

How to make your service more resilient in case of traffic spikes

Ivan Lemeshev

Ivan Lemeshev

Senior software engineer @ Unity



<https://www.linkedin.com/in/ivanlemeshev>



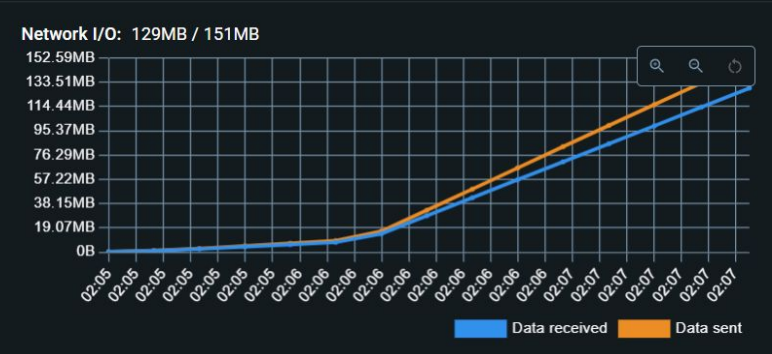
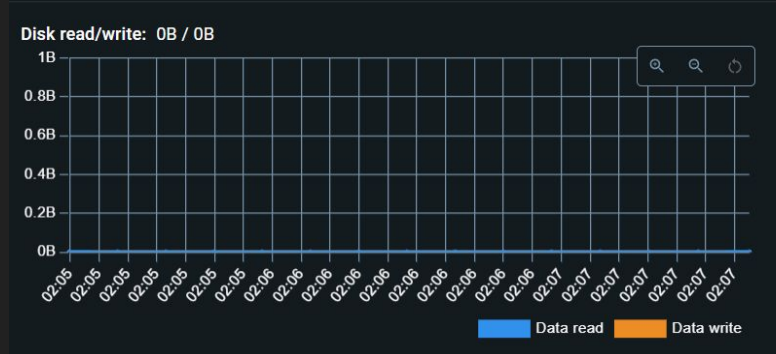
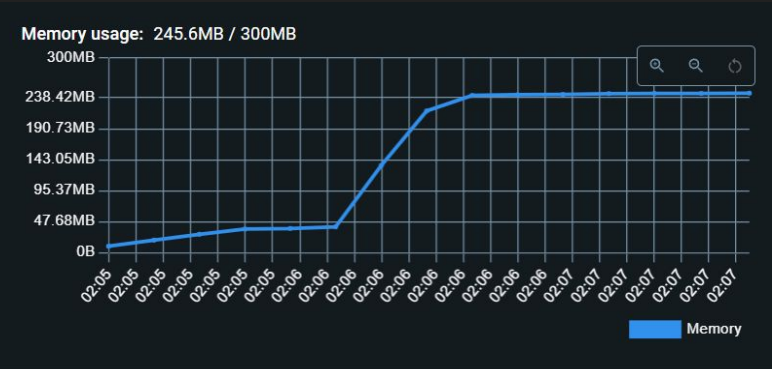
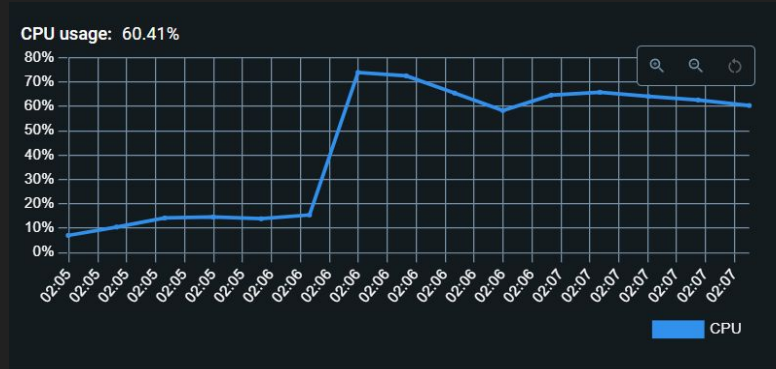
<https://github.com/ivanlemeshev>



Agenda

- Server overload
- Common causes lead to server overload
- Server overload on the example of a simple HTTP service
- Rate limiting
- Common rate-limiting algorithms
- Example of a rate limiter
- Load shedding
- Example of load shedding
- Conclusion

Server overload



Common causes of traffic bursts

- Predictable:
 - Scheduled events
 - Seasonal traffic
 - Time of day/week
- Unpredictable:
 - Viral events
 - Technical issues
 - Denial-of-service (DoS) attacks
 - Bot activity
 - External dependencies

Example of server overload

```
cmd/service/main.go

1 package main
2
3 import (
4     "fmt"
5     "log"
6     "net/http"
7     "strconv"
8
9     "github.com/ivanlemeshev/serveroverload/internal/password"
10 )
11
12 func main() {
13     http.HandleFunc("GET /password/{length}", handler)
14     log.Fatal(http.ListenAndServe(":8000", nil))
15 }
16
```

Example of server overload

```
cmd/service/main.go
17 func handler(w http.ResponseWriter, r *http.Request) {
18     length, err := strconv.Atoi(r.PathValue("length"))
19     if err != nil {
20         w.WriteHeader(http.StatusBadRequest)
21         fmt.Fprint(w, http.StatusText(http.StatusBadRequest))
22         return
23     }
24     pwd := password.Generate(length)
25     w.WriteHeader(http.StatusOK)
26     fmt.Fprint(w, pwd)
27 }
28
```

Example of server overload

```
cmd/service/Dockerfile

1 FROM golang:1.22 AS build-stage
2 RUN mkdir /code
3 COPY . /code
4 WORKDIR /code
5 RUN CGO_ENABLED=0 GOOS=linux go build -o /service ./cmd/service/main.go
6
7 FROM gcr.io/distroless/base:latest AS build-release-stage
8 WORKDIR /
9 COPY --from=build-stage /service /service
10 EXPOSE 8000
11 ENTRYPOINT ["/service"]
12
```



