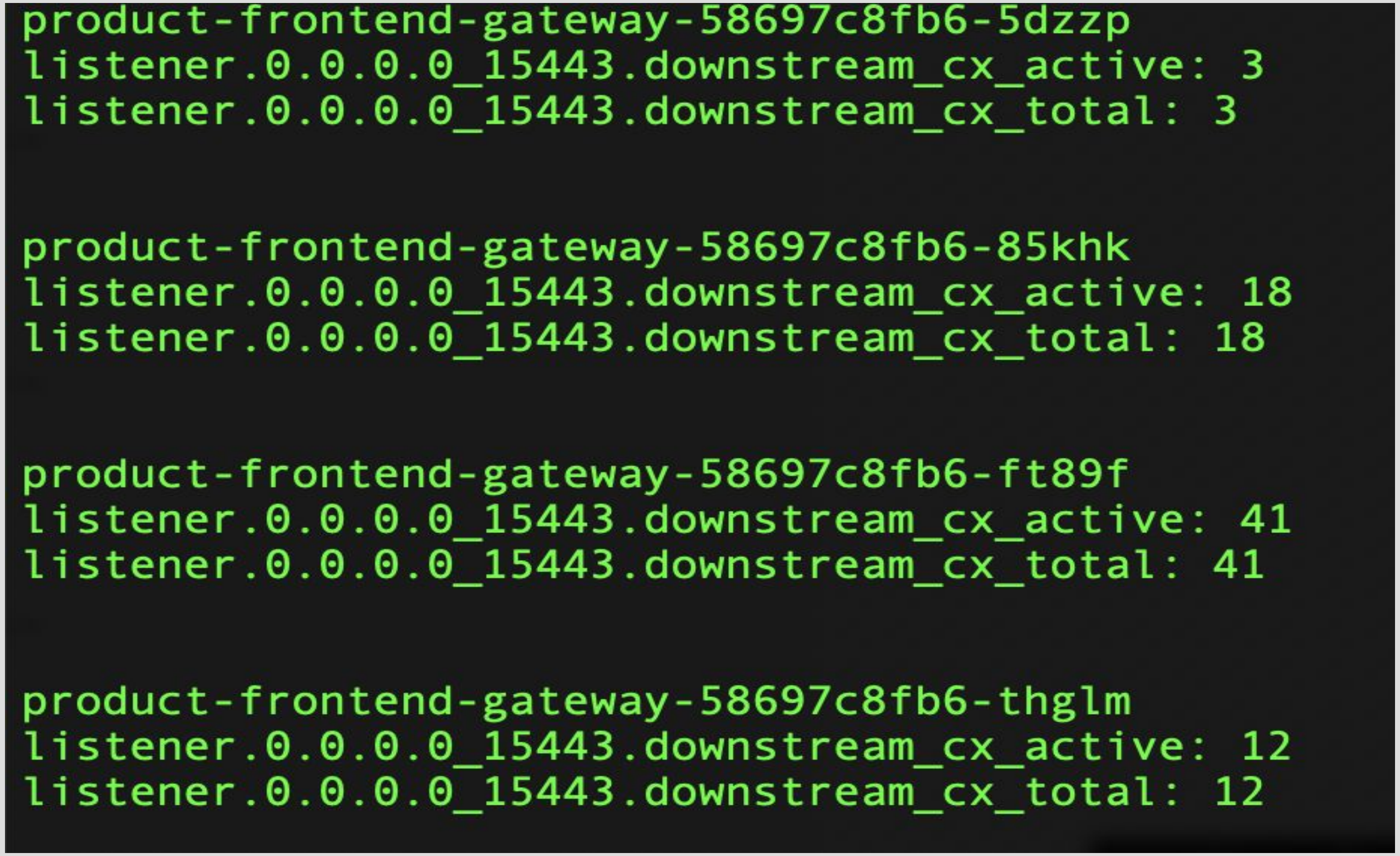
EXERCISE: FULL SCALE LOAD 500,000 RPS

Improvise Cross Cluster Endpoints Discovery

- Too many hops in cross cluster communication
- Client side envoys would see only one endpoint that of the NLB and hence a single h2 connection per client thread. In such scenarios, NLB + kubeproxy combination is likely to result in uneven connection distribution at the remote pods
- From the client envoy perspective new remote server pods would not be visible and hence wouldn't become part of connection LB pool.



```
product-frontend-gateway-58697c8fb6-5dzzp   31m     44Mi
product-frontend-gateway-58697c8fb6-85khk   515m    53Mi
product-frontend-gateway-58697c8fb6-ft89f   1969m   88Mi
product-frontend-gateway-58697c8fb6-thglm   273m    51Mi
```



```
product-frontend-gateway-58697c8fb6-5dzzp
listener.0.0.0.0_15443.downstream_cx_active: 3
listener.0.0.0.0_15443.downstream_cx_total: 3


product-frontend-gateway-58697c8fb6-85khk
listener.0.0.0.0_15443.downstream_cx_active: 18
listener.0.0.0.0_15443.downstream_cx_total: 18


product-frontend-gateway-58697c8fb6-ft89f
listener.0.0.0.0_15443.downstream_cx_active: 41
listener.0.0.0.0_15443.downstream_cx_total: 41


product-frontend-gateway-58697c8fb6-thglm
