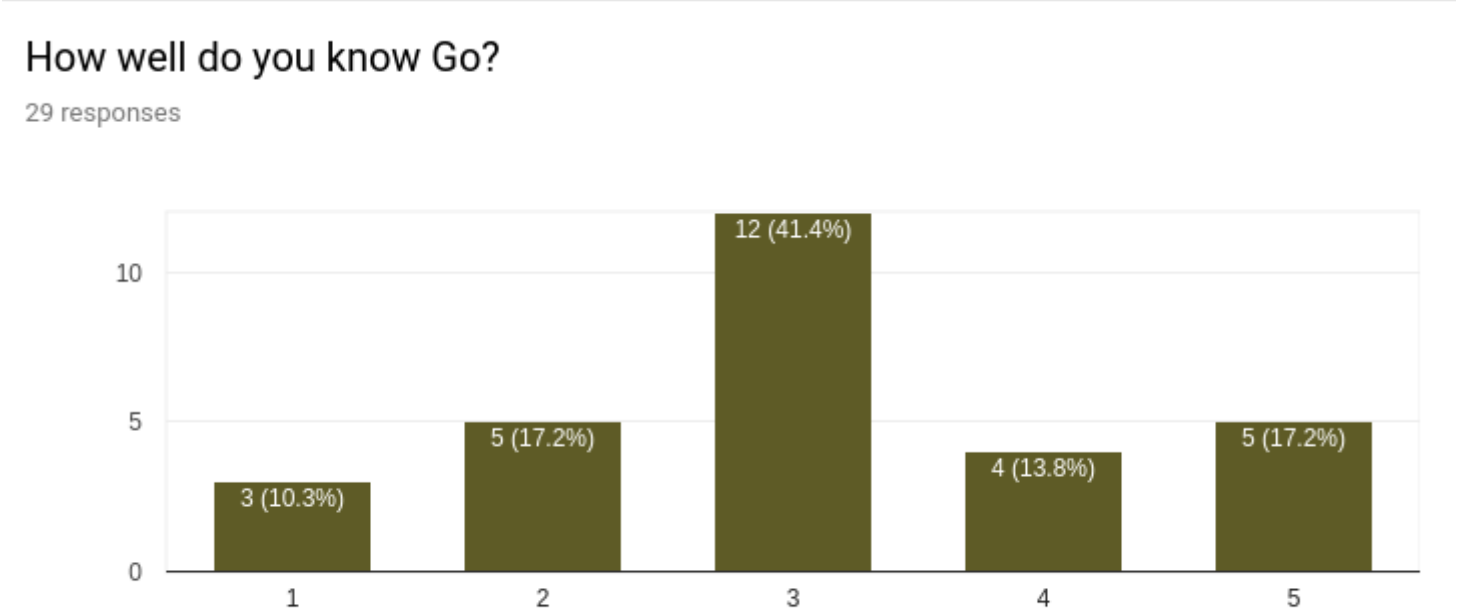
Websocket Testing in Go

Eyal Posener
Stratoscale

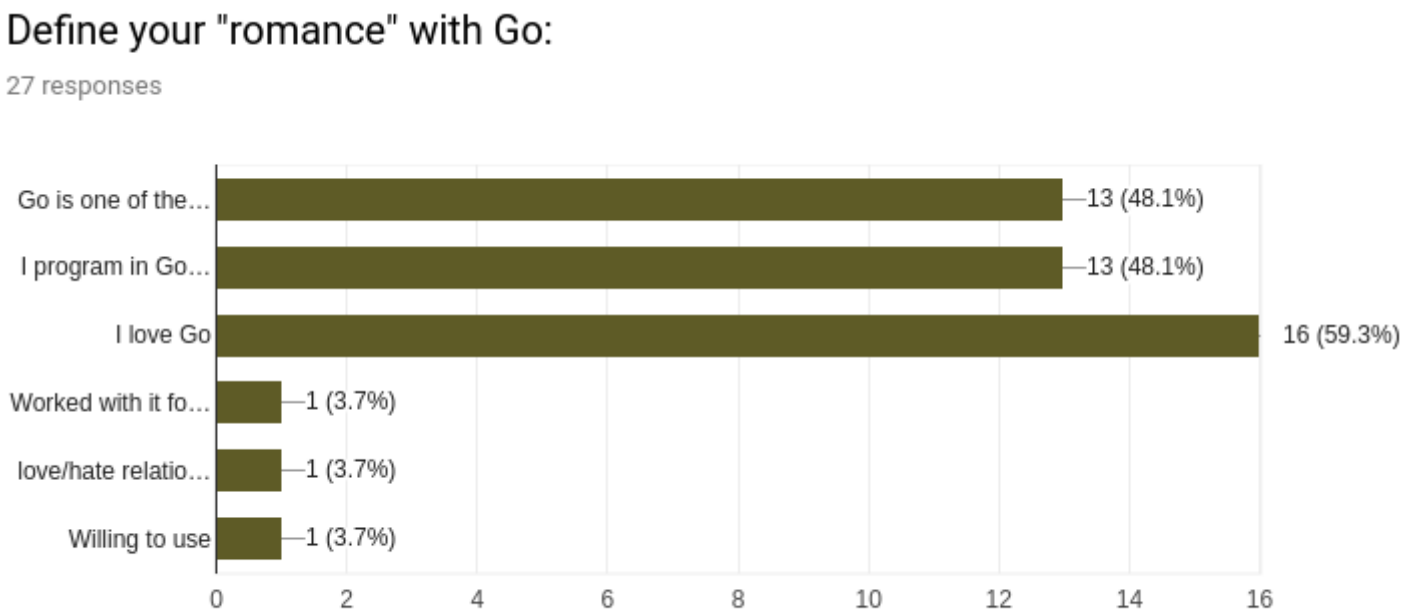
Community Survey



1: Heard about it

5: Expert

Community Survey



