HTTP Rate-limiting

@VojtechVitek
backend developer at @Pressly

Rate Limiting vs. Throttling

- **Rate limiting** defines how many requests/packets can be accepted within a defined period of time. Any requests beyond that period are rejected.

- **Throttling** works a lot like rate limiting, except that requests that exceed the limits are not rejected but delayed and put in a queue.

HTTP 429

- RFC 6585 - Additional HTTP Status Codes
- Status Code 429 - Too Many Requests
- Recommended: Retry-After header

What do we want to limit?

- Rate-limit requests per IP
  - Attacker might have **many** IPv6 addresses, not easy
  - We can leave this job to CDN / DNS / IAAS provider
- Business logic rate-limits
  - requests per OAuth token (per endpoint)
  - requests per Session.User.ID etc.
  - login attempts (failures) per username/email

Twitter API

- Per-user and per-application rate-limits

  15 calls every 15 minutes
  180 calls every 15 minutes (search etc.)

- **X-Rate-Limit-Limit**: the rate limit ceiling for that given request

- **X-Rate-Limit-Remaining**: the number of requests left for the 15 minute window

- **X-Rate-Limit-Reset**: the remaining window before the rate limit resets in UTC epoch seconds

Rate Limits: Chart

Title	Resource family	Requests / 15-min window (user auth)	Requests / 15-min window (app auth)
GET application/rate_limit_status	application	180	180
GET favorites/list	favorites	15	15
GET followers/ids	followers	15	15
GET followers/list	followers	15	30
GET friends/ids	friends	15	15
GET friends/list	friends	15	30
GET friendships/show	friendships	180	15
GET help/configuration	help	15	15
GET help/languages	help	15	15
GET help/privacy	help	15	15
GET help/tos	help	15	15
GET lists/list	lists	15	15
GET lists/members	lists	180	15
GET lists/members/show	lists	15	15

Github API

- 5000 requests per hour (auth, OAuth)
  60 requests per hour (non-auth)

- **X-RateLimit-Limit**
  (Twitter: X-Rate-Limit-Limit)

- **X-RateLimit-Remaining**
  (Twitter: X-Rate-Limit-Remaining)

- **X-RateLimit-Reset**
  (Twitter: X-Rate-Limit-Reset)

