GoFunc
2024

gRPC: Under the Hood

Алексей Акулович
Backend developer 🥑

```
Доклад struct {
    Введение   Chapter
    HTTP1      Chapter
    HTTP2      Chapter
    GRPC       Chapter
    Выводы     *Chapter
}
```

Введение

gRPC - детище Google. Первое?

Stubby в 2001

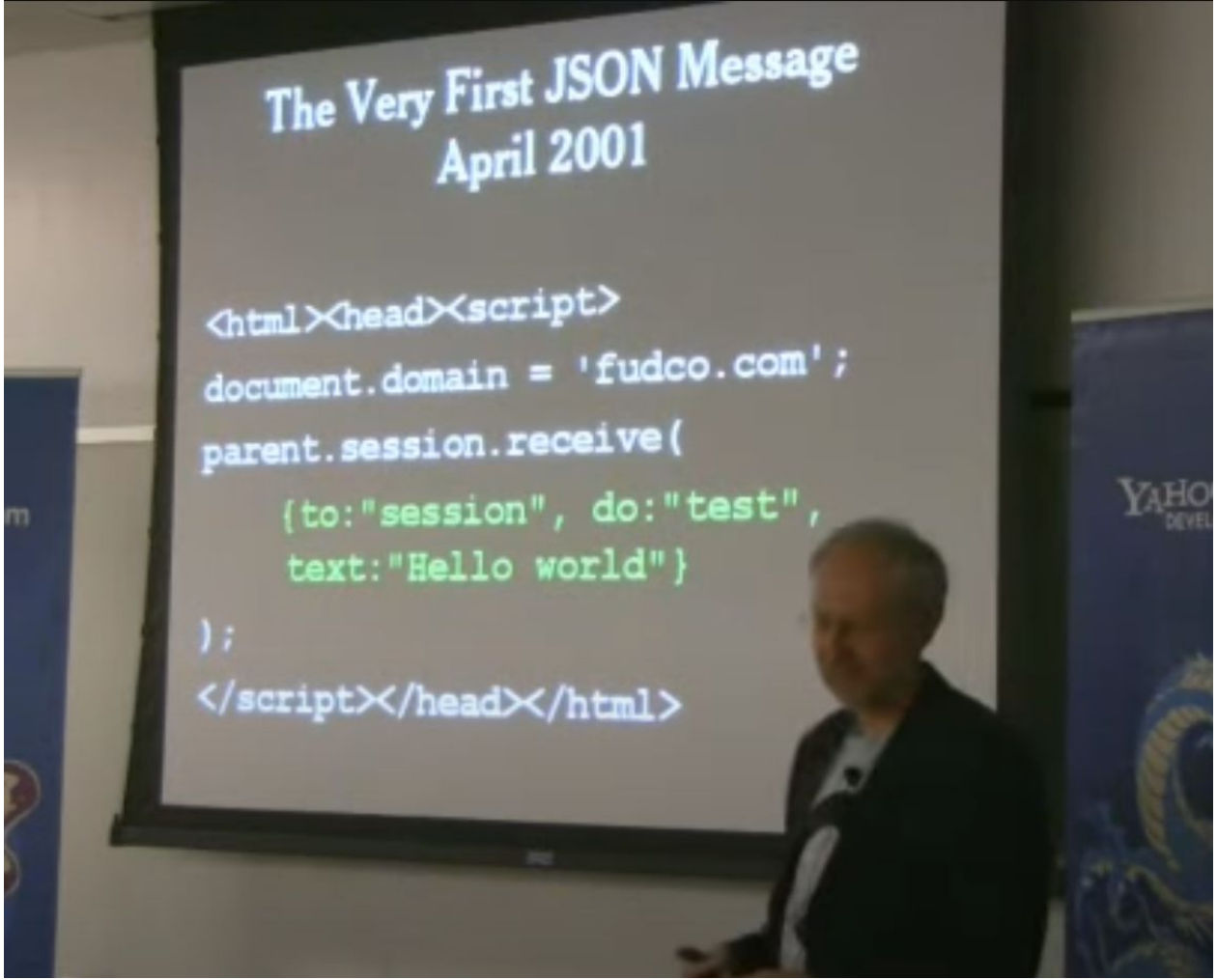
> “an RPC framework that consists of a core RPC layer that can handle internet-scale of tens of billions of requests per second (yes, billions!)”

cloud.google.com/blog/products/gcp/grpc-a-true-internet-scale-rpc-framework-is-now-1-and-ready-for-production-deployments

Stubby в 2001

> “an RPC framework that consists of a core RPC layer that can handle internet-scale of tens of billions of requests per second (yes, billions!)”