Example of server overload

```
$ docker build -t service:latest -f ./cmd/service/Dockerfile .
$ docker run -d --rm --name service \
  --cpus="1" \
  --memory="300m" \
  --memory-swap="300m" \
  -p 8000:8000 service:latest
```

Example of server overload

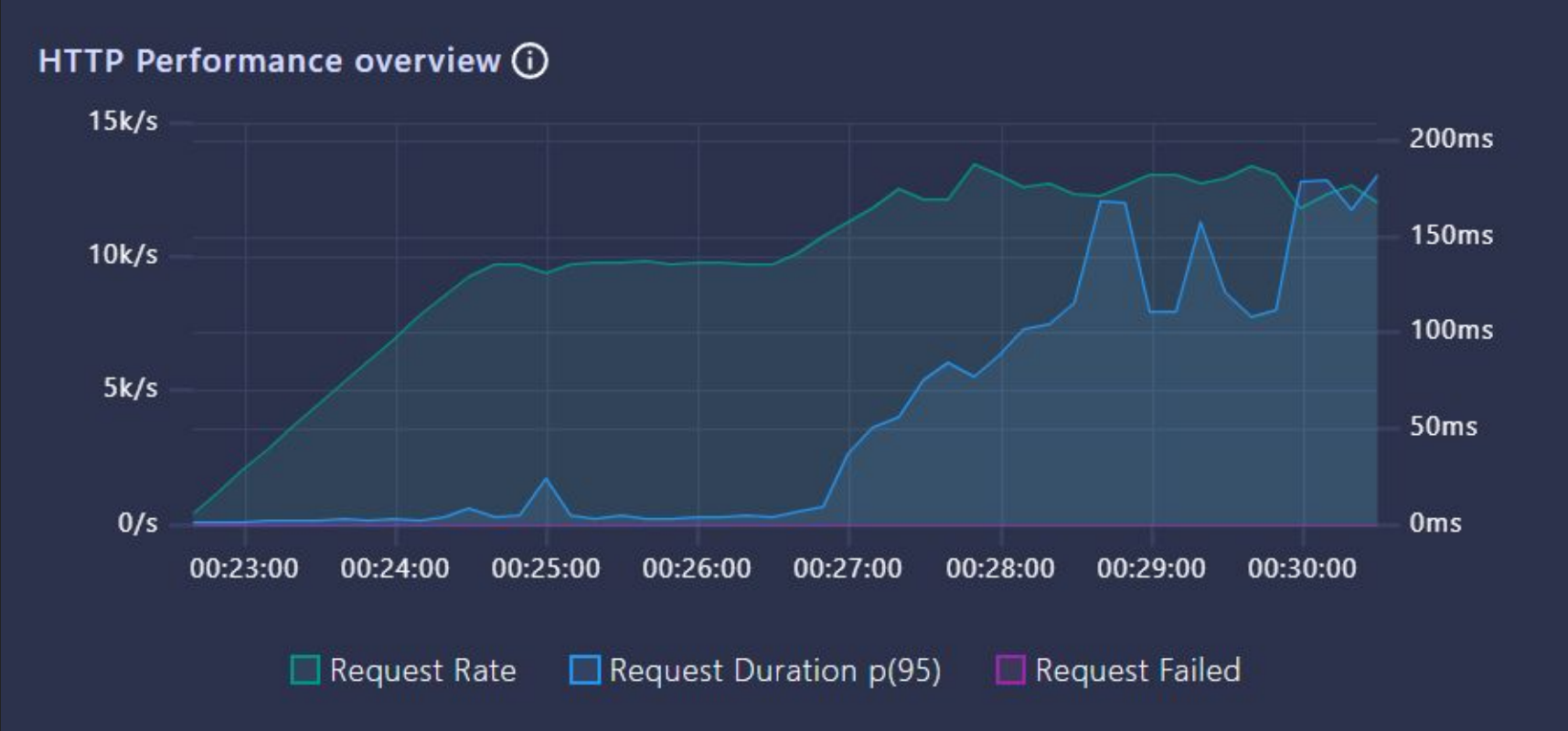
```
JS script.js
1 import http from 'k6/http';
2 import { sleep } from 'k6';
3
4 export const options = {
5   stages: [
6     { duration: '2m', target: 1000 },
7     { duration: '2m', target: 1000 },
8     { duration: '2m', target: 2000 },
9     { duration: '2m', target: 2000 },
10  ],
11 };
12
13 export default function () {
14   http.get('http://localhost:8000/password/32');
15   sleep(0.1); // 100ms
16 }
17
```