listener.0.0.0.0_15443.downstream_cx_active: 12
listener.0.0.0.0_15443.downstream_cx_total: 12
```

EXERCISE: FULL SCALE LOAD 500,000 RPS

Improvise Cross Cluster Endpoints Discovery...

NodePort Approach

- *Expose remote gateway as NodePort type service. Client envoy sees node IPs instead of single NLB IP*
- *Change externalTrafficPolicy to local of the remote gateway Service. Better load balancing if pods are evenly distributed across nodes*
- *Works best if nodes are prescaled for remote gateway pod scheduling*
- *Needs mechanism to discover node IPs of remote cluster*
- *Remote node IPs have to be reachable*

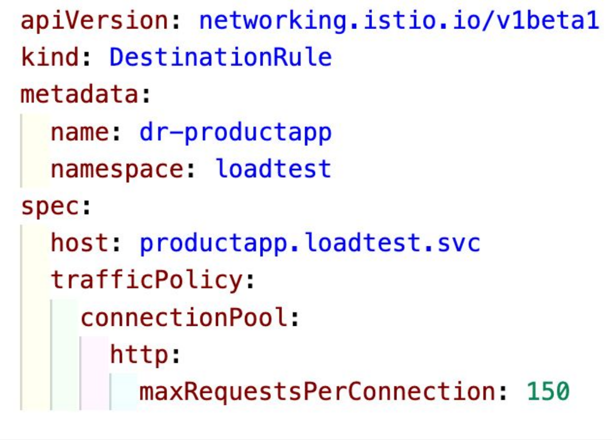
EXERCISE: FULL SCALE LOAD 500,000 RPS

Improvise Cross Cluster Endpoints Discovery...

Stick to NLB

- *Only way to distribute client h2 connections near evenly across the remote gateway pods is to frequently close and reconnect*
- *Need to figure out the optimal logic to determine when to close the connections for the remote host*
- *Closing of connections from server end would result in uptick of errors*
- *Would still have to go through NLB and extra hop across nodes between kubeproxy and gateway pod*
- *There would be a marginal increase in overall latency due to frequent closing and reopening of connections*