JSON over TCP (HTTP/1)	



youtube.com/watch?v=-C-JoyNuQJs

Stubby в 2001

> “an RPC framework that consists of a core RPC layer that can handle internet-scale of tens of billions of requests per second (yes, billions!)”

JSON over TCP (HTTP/1) 🤔	protobuf over HTTP/1

protobuf.dev/history/

groups.google.com/g/golang-nuts/c/34ITEzwxmE4/m/fAhG9bA72poJ

OpenSource Stubby 2011?

Russ Cox 5 авг. 2011 г., 15:41:22

– tsuna, golan...@googlegroups.com

> at OSCON Rob Pike said that the Go implementation of Stubby RPC was
> probably the easiest to open-source. That would be really awesome :)
>
> I think pretty much all the Google Go team attended OSCON, that was
> pretty cool. Thanks for coming.

If you are interested in RPC in the context of Go alone,
an alternative that is available today is Go's own
rpc package: http://golang.org/pkg/rpc/. It uses a
Go-specific encoding called gob instead of protocol
buffers, but it provides the same functionality.
(Again the drawback is that it is Go-only, not cross
language like protobufs, but that's not too much
different from having only the Go Stubby available.)

Russ

groups.google.com/g/golang-nuts/c/34ITEzwxmE4/m/gNEfNOc3eJYJ

go.dev/blog/gob

В 2015 как opensource замена Stubby появляется gRPC

Почему в 2015?

2001 - protobuf, Stubby

2001 - protobuf, Stubby

2011 - Go rpc (gob)

2001 - protobuf, Stubby

2011 - Go rpc (gob)

2010 - spdy

2014 - http/2

2001 - protobuf, Stubby

2011 - Go rpc (gob)

2010 - spdy

2014 - http/2

2015 - gRPC

2016 - quic

2022 - http/3

2001 - protobuf, Stubby

2011 - Go rpc (gob)

2010 - spdy

2014 - http/2 новый транспорт!