Example of server overload

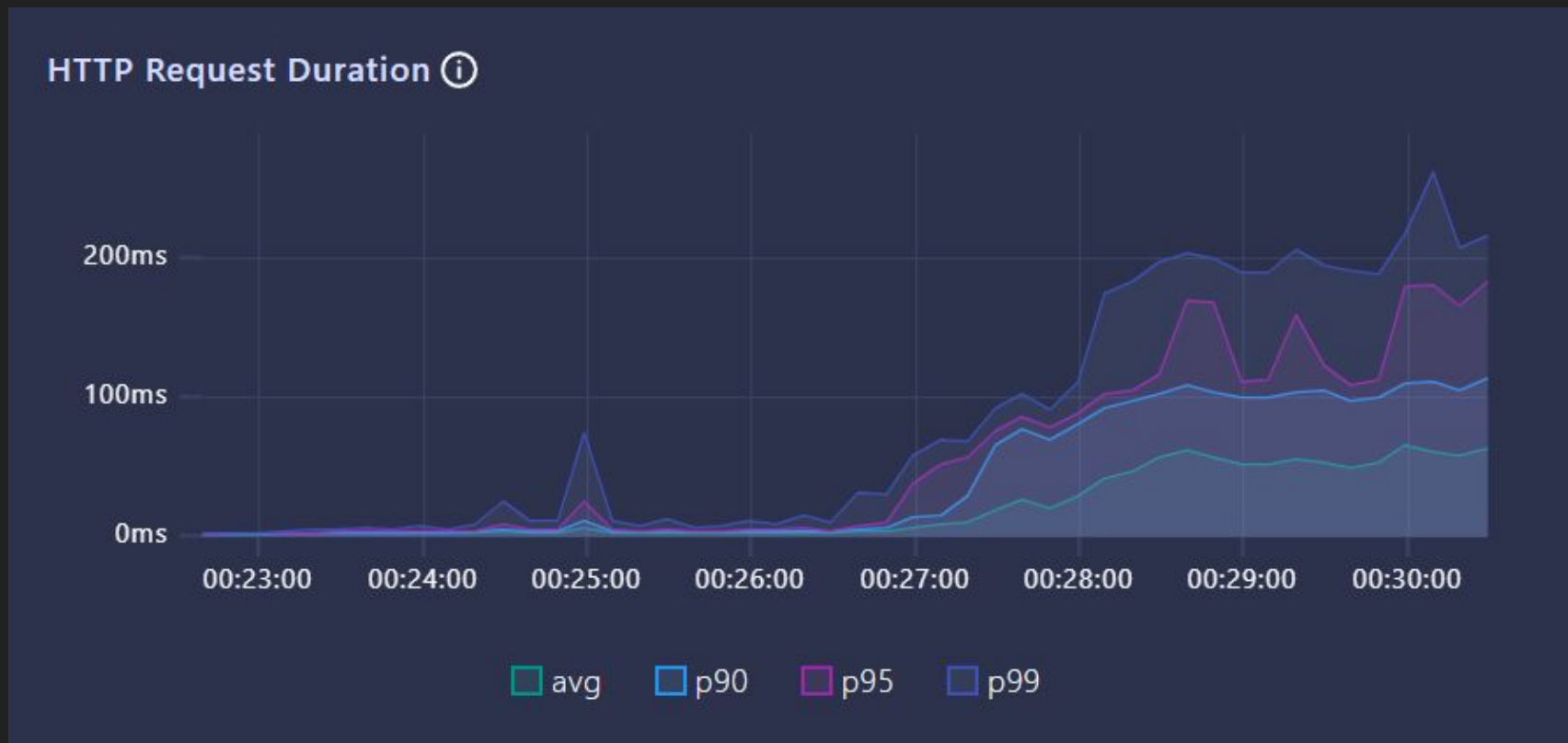
A terminal window with a dark blue background and a title bar containing three colored circles (red, yellow, green). The terminal displays a single line of text: "\$ k6 run --out web-dashboard script.js".

```
$ k6 run --out web-dashboard script.js
```

Example of server overload



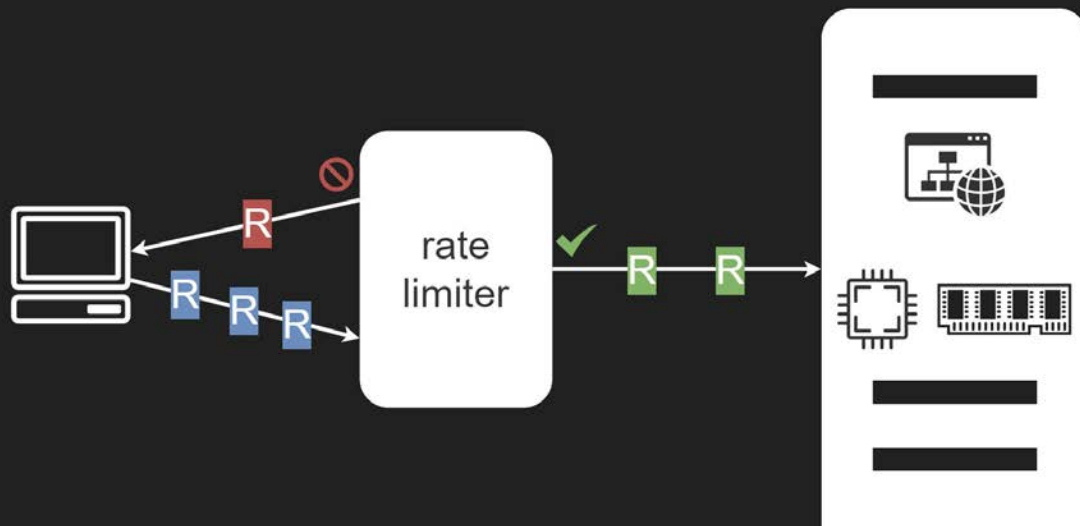
Example of server overload



Agenda

- Server overload
- Common causes lead to server overload
- Server overload on the example of a simple HTTP service
- Rate limiting
- Common rate-limiting algorithms
- Example of a rate limiter
- Load shedding
- Example of load shedding
- Conclusion

