Rate Limiting

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 pkg.go.dev for rate limit).

```
import "time"

const rateLimit = time.Second / 10  // 10 calls per second

// Client is an interface that calls something with a payload.
type Client interface {
  Call(*Payload)
}

// Payload is some payload a Client would send in a call.
type Payload struct {}

// RateLimitCall rate limits client calls with the payloads.
func RateLimitCall(client Client, payloads []*Payload) {
  throttle := time.Tick(rateLimit)

  for _, payload := range payloads {
    <-throttle  // rate limit our client calls
    go client.Call(payload)
  }
}
```

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

```
import "time"

const rateLimit = time.Second / 10  // 10 calls per second

// Client is an interface that calls something with a payload.
type Client interface {
  Call(*Payload)
}

// Payload is some payload a Client would send in a call.
type Payload struct {}

// BurstRateLimitCall allows burst rate limiting client calls with the
// payloads.
func BurstRateLimitCall(ctx context.Context, client Client, payloads []*Payload,
burstLimit int) {
  throttle := make(chan time.Time, burstLimit)

  ctx, cancel := context.WithCancel(ctx)
  defer cancel()
```

```
  go func() {
    ticker := time.NewTicker(rateLimit)
    defer ticker.Stop()
    for t := range ticker.C {
        select {
        case throttle <- t:
        case <-ctx.Done():
            return // exit goroutine when surrounding function returns
        }
    }
  }()

  for _, payload := range payloads {
    <-throttle  // rate limit our client calls
    go client.Call(payload)
  }
}
```