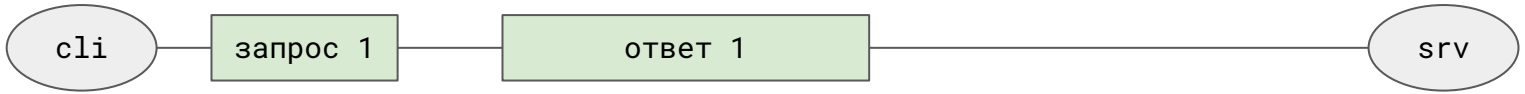
2015 - gRPC

2016 - quic

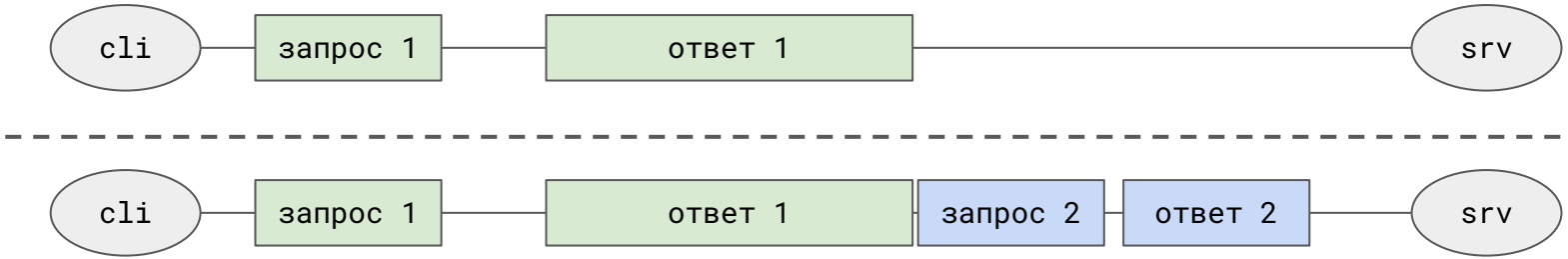
2022 - http/3

grpc.io/blog/principles/

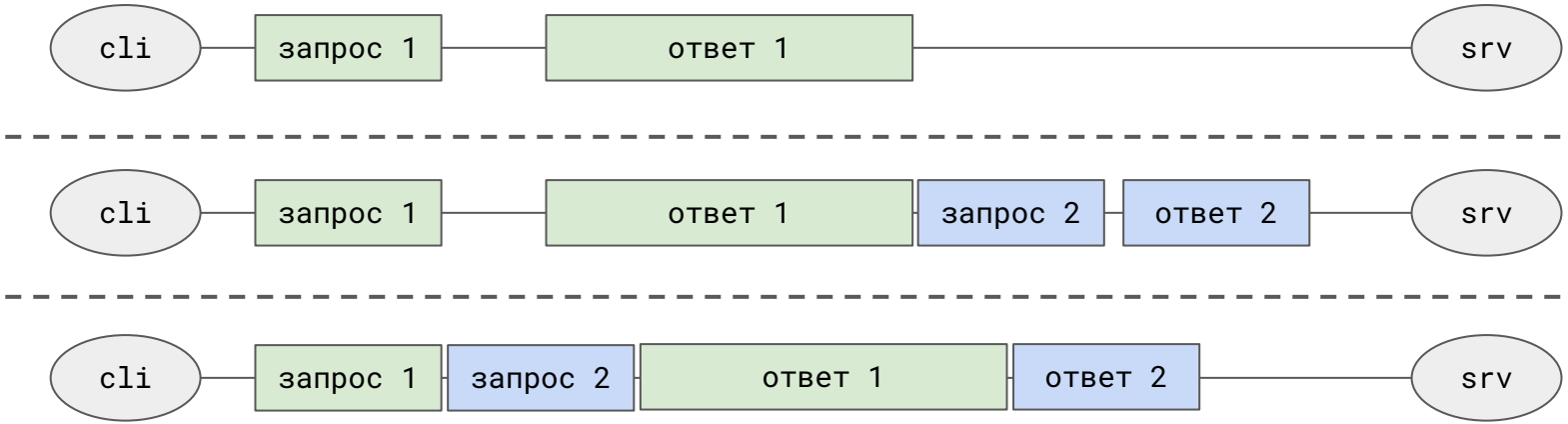
HTTP/1



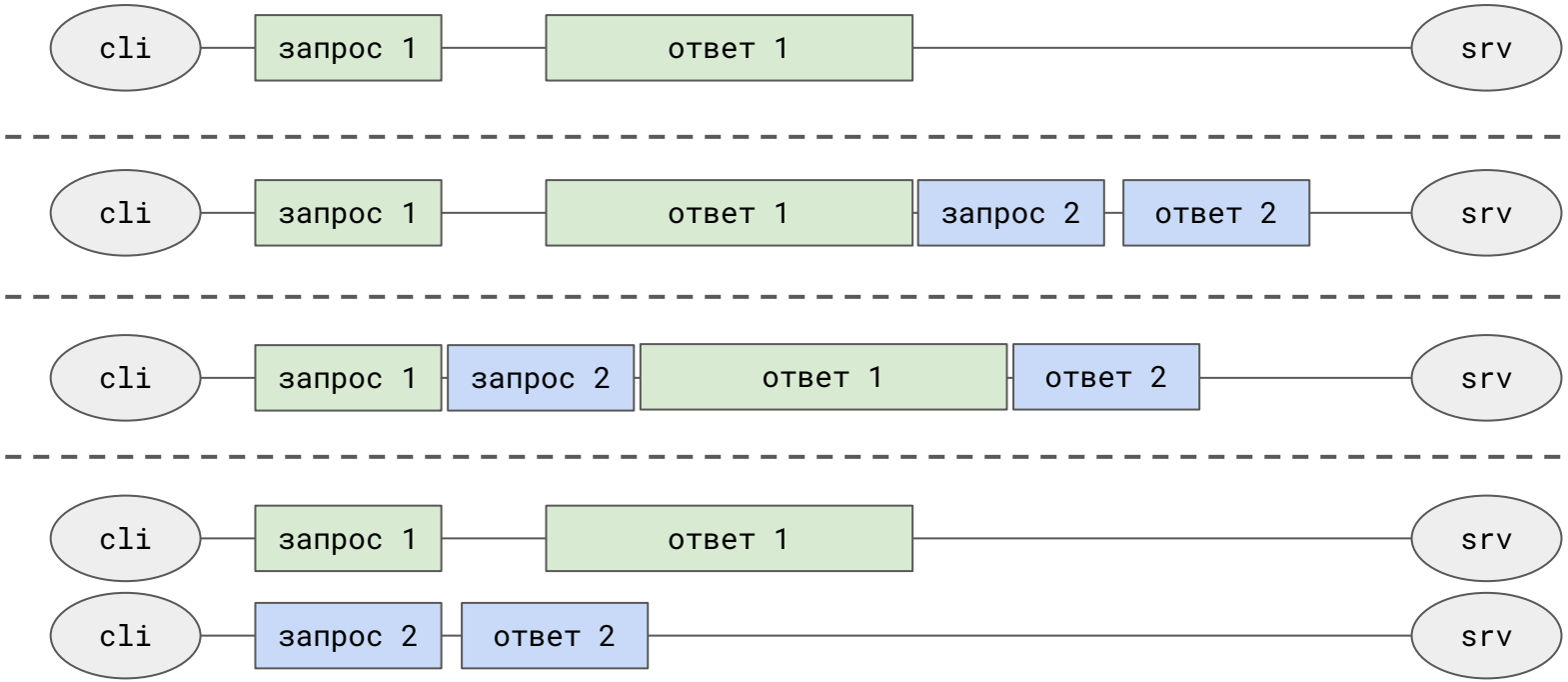
HTTP/1


cli
запрос 1
ответ 1
srv
cli
запрос 1
ответ 1
запрос 2
ответ 2
srv

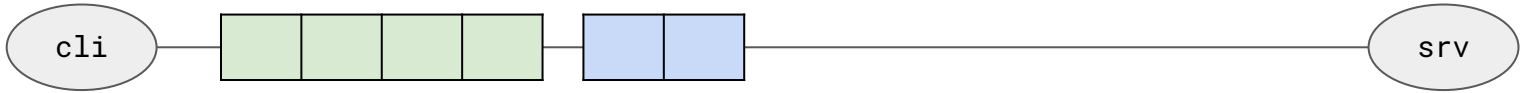
HTTP/1


cli
запрос 1
ответ 1
srv
cli
запрос 1
ответ 1
запрос 2
ответ 2
srv
cli
запрос 1
запрос 2
ответ 1
ответ 2
srv