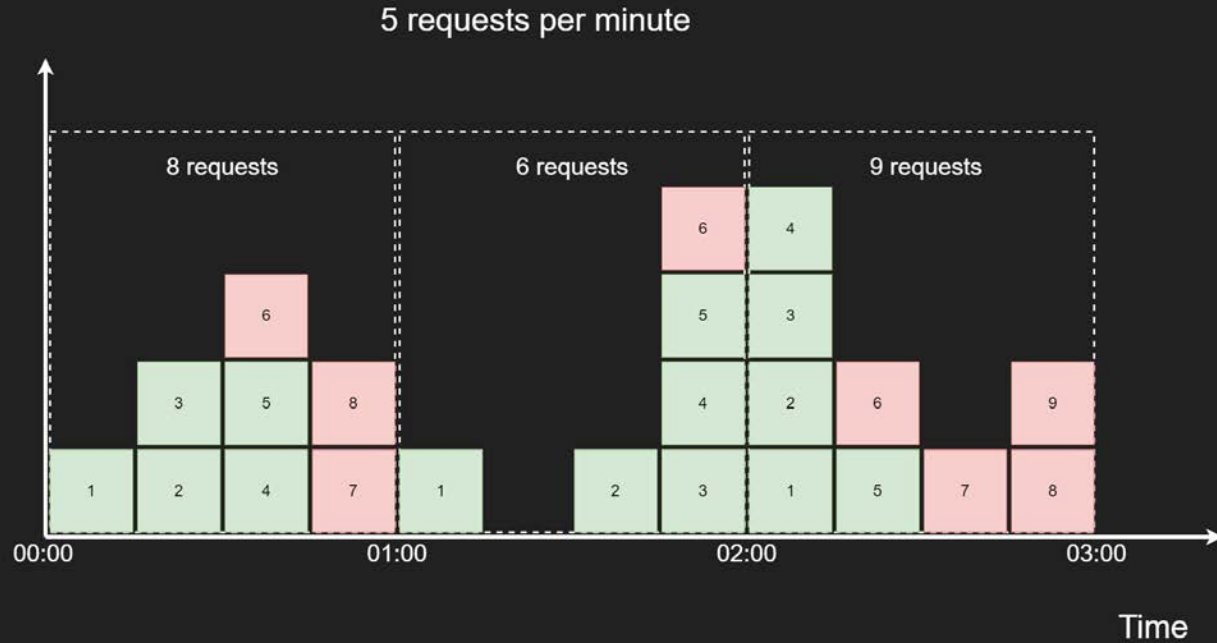
Rate limiting



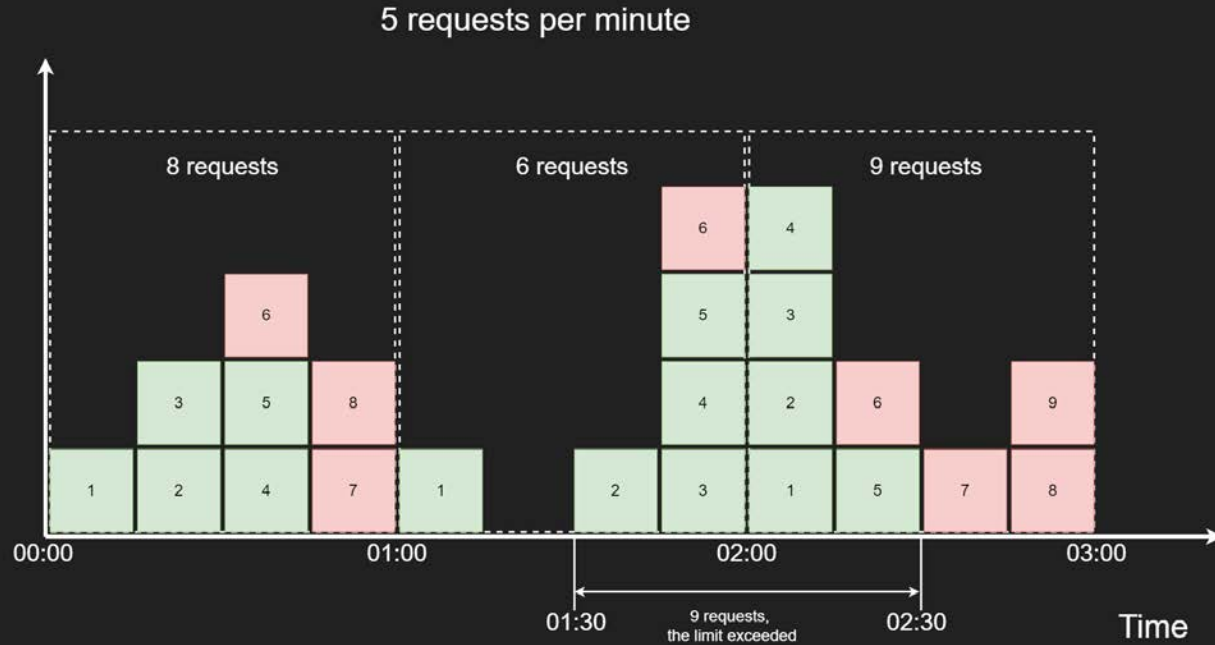
Common rate-limiting algorithms

- Fixed window counter
- Sliding window log
- Sliding window counter
- Token bucket
- Leaky bucket