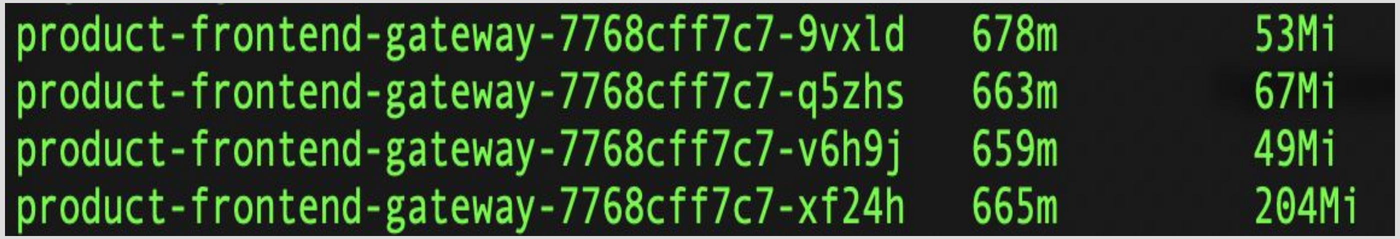
Tune:- MaxRequestsPerConnection



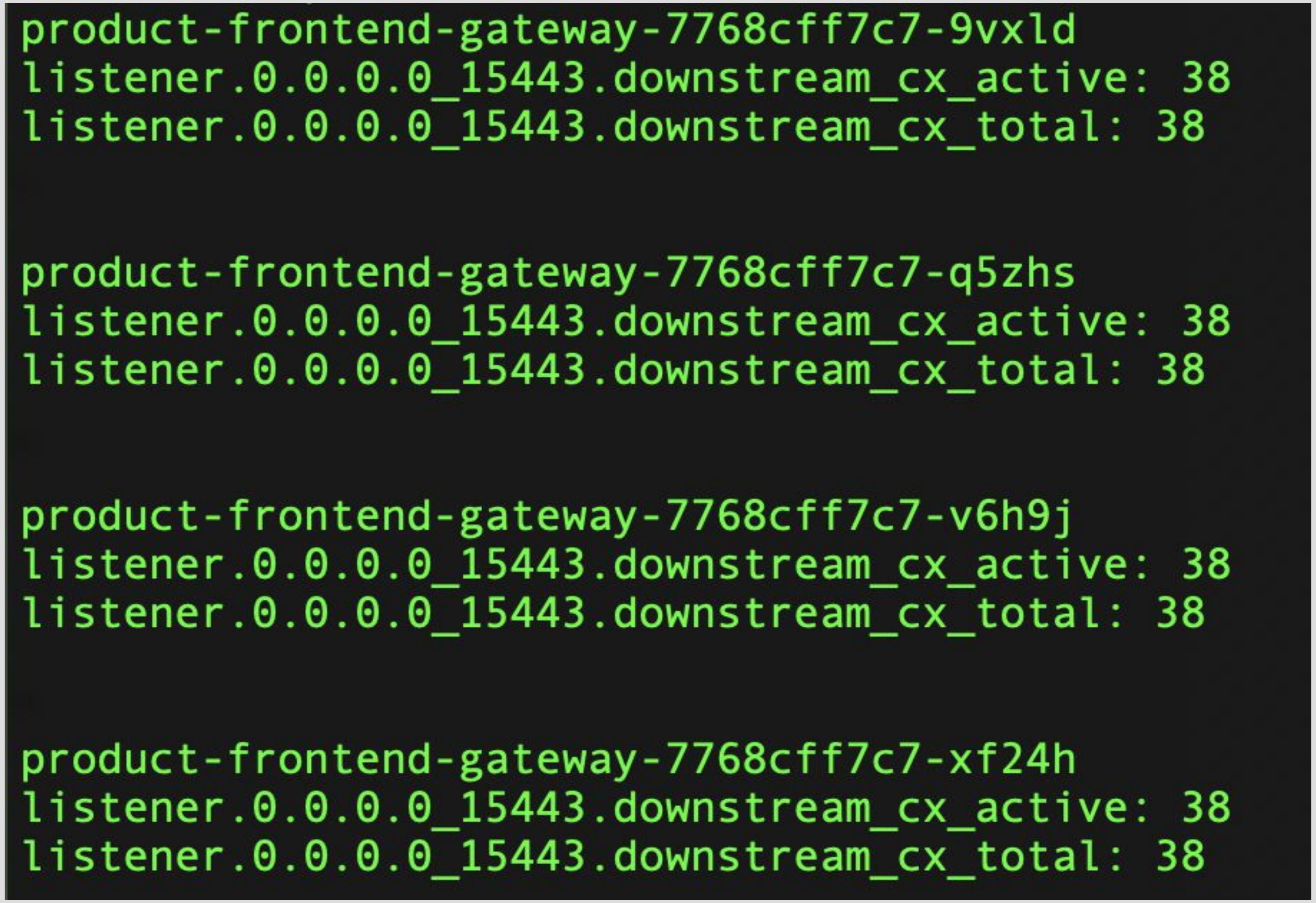
```
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: dr-productapp
  namespace: loadtest
spec:
  host: productapp.loadtest.svc
  trafficPolicy:
    connectionPool:
      http:
        maxRequestsPerConnection: 150
```

Improvise Cross Cluster Endpoints Discovery...

- *Results of NodePort approach on the right*
- *NLB with MaxRequestsPerConnection set would result in a slightly poorer distribution of connections at steady state, but overall negative impact on latency would be only marginal*