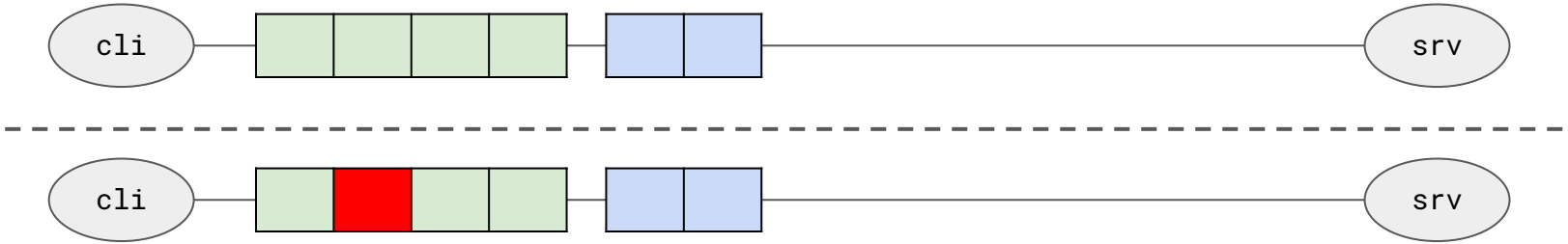
HTTP/1


cli
запрос 1
ответ 1
srv
cli
запрос 1
ответ 1
запрос 2
ответ 2
srv
cli
запрос 1
запрос 2
ответ 1
ответ 2
srv
cli
запрос 1
ответ 1
srv
cli
запрос 2
ответ 2
srv

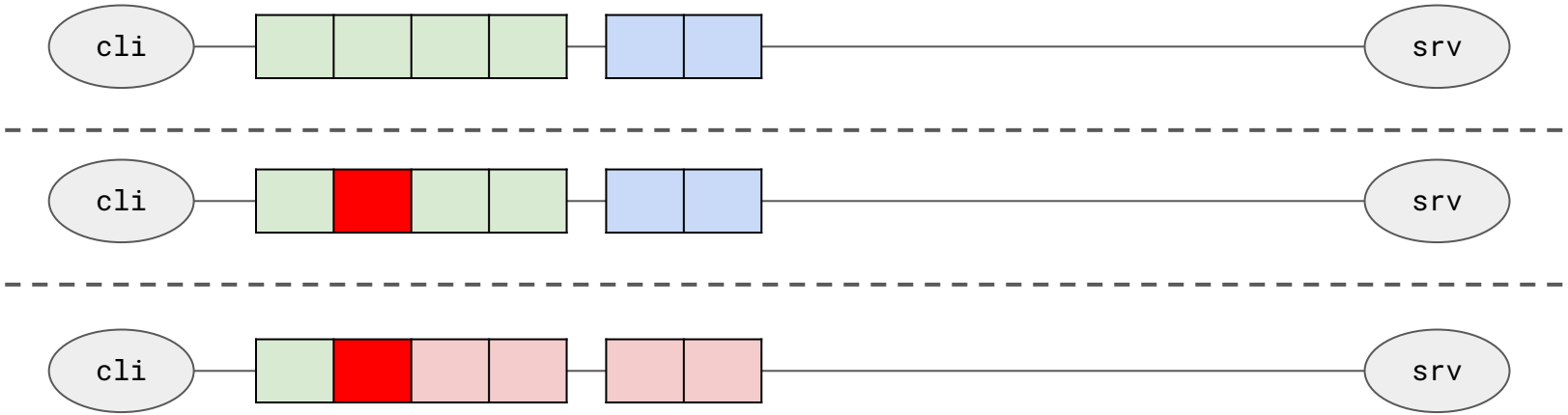
HTTP/1



HTTP/1



HTTP/1



Чем HTTP/2 лучше HTTP/1 для gRPC?

Чем HTTP/2 лучше HTTP/1 для gRPC?

```
service GrpcTestService {
  rpc Ping(PingRequest) returns (PingResponse);
}

message PingRequest {
  int32 val = 1;
}

message PingResponse {
  int32 val = 1;
}
```

История одного байта

TCP ->

```
l, _ := net.Listen("tcp", addr)

for {
    conn, _ := l.Accept()

    go func() {
        serveConn(conn)
        _ = conn.Close()
    }()
}
```

TCP -> ?

TCP -> ?

Читать RFC? Пффф! Реверсим! *

* Не является рекомендацией :)

Как подключаться к серверу?