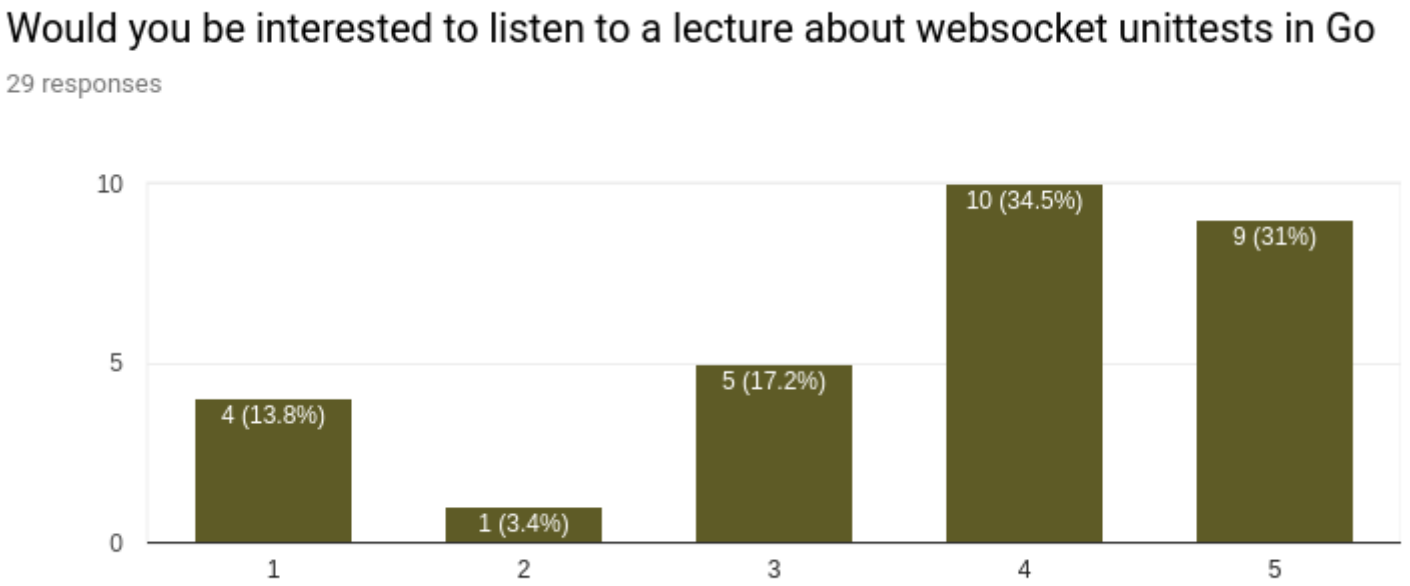
- Go is one of the languages I use at work
- I program in Go in my spare time
- I love Go
- Worked with it for a while / Love/hate relationship / Willing to use

