Golang Taipei Meetup #46

Why Diode client uses Go language?

Peter Lai, Blockchain Engineer at Diode

Maicoin HQ, Taipei City, Taiwan

December 24, 2019 🎅🎄

About Me

- Blockchain Engineer at Diode
- Editor of Taipei Ethereum Meetup
- Open source contributor
- Love to learn new technology
- Programming languages: JS, GO, PHP, C, PYTHON
- Twitter: @alk03073135 | Github: @sc0vu

Web3

- Peer-to-Peer networks
- Incentive for nodes who transfer
- Remote procedure calls
  - JSON RPC, eg {id: 1, method: “ping”, params: []}
  - Edge RPC, eg __[“ping”]
  - ……

Concurrency in GO

- Goroutines and channels

- Event system (epoll, kqueue)
  - 1m-go-websockets

Unbuffered Channel

- Block the execution until dataChannel has Data

```
Peters-MacBook-Pro:diode_go_client peterlai$ go run t.go
```

```go
type Data struct {
    DataType string
    Data     []byte
}

func PrintData(d Data) {
    if d.DataType == "string" {
        log.Println(string(d.Data))
    } else {
        log.Println(d.Data)
    }
}

func main() {
    dataChannel := make(chan Data)
    go func() {
        receivedData := Data{
            DataType: "string",
            Data:     []byte("hello world!"),
        }
        time.Sleep(1 * time.Second)
        dataChannel <- receivedData
    }()
    data := <-dataChannel
    PrintData(data)
}
```

Buffered Channel

- Won’t block the execution

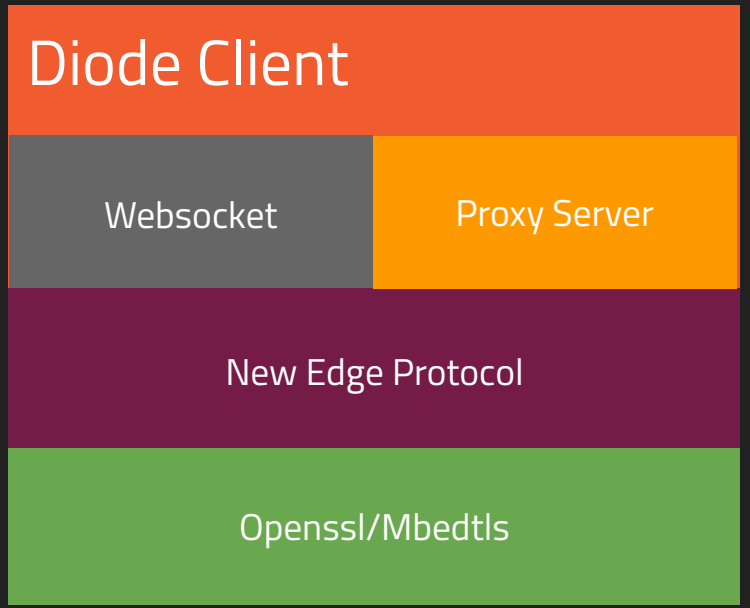
- Loop through the channel


```go
func main() {
    dataChannel := make(chan Data, 10)
    go func() {
        for i := 0; i <= 9; i++ {
            receivedData := Data{
                DataType: "string",
                Data:     []byte(strconv.Itoa(i)),
            }
            time.Sleep(1 * time.Second)
            select {
            case dataChannel <- receivedData:
                continue
            default:
                break
            }
        }
        close(dataChannel)
    }()
    for data := range dataChannel {
        PrintData(data)
    }
```

How do we design client

- Use openssl instead of crypto/tls
- Separate tcp read and write
- Use proxy to delegate request
- Use websocket to stream real-time

  data



Separate tcp read/write

```
type SSL struct {
    callChannel        chan Call
    calls              []Call
    tcpIn              chan []byte
    conn               *openssl.Conn
    ctx                *openssl.Ctx
    tcpConn            *tcpkeepalive.Conn
    addr               string
    mode               openssl.DialFlags
    isValid            bool
    enableKeepAlive    bool
    keepAliveCount     int
    keepAliveIdle      time.Duration
    keepAliveInterval  time.Duration
    memoryCache        *cache.Cache
    closed             bool
    totalConnections   int64
    totalBytes         int64
    counter            int64
    rm                 sync.Mutex
    clientPrivKey      *ecdsa.PrivateKey
    RegistryAddr       [20]byte
    FleetAddr          [20]byte
    RPCServer          *RPCServer
}
```

```
rpcServer.addWorker(func() {
    // infinite read from stream
    for {
        err := rpcServer.s.readContext()
        if err != nil {
            rpcServer.rm.Lock()
            if !rpcServer.closed {
                rpcServer.rm.Unlock()
                rpcServer.Close()
            } else {
                rpcServer.rm.Unlock()
            }
            return
        }
    }
})
```