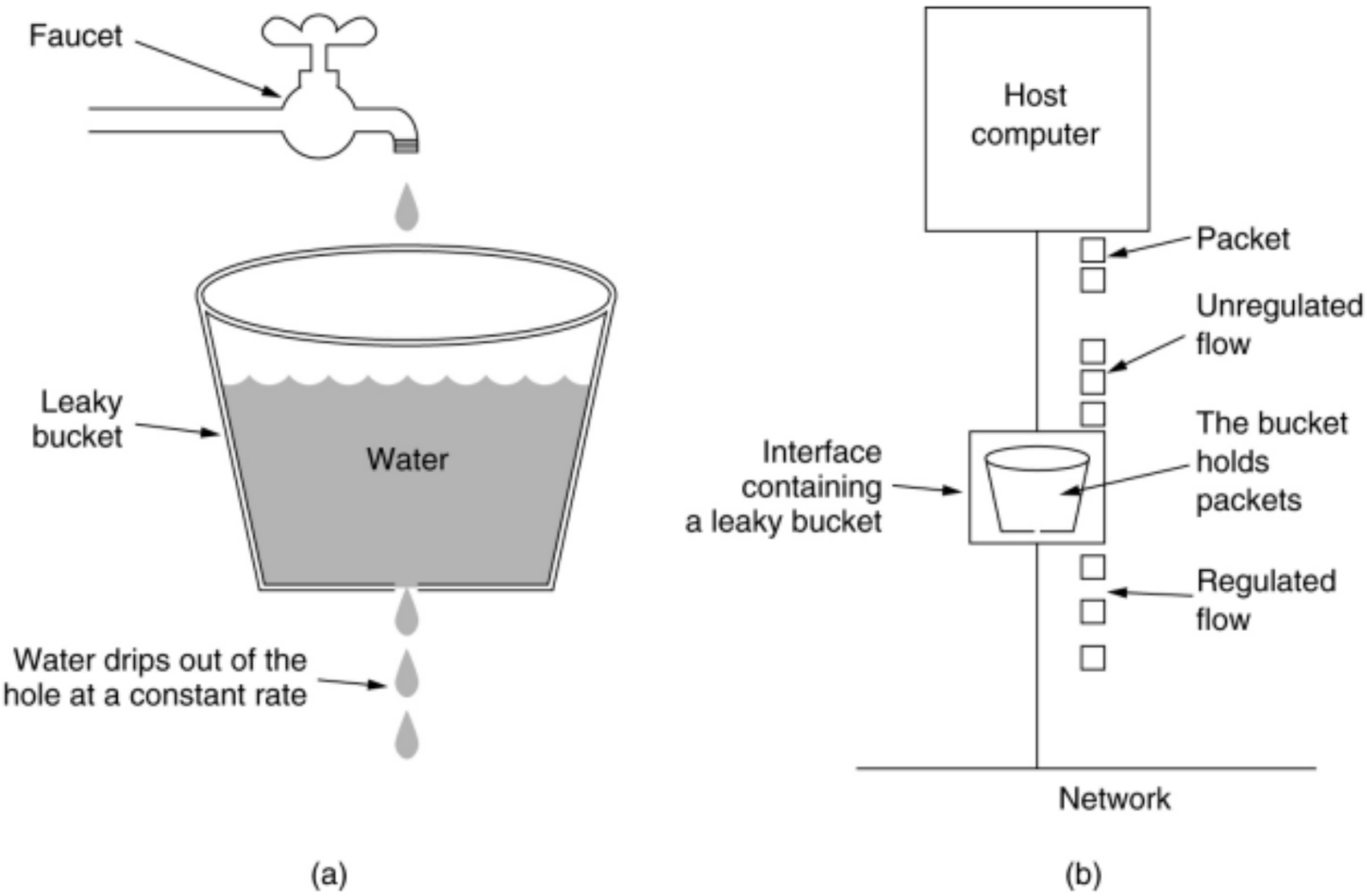
Leaky bucket - golang wiki

To limit the rate of operations per unit time, use a time.Ticker. This works well for rates up to tens of operations per second. For higher rates, prefer a token bucket rate limiter such as golang.org/x/time/rate.Limiter (also search godoc.org for rate limit).

```
import "time"

rate := time.Second / 10
throttle := time.Tick(rate)
for req := range requests {
  <-throttle  // rate limit our Service.Method RPCs
  go client.Call("Service.Method", req, ...)
}
```

The Leaky Bucket Algorithm



(a) A leaky bucket with water. (b) a leaky bucket with packets.