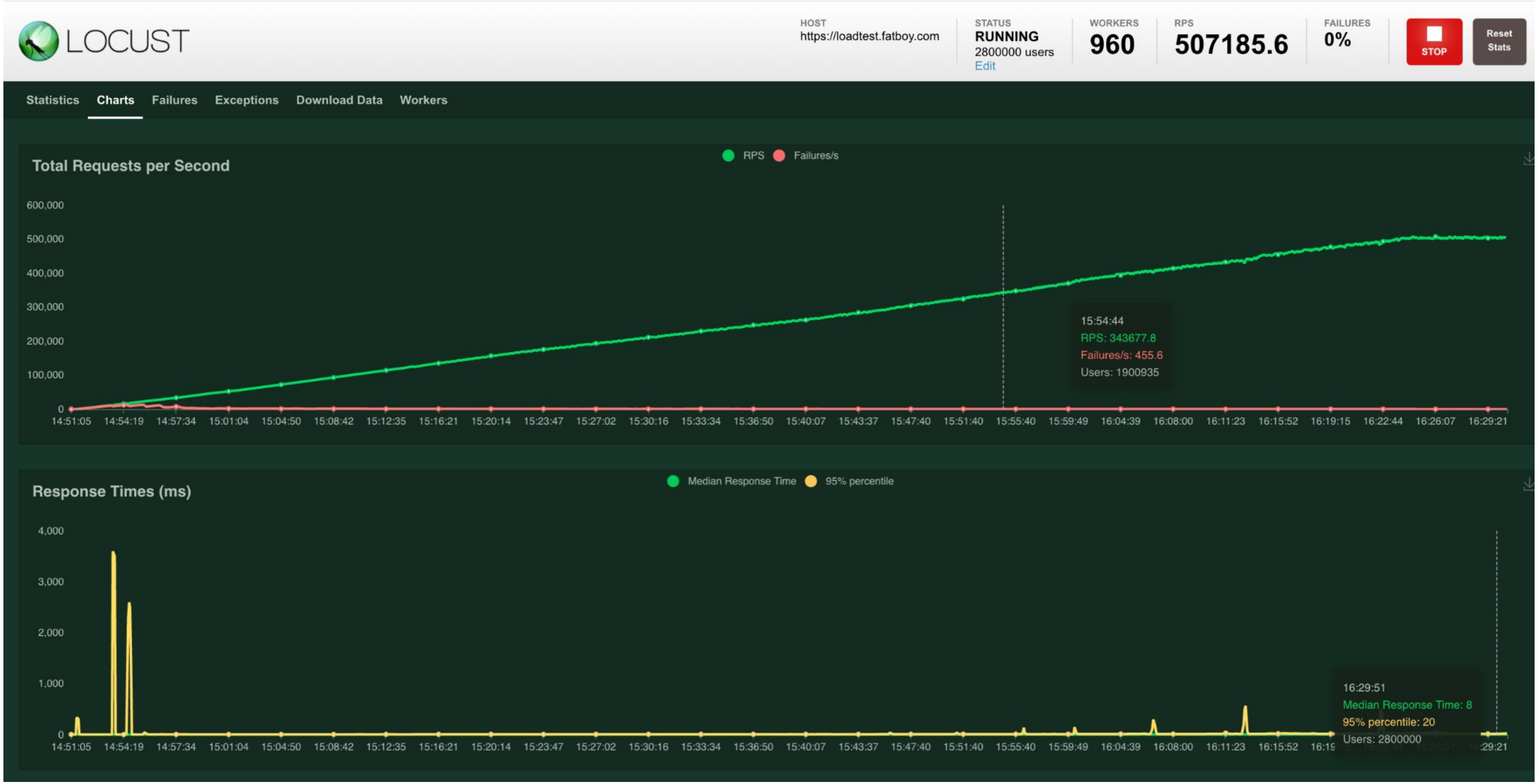


```
product-frontend-gateway-7768cff7c7-9vxld   678m          53Mi
product-frontend-gateway-7768cff7c7-q5zhs   663m          67Mi
product-frontend-gateway-7768cff7c7-v6h9j   659m          49Mi
product-frontend-gateway-7768cff7c7-xf24h   665m          204Mi
```



```
product-frontend-gateway-7768cff7c7-9vxld
listener.0.0.0.0_15443.downstream_cx_active: 38
listener.0.0.0.0_15443.downstream_cx_total: 38


product-frontend-gateway-7768cff7c7-q5zhs
listener.0.0.0.0_15443.downstream_cx_active: 38
listener.0.0.0.0_15443.downstream_cx_total: 38


product-frontend-gateway-7768cff7c7-v6h9j
listener.0.0.0.0_15443.downstream_cx_active: 38
listener.0.0.0.0_15443.downstream_cx_total: 38


product-frontend-gateway-7768cff7c7-xf24h
listener.0.0.0.0_15443.downstream_cx_active: 38
listener.0.0.0.0_15443.downstream_cx_total: 38
```

Result with 500,000 RPS



LOCUST

HOST
https://loadtest.fatboy.com

STATUS
RUNNING
2800000 users

WORKERS
960

RPS
507185.6

FAILURES
0%

tetrate

Thank You

Contact

@tetrateio | Tetrate | www.tetrate.io