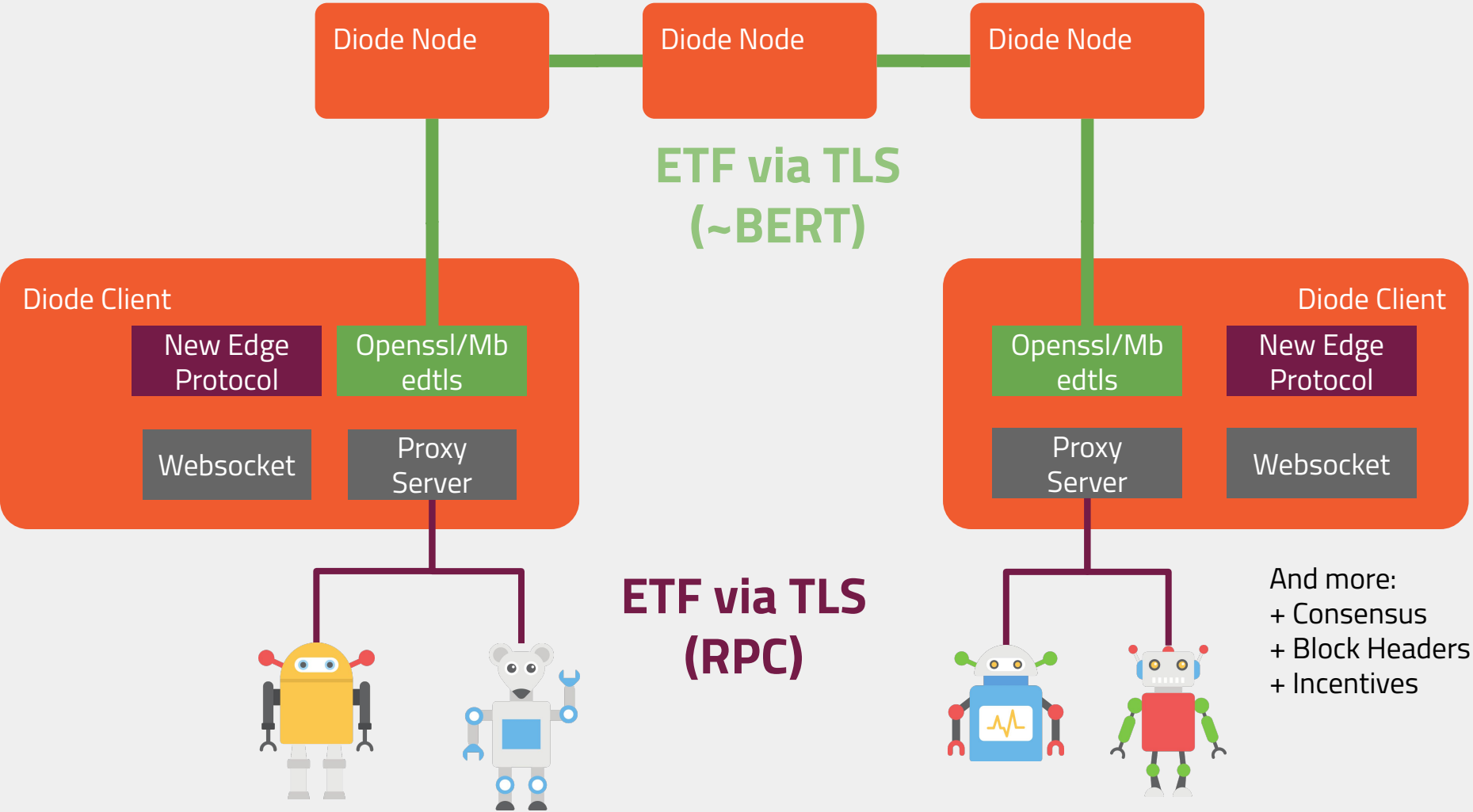
Separate tcp read/write

```
case res := <-rpcServer.s.tcpIn:
    rpcServer.s.CheckTicket()
    // log.Printf("GOT: isResp=%v %v\n", isResponseType, string(res))
    if isResponseType(res) || isErrorType(res) {
        call := rpcServer.s.calls[0]
        // log.Printf("recv: %v", responseMethod(res))
        if responseMethod(res) != call.method {
            log.Printf("Uh, got different response type: %v %v", call.method, string(res))
        }

        rpcServer.s.calls = rpcServer.s.calls[1:]
        call.responseChannel <- res
        close(call.responseChannel)
        continue
    }
    request, err := parseRPCRequest(res)
    if err != nil {
        log.Printf("This is not an RPC request %v\n", err)
        continue
    }
    rpcServer.requestChan <- request
```

Concurrency bugs in GO

- Blocking bugs
  - (mis) Protection of shared memory
  - Misuse of message passing
- Non-blocking bugs
  - Failing to protect shared memory
  - Error during message passing
- Understanding Real-World Concurrency Bugs in Go


Diode Node
Diode Node
Diode Node
ETF via TLS
(~BERT)
Diode Client
New Edge Protocol
Openssl/Mb edtls
Websocket
Proxy Server
Diode Client
Openssl/Mb edtls
New Edge Protocol
Proxy Server
Websocket
ETF via TLS
(RPC)
And more:
+ Consensus
+ Block Headers
+ Incentives

Demo

- SSH into pi through Diode network
- View video stream

Reference

- Blockquick
- Diode_wiki
- Understanding Real-World Concurrency Bugs in Go

IOT SECURITY IS BROKEN
MAKE IT ROCK SOLID

https://diode.io
https://github.com/diodechain
Get Involved