Как подключаться к серверу?

```
curl 'http://127.0.0.1:8085/some/path'
```

```
conn,    := grpc.Dial(l.Addr().String())
defer conn.Close()

client := grpcTestPb.NewGrpcTestServiceClient (conn)
resp, _ := client.Ping(ctx, &grpcTestPb.PingRequest{Val: 42})
```

Как подключаться к серверу?

```
curl --http2 'http://127.0.0.1:8085/some/path'
```

```
▾ Hypertext Transfer Protocol
  ▸ POST /some/path HTTP/1.1\r\n
    Host: 127.0.0.1:8085\r\n
    User-Agent: curl/7.88.1\r\n
    Accept: */*\r\n
    Connection: Upgrade, HTTP2-Settings\r\n
    Upgrade: h2c\r\n
  ▸ HTTP2-Settings: AAMAAABkAAQCAAAAAAIAAAAA\r\n
    \r\n
```

HTTP/1.1 запрос с доп заголовками

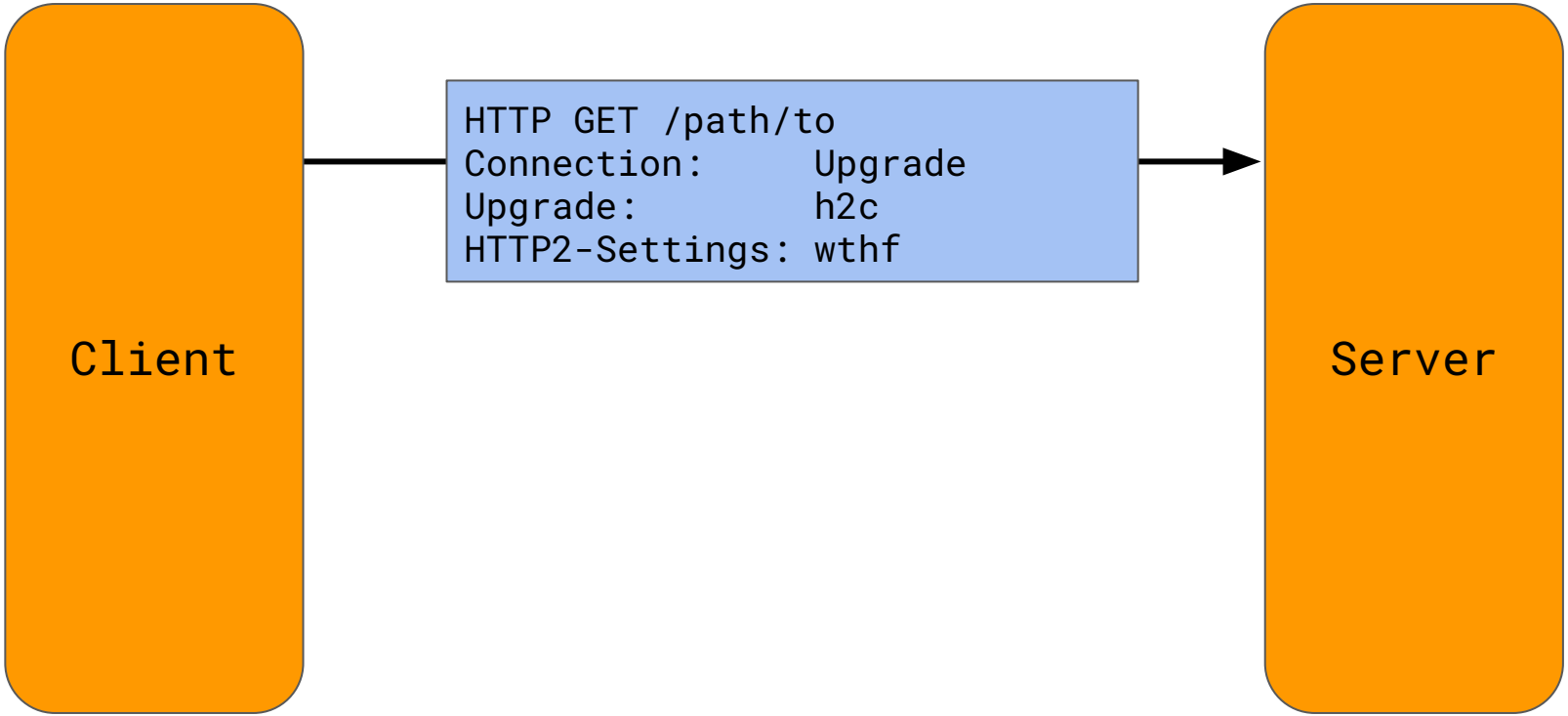
```
Connection: Upgrade, HTTP2-Settings\r\n
Upgrade: h2c\r\n
HTTP2-Settings: AAMAAABkAAQCAAAAAAIAAAAA\r\n
```