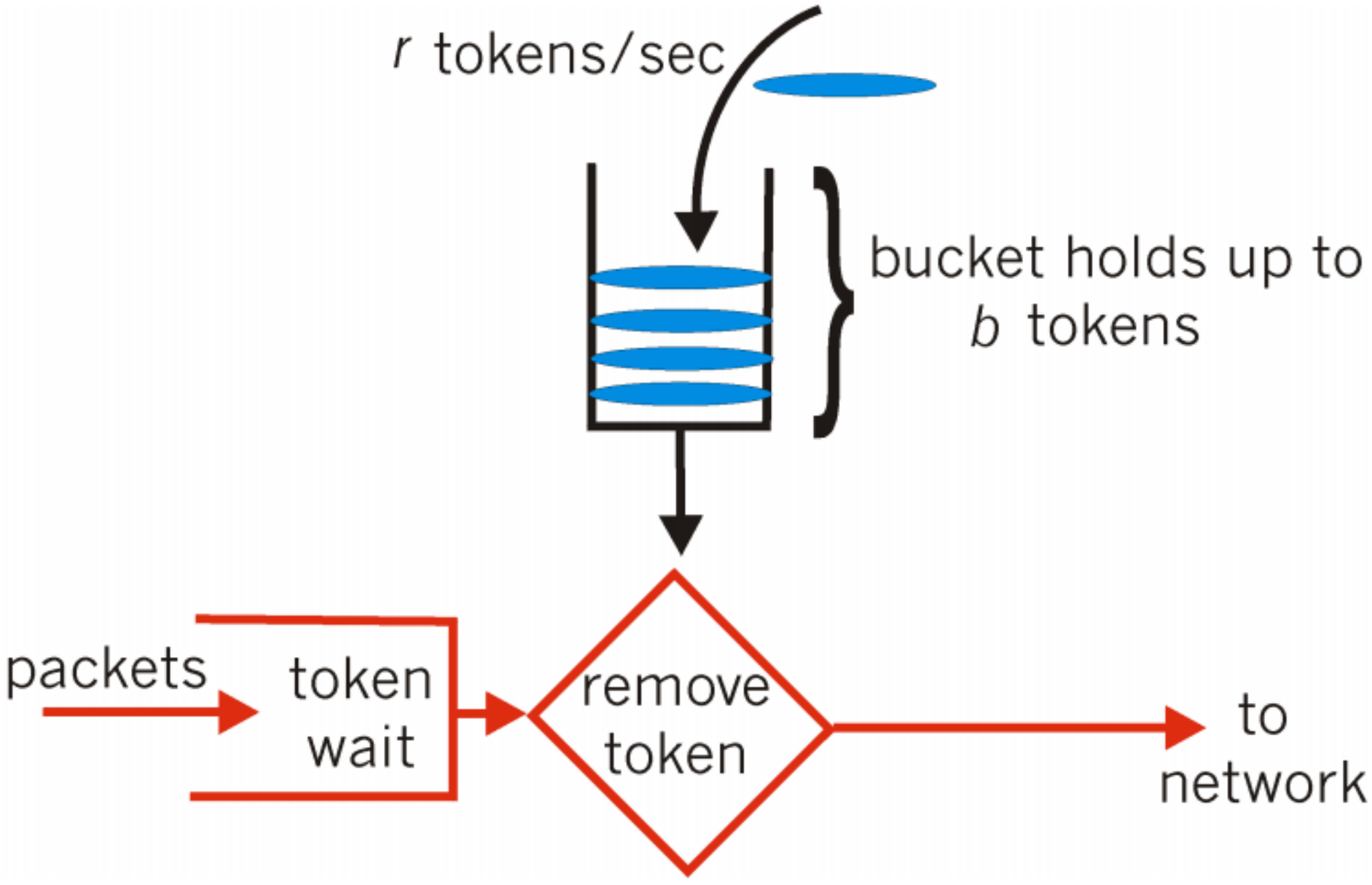
Token bucket - golang wiki

To allow some bursts, add a buffer to the throttle:

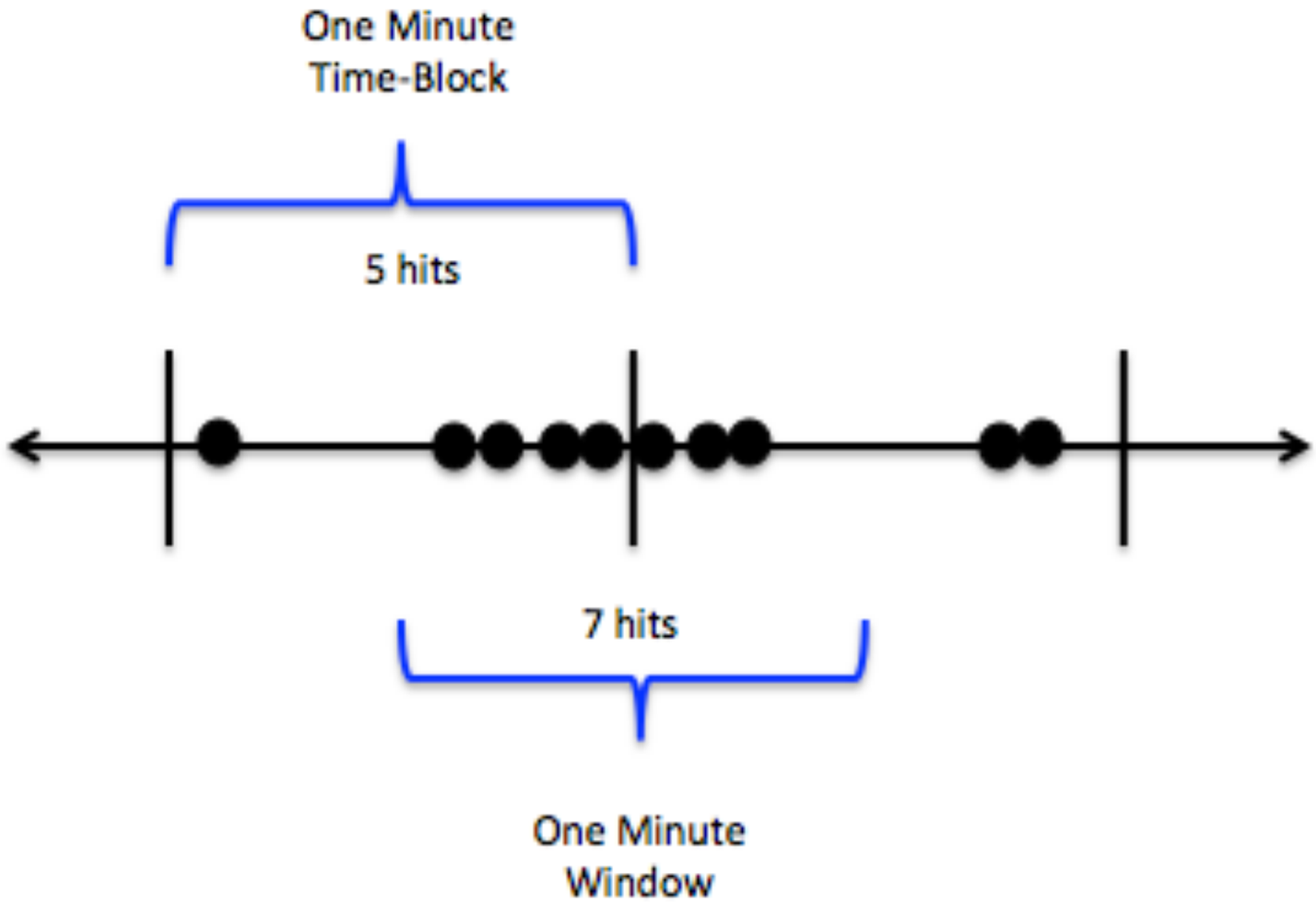
```
import "time"

rate := time.Second / 10
burstLimit := 100
tick := time.NewTicker(rate)
defer tick.Stop()
throttle := make(chan time.Time, burstLimit)
go func() {
  for t := range tick.C {
    select {
      case throttle <- t:
      default:
    }
  }  // exits after tick.Stop()
}()
for req := range requests {
  <-throttle  // rate limit our Service.Method RPCs
  go client.Call("Service.Method", req, ...)
}
```