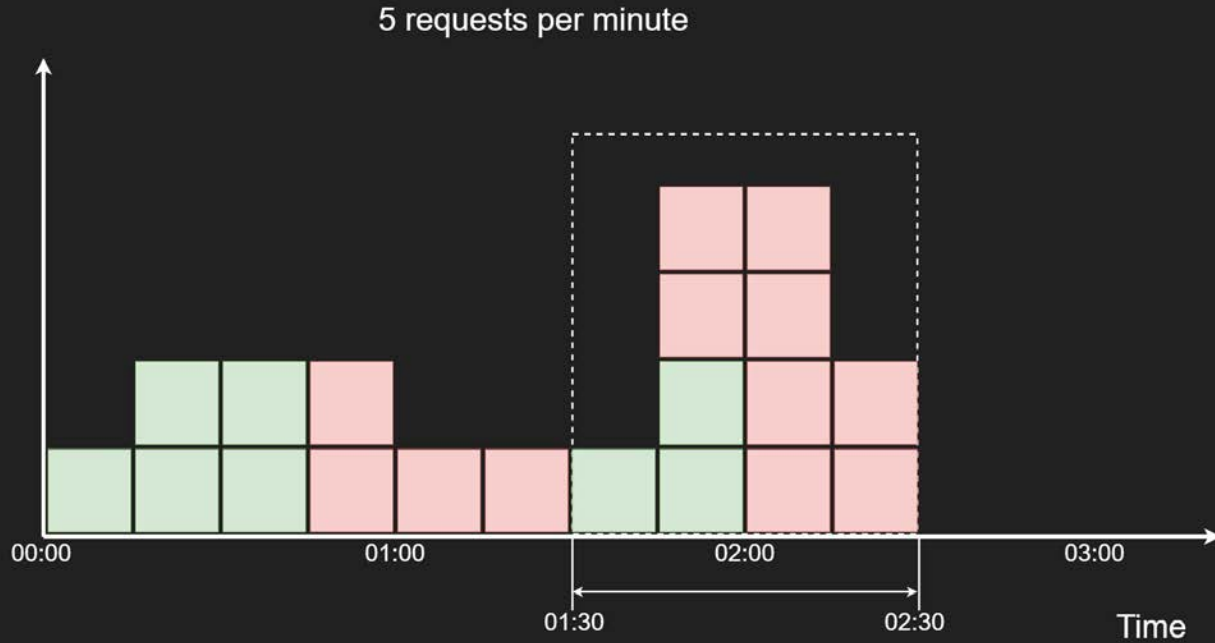
Fixed window counter algorithm



Fixed window counter algorithm

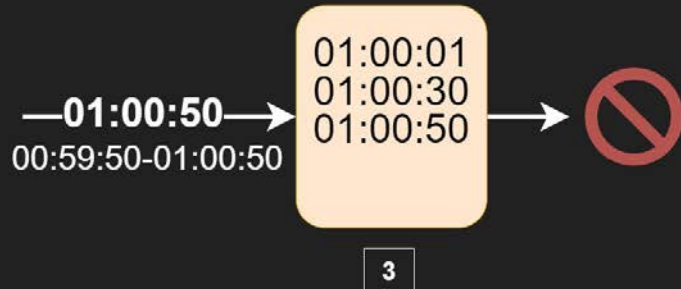
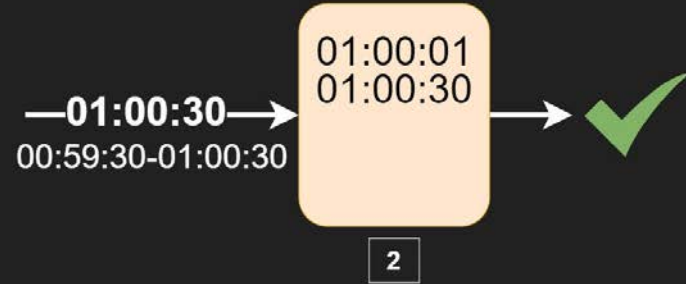