Agenda

- Go and testing
- Go and HTTP
- Go and HTTP and testing
- Go and Websockets
- Go and Websockets and testing
- What is missing
- Solution
- The `net.Pipe` challenge

Everybody loves testing

From the survey:



Go + tests = <3

- Seriously, go was built so you could test your code easily
- ... Super easily
- `testing` package in the standard library
- `go test` tool as part of the go command line
- Benchmarks
- Run tests in parallel
- Coverage as a standard tool
- + a lot more...

Go + HTTP Server = Big <3

HTTP Handler interface (sinners are frameworks who wrap it)

```
type Handler interface {
    ServeHTTP(ResponseWriter, *Request)
}
```

You get a strong and concurrent web server out of the box!

Example:

```
func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "<h1>I love %s!</h1>", r.URL.Path[1:])
    })
    http.ListenAndServe(":8080", nil)
}
```

Run

localhost:8080/Go (http://localhost:8080/Go)

localhost:8080/Stratoscale (http://localhost:8080/Stratoscale)

Go + Testing + HTTP Servers > 3

`httptest.NewServer` option:

```
func main() {
    server := httptest.NewServer(handler)
    defer server.Close()
    response, _ := http.Get(server.URL + "/Go")
    defer response.Body.Close()
    body, _ := ioutil.ReadAll(response.Body)
    fmt.Println(string(body))
}
```

Run

`httptest.NewRecorder` option:

```
func main() {
    rec := httptest.NewRecorder()
    request, _ := http.NewRequest(http.MethodGet, "http://example.com/Stratoscale", nil)
    handler.ServeHTTP(rec, request)
    fmt.Println(rec.Body.String())
}
```

Run

Unless you have a very good reason, prefer the second.

Websockets:

- Give full duplex communication. (Instead of polling / long polling)
- Low messages overhead
- Use existing http/s ports: good for strict firewalled environments.
- Ride over existing http/s protocol.

Websockets Basics:

HTTP handshake:

- Request:

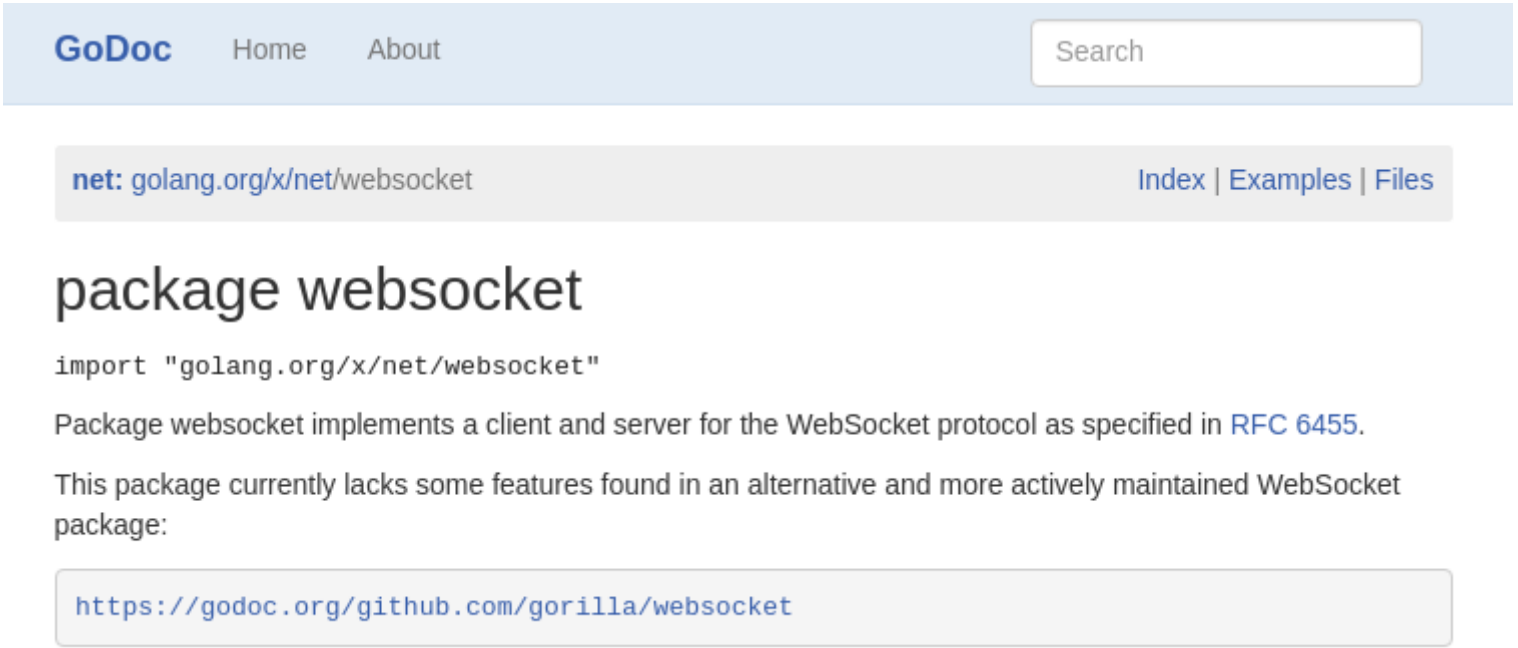
```
GET /ws HTTP/1.1
Upgrade: websocket
Connection: Upgrade
...
```

- Response:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
...
```

--> TCP based Full duplex communication

Websockets in Go



Example: echo server

```
func Handler(w http.ResponseWriter, r *http.Request) {
    c, err := upgrader.Upgrade(w, r, nil)
    if err != nil {
        log.Print("upgrade:", err)
        return
    }
    defer c.Close()
    var message string
    for {
        err = c.ReadJSON(&message)
        if err != nil {
            log.Println("read:", err)
            return
        }
        log.Printf("recv: %s", message)
        err = c.WriteJSON(message)
        if err != nil {
            log.Println("write:", err)
            return
        }
    }
}
```

Test it?

- Use the httptest.NewServer!

```
func main() {
    server := httptest.NewServer(http.HandlerFunc(echo.Handler))
    defer server.Close()

    d := websocket.Dialer{}
    c, resp, err := d.Dial("ws://"+server.Listener.Addr().String()+"/ws", nil)
    if err != nil {
        panic(err)
    }
    defer c.Close()
    fmt.Println("resp status code:", resp.StatusCode)
    c.WriteJSON("test")
    var ret string
    c.ReadJSON(&ret)
    fmt.Println("read:", ret)
}
```

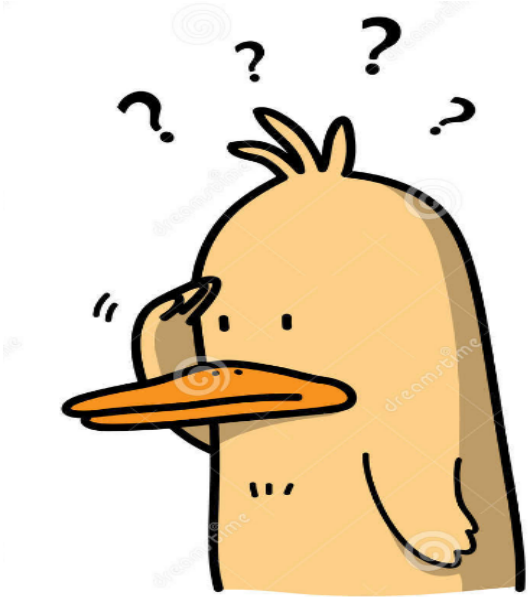
Run

Test it?

- Use `httptest.NewRecorder`?