Token Bucket Regulator (Shaper)



Fixed vs. Moving window



* Moving == Sliding

```go
// net/http
func (srv *Server) Serve(l net.Listener) error {
        defer l.Close()
        // ...

        baseCtx := context.Background()
        ctx := context.WithValue(baseCtx, ServerContextKey, srv)
        ctx = context.WithValue(ctx, LocalAddrContextKey, l.Addr())
        for {
                rw, e := l.Accept()
                if e != nil {
                        // ...
                        return e
                }
                tempDelay = 0
                c := srv.newConn(rw)
                c.setState(c.rwc, StateNew) // before Serve can return

                go c.serve(ctx) // <= Each request is already in a goroutine!
        }
}
```

Open-source pkgs

- github.com/juju/ratelimit
  - Token bucket algorithm, in-memory only
  - Simple pkg, good impl. - but not a HTTP middleware
    - `func NewBucketWithRate(rate float64, capacity int64) *Bucket`
    - `func (tb *Bucket) TakeAvailable(count int64) int64`
- No concept of storage for request keys
- Juju, Kubernetes, Go-kit, Rancher DNS

Open-source pkgs

- github.com/didip/tollbooth
  - HTTP middleware, github.com/juju/ratelimit under the hood
  - Pre-defined keys (remote IP, path, methods, custom headers)

```
limiter := tollbooth.NewLimiter(1, time.Second)

limiter.IPLookups = []string{"RemoteAddr", "X-Forwarded-For", "X-Real-IP"}

limiter.Methods = []string{"GET", "POST"}

limiter.Headers = map[string][]string{"X-Access-Token": []string{"abc123", "xyz098"}}

func LimitHandler(limiter *config.Limiter, next http.Handler) http.Handler
```

Open-source pkgs

- github.com/VojtechVitek/ratelimit (Proof-of-concept, WIP)
  - HTTP middleware to rate-limit Requests, Download/ Upload speed etc., Throttler
  - Buckets referenced by string keys, generated on-the-fly
    - `func Key(r *http.Request) string`
  - Storage independent interface for taking tokens
    - Token bucket algorithm (fixed or sliding window)
    - ie. Redis with In-Memory backup

DEMO

- github.com/VojtechVitek/ratelimit

Pressly is hiring!

Pressly is knowledge sharing platform for teams and communities.

www.golang.to