Sliding window log algorithm

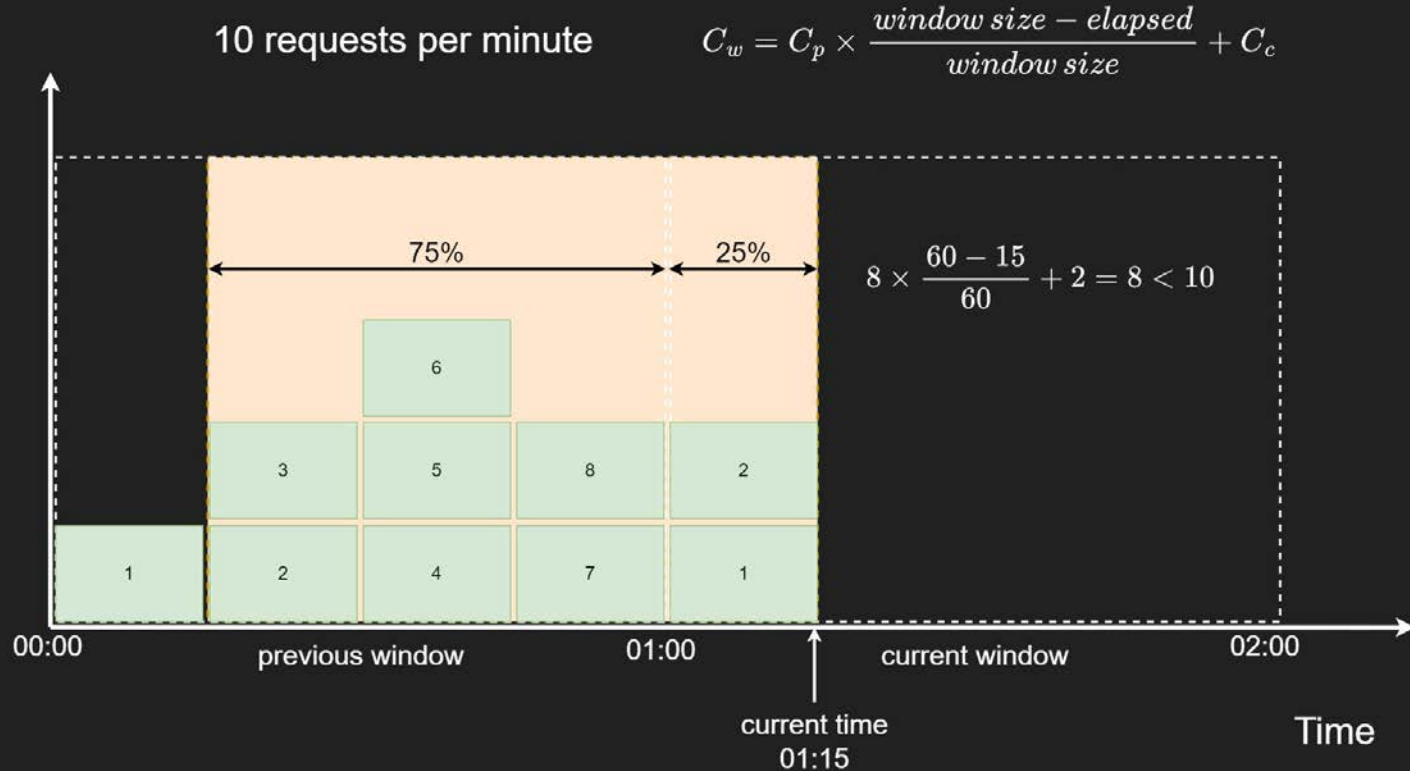


Sliding window log algorithm

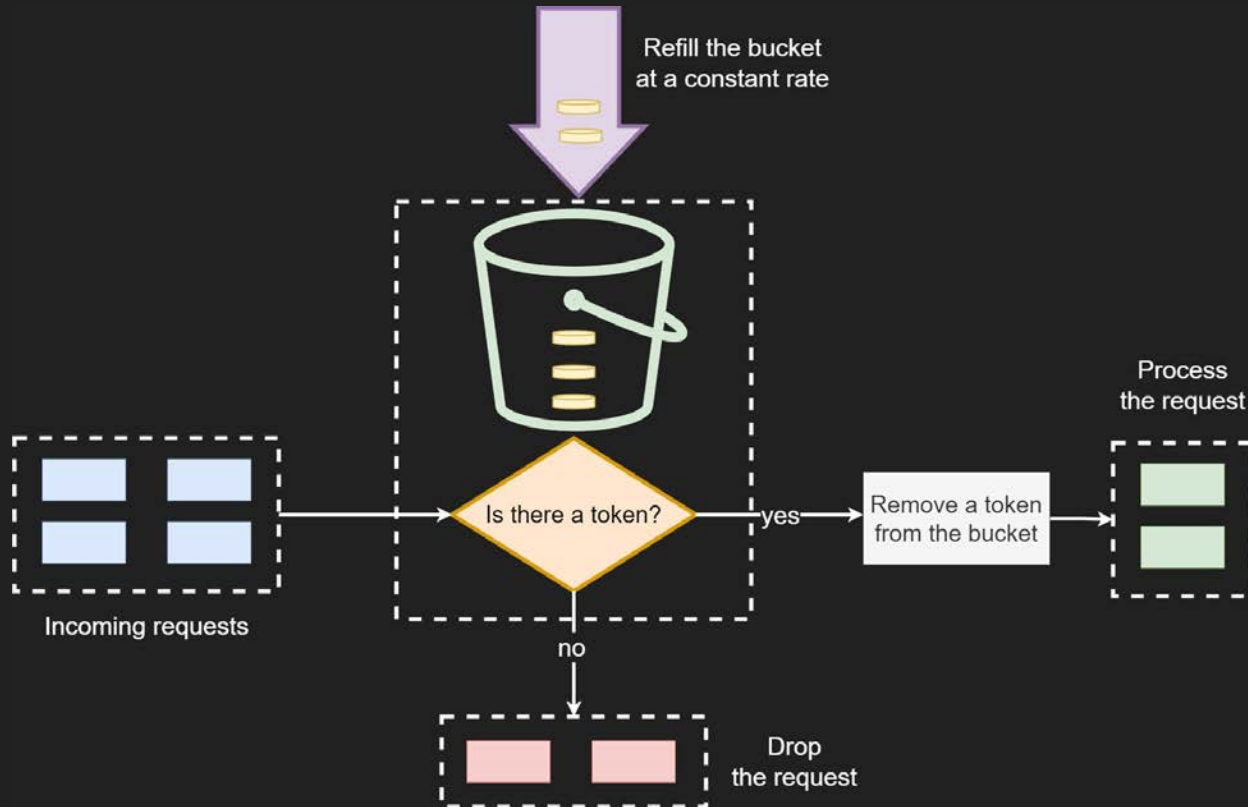
2 request per minute



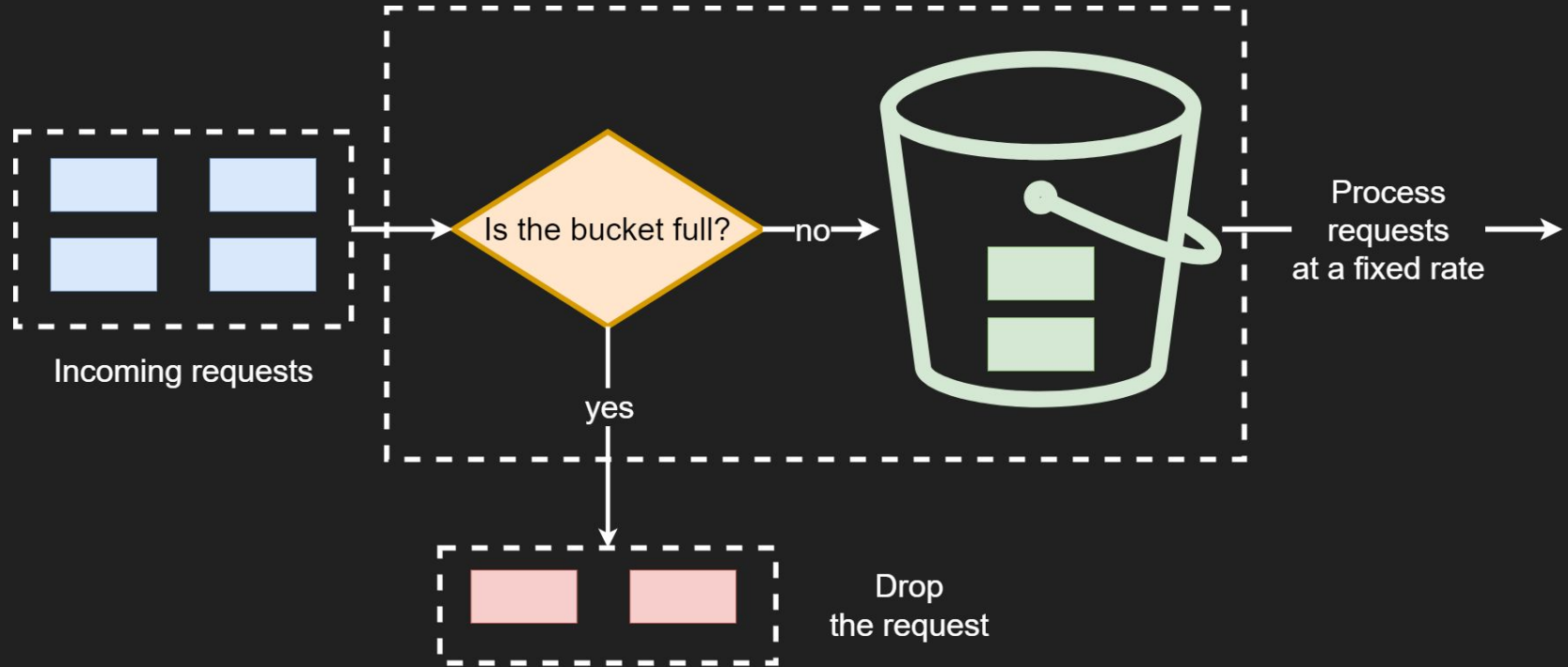
Sliding window counter algorithm



Token bucket algorithm



Leaky bucket algorithm



Common rate-limiting algorithms

- ✓ Fixed window counter
- ✓ Sliding window log
- ✓ Sliding window counter
- ✓ Token bucket
- ✓ Leaky bucket