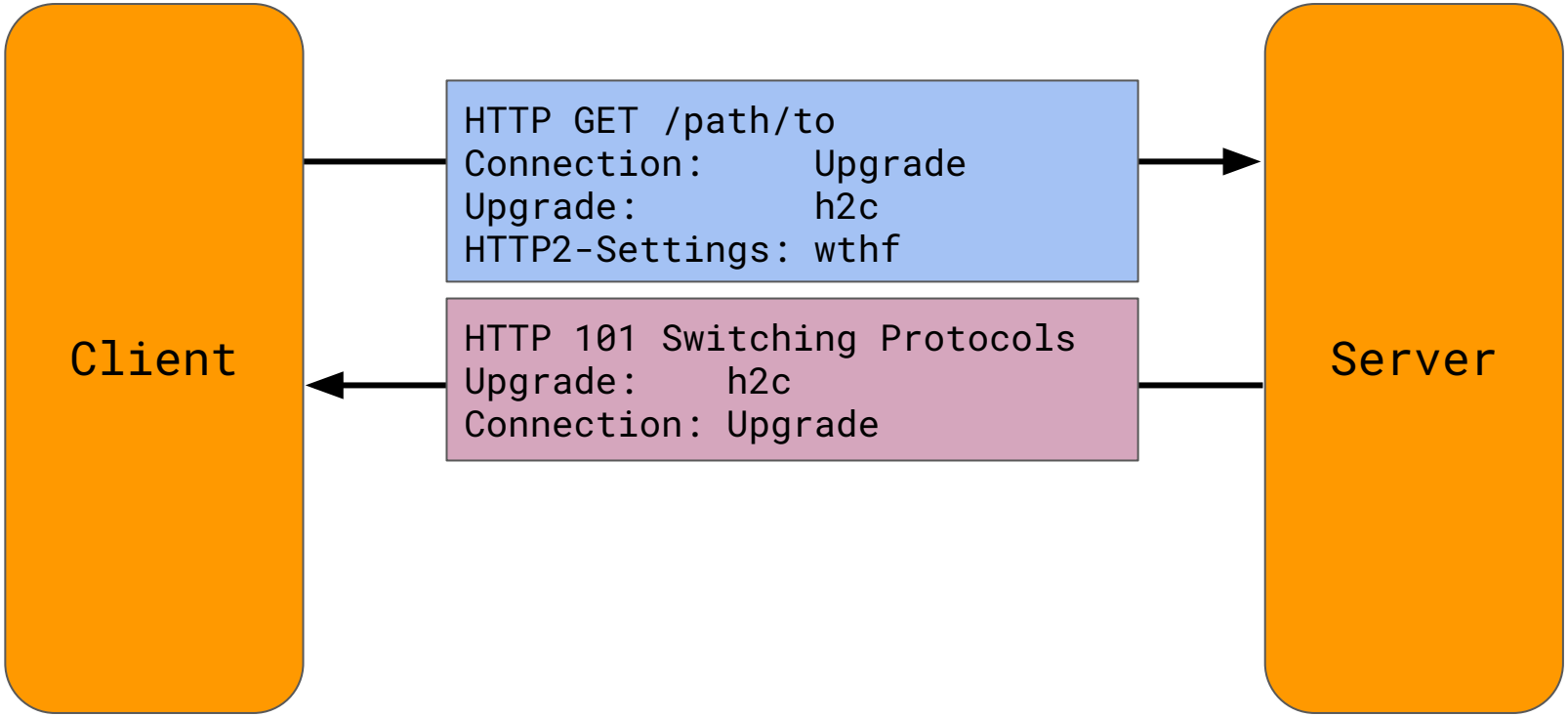
HTTP/1.1 запрос с доп заголовками