```
Not possible!
```

- Why do we want something like `httptest.NewRecorder`?



Let's build it together!

We want to achieve something of the form:

```
d := NewTestDialer(http.HandlerFunc(echo.Handler))
c, resp, err := d.Dial("ws://whatever/ws", nil)
```

- Use the gorilla's `webscoket.Dialer.NetDial` field:

```
NetDial: func(network, addr string) (net.Conn, error)
```

- Use `net.Pipe` that returns two paired connections:

```
client, server := net.Pipe()
```

net.Conn interface (https://golang.org/pkg/net/#Conn)

net.Pipe implementation (https://golang.org/src/net/pipe.go?s=461:485#L8)

Let's build it together!

```
func NewTestDialer(h http.Handler) *websocket.Dialer {
    client, server := net.Pipe()

    go func() {
        req, err := http.ReadRequest(bufio.NewReader(server))
        if err != nil {
            return
        }
        rec := &recorder{conn: server}
        h.ServeHTTP(rec, req)
    }()

    return &websocket.Dialer{
        NetDial: func(network, addr string) (net.Conn, error) {
            return client, nil
        },
    }
}
```

recorder: implements ResponseWriter + Hijacker

```
// recorder implements http.ResponseWriter and http.Hijacker interface
type recorder struct {
    httptest.ResponseRecorder
    conn net.Conn
}

// Hijack implements the Hijacker interface
func (r *recorder) Hijack() (net.Conn, *bufio.ReadWriter, error) {
    rw := bufio.NewReadWriter(bufio.NewReader(r.conn), bufio.NewWriter(r.conn))
    return r.conn, rw, nil
}

// WriteHeader is part of the ResponseWriter interface
func (r *recorder) WriteHeader(code int) {
    resp := http.Response{StatusCode: code, Header: r.Header()}
    resp.Write(r.conn)
}
```

Works?

```
func main() {
    d := NewTestDialer(http.HandlerFunc(echo.Handler))
    c, resp, err := d.Dial("ws://whatever/ws", nil)
    if err != nil {
        panic(err)
    }
    fmt.Println("resp status code:", resp.StatusCode)
    err = c.WriteJSON("test")
    if err != nil {
        panic(err)
    }
    var ret string
    c.ReadJSON(&ret)
    fmt.Println("read:", ret)
}
```

Run

panic: set pipe: deadline not supported

Actually: net.Pipe fails the `nettest.TestConn` tests.

TestConn tests net pipes implementations (https://github.com/golang/net/blob/master/nettest/conntest.go#L37)

```
func TestNetPipe(t *testing.T) {
    nettest.TestConn(t,
        func() (c1, c2 net.Conn, stop func(), err error) {
            c1, c2 = net.Pipe()
            stop = func() { c1.Close(); c2.Close() }
            return
        },
    )
}
```

==> Fails

Let's write a net.Pipe that supports deadlines!

(Actually someone already PR this) (https://go-review.googlesource.com/c/37402/)

The challenges

- Read/Write methods blocks, how do you cancel them?

My solution:

- `bytes.Buffer` which is not blocking, returns EOF error on read when empty
- `sync.Cond` for notifying on new content in pipe
- After write: signal.
- After deadline: set error and broadcast.

Voilà (https://github.com/posener/wstest#examples)

Thanks

- The slides are available on github.com/posener/meetups (https://github.com/posener/meetups)
- wstest package is available on github.com/posener/wstest (https://github.com/posener/wstest)

Last thing: Bash completion for the go command:

```
go get -u github.com/posener/complete/gocomplete && gocomplete -install
```

References

Gorilla server Upgrade() function (https://github.com/gorilla/websocket/blob/master/server.go#L106-L229)

Gorilla client Dial function (https://github.com/gorilla/websocket/blob/master/client.go#L167-L392)

Thank you

Eyal Posener
Stratoscale
posener@gmail.com (mailto:posener@gmail.com)