Example of token bucket algorithm

internal/ratelimiter/token_bucket.go

```
1 package ratelimiter
2
3 import (
4     "math"
5     "sync"
6     "time"
7 )
8
9 type bucket struct {
10     currentTokens float64
11     lastRefillTime time.Time
12 }
13
```

Example of token bucket algorithm

```
internal/ratelimiter/token_bucket.go

14 type TokenBucket struct {
15     mu          sync.Mutex
16     buckets    map[string]*bucket
17     bucketSize float64
18     refillRate float64
19 }
20
21 func NewTokenBucket(bucketSize, refillRate float64) *TokenBucket {
22     return &TokenBucket{
23         bucketSize: bucketSize,
24         refillRate: refillRate,
25         buckets:    make(map[string]*bucket),
26     }
27 }
28
```

Example of token bucket algorithm

```
internal/ratelimiter/token_bucket.go

29 func (rl *TokenBucket) IsAllowed(key string) bool {
30     rl.mu.Lock()
31     defer rl.mu.Unlock()
32     if _, ok := rl.buckets[key]; !ok {
33         rl.buckets[key] = &bucket{
34             currentTokens: rl.bucketSize - 1,
35             lastRefillTime: time.Now(),
36         }
37         return true
38     }
39     rl.refill(key)
40     if int(rl.buckets[key].currentTokens) > 0 {
41         rl.buckets[key].currentTokens--
42         return true
43     }
44     return false
45 }
```

Example of token bucket algorithm



internal/ratelimiter/token_bucket.go

```
46
47 func (rl *TokenBucket) refill(key string) {
48     now := time.Now()
49     duration := now.Sub(rl.buckets[key].lastRefillTime)
50     tokensToAdd := rl.refillRate * duration.Seconds()
51     rl.buckets[key].currentTokens = math.Min(rl.bucketSize,
52         rl.buckets[key].currentTokens+tokensToAdd)
53     rl.buckets[key].lastRefillTime = now
54 }
55
```

Example of token bucket algorithm

internal/middleware/rate_limiting.go

```
1 package middleware
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 type RateLimiter interface {
9     IsAllowed(string) bool
10 }
11
```

Example of token bucket algorithm

```
internal/middleware/rate_limiting.go

12 func RateLimiting(rl RateLimiter, f http.HandlerFunc) http.HandlerFunc {
13     return func(w http.ResponseWriter, r *http.Request) {
14         if !rl.IsAllowed(r.RemoteAddr) {
15             w.WriteHeader(http.StatusTooManyRequests)
16             fmt.Fprint(w, http.StatusText(http.StatusTooManyRequests))
17             return
18         }
19         f(w, r)
20     }
21 }
22
```

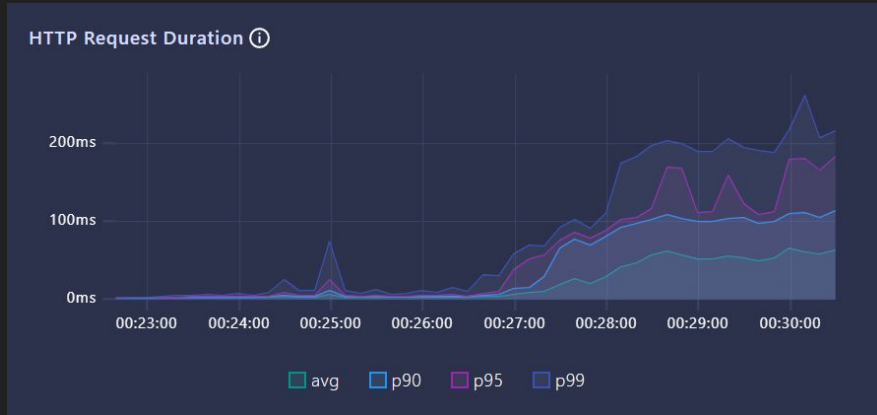
Example of token bucket algorithm

```
cmd/service/main.go

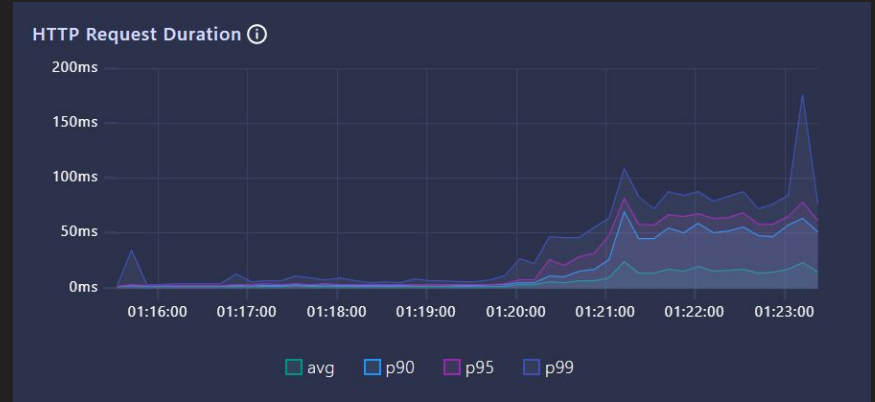
13
14 func main() {
15     rl := ratelimiter.NewTokenBucket(1, 1)
16     http.HandleFunc("GET /password/{length}", middleware.RateLimiting(rl, handler))
17     log.Fatal(http.ListenAndServe(":8000", nil))
18 }
19
```

Comparison of results

The service without rate limiter



The service with rate limiter



Rate limiter packages

- Fixed window counter
 - <https://github.com/mennanov/limiters>
 - <https://github.com/abo/rerate>
- Sliding window counter
 - <https://github.com/mennanov/limiters>
 - <https://github.com/RussellLuo/slidingwindow>
- Token bucket
 - <https://github.com/mennanov/limiters>
 - <https://github.com/juju/ratelimit>
 - <https://pkg.go.dev/golang.org/x/time/rate>
 - <https://github.com/didip/tollbooth>
- Leaky bucket
 - <https://github.com/uber-go/ratelimit>
 - <https://github.com/mennanov/limiters>

Agenda

- Server overload
- Common reasons lead to server overload
- Server overload on the example of a simple HTTP service
- Rate limiting
- Common rate-limiting algorithms
- Example of a rate limiter
- Load shedding
- Example of load shedding
- Conclusion

Load shedding



Example of loading shedding

internal/overloaddetector/overload_detector.go

```
1 package overloaddetector
2
3 import (
4     "context"
5     "sync/atomic"
6     "time"
7 )
8
9 type OverloadDetector struct {
10     checkInterval time.Duration
11     overloadFactor time.Duration
12     isOverloaded  atomic.Bool
13 }
14
```

Example of loading shedding

```
internal/overloaddetector/overload_detector.go

15 func New(ctx context.Context, checkInterval, overloadFactor time.Duration) *OverloadDetector {
16     od := OverloadDetector{
17         checkInterval: checkInterval,
18         overloadFactor: overloadFactor,
19     }
20     go od.run(ctx)
21     return &od
22 }
23
24 func (od *OverloadDetector) IsOverloaded() bool {
25     return od.isOverloaded.Load()
26 }
27
```

Example of loading shedding

```
internal/overloaddetector/overload_detector.go

28 func (od *OverloadDetector) run(ctx context.Context) {
29     ticker := time.NewTicker(od.checkInterval)
30     defer ticker.Stop()
31     checkTime := time.Now()
32     for {
33         select {
34             case <-ctx.Done():
35                 return
36             case <-ticker.C:
37                 if time.Since(checkTime) > od.overloadFactor {
38                     od.isOverloaded.Store(true)
39                 } else {
40                     od.isOverloaded.Store(false)
41                 }
42                 checkTime = time.Now()
43         }
44     }
45 }
```

Example of loading shedding

internal/middleware/overloa_detecting.go

```
1 package middleware
2
3 import (
4     "fmt"
5     "net/http"
6 )
7
8 type OverloadDetector interface {
9     IsOverloaded() bool
10 }
11
```

Example of loading shedding

```
internal/middleware/overloa_detecting.go

12 func OverloadDetecting(od OverloadDetector, f http.HandlerFunc) http.HandlerFunc {
13     return func(w http.ResponseWriter, r *http.Request) {
14         if od.IsOverloaded() {
15             w.WriteHeader(http.StatusServiceUnavailable)
16             fmt.Fprint(w, http.StatusText(http.StatusServiceUnavailable))
17             return
18         }
19         f(w, r)
20     }
21 }
22
```

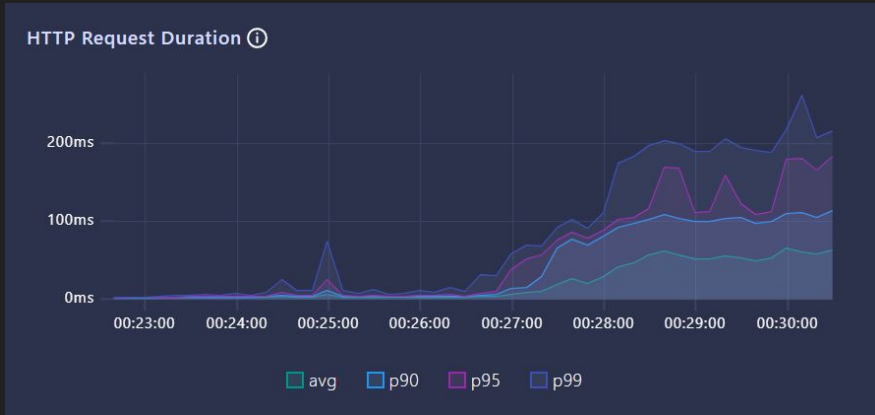

Example of loading shedding

```
cmd/service/main.go

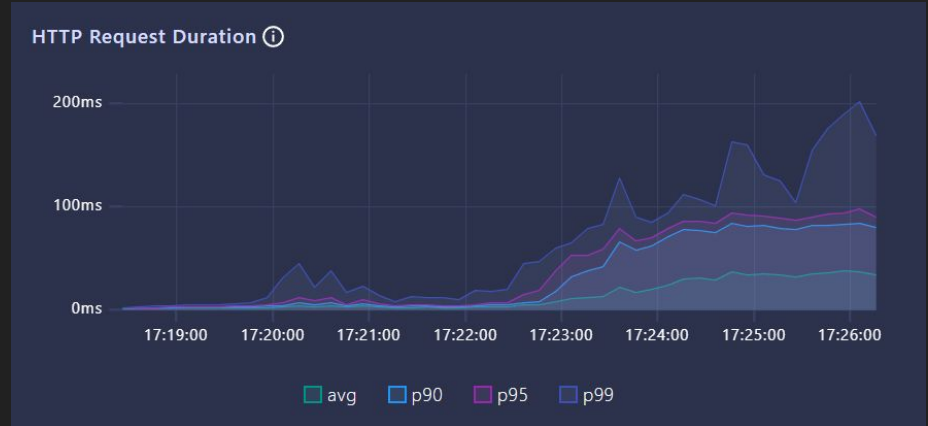
15
16 func main() {
17     ctx, cancel := context.WithCancel(context.Background())
18     defer cancel()
19     od := overloaddetector.New(ctx, 10*time.Microsecond, 11*time.Millisecond)
20     http.HandleFunc("GET /password/{length}", middleware.OverloadDetecting(od, handler))
21     log.Fatal(http.ListenAndServe(":8000", nil))
22 }
23
```

Comparison of results

The service without load shedding



The service with load shedding



To read

- Using load shedding to avoid overload from Senior Principal Engineer at Amazon
- <https://aws.amazon.com/builders-library/using-load-shedding-to-avoid-overload/>



Agenda

- ✓ Server overload
- ✓ Common reasons lead to server overload
- ✓ Server overload on the example of a simple HTTP service
- ✓ Rate limiting
- ✓ Common rate-limiting algorithms
- ✓ Example of a rate limiter
- ✓ Load shedding
- ✓ Example of load shedding
- ✓ Conclusion

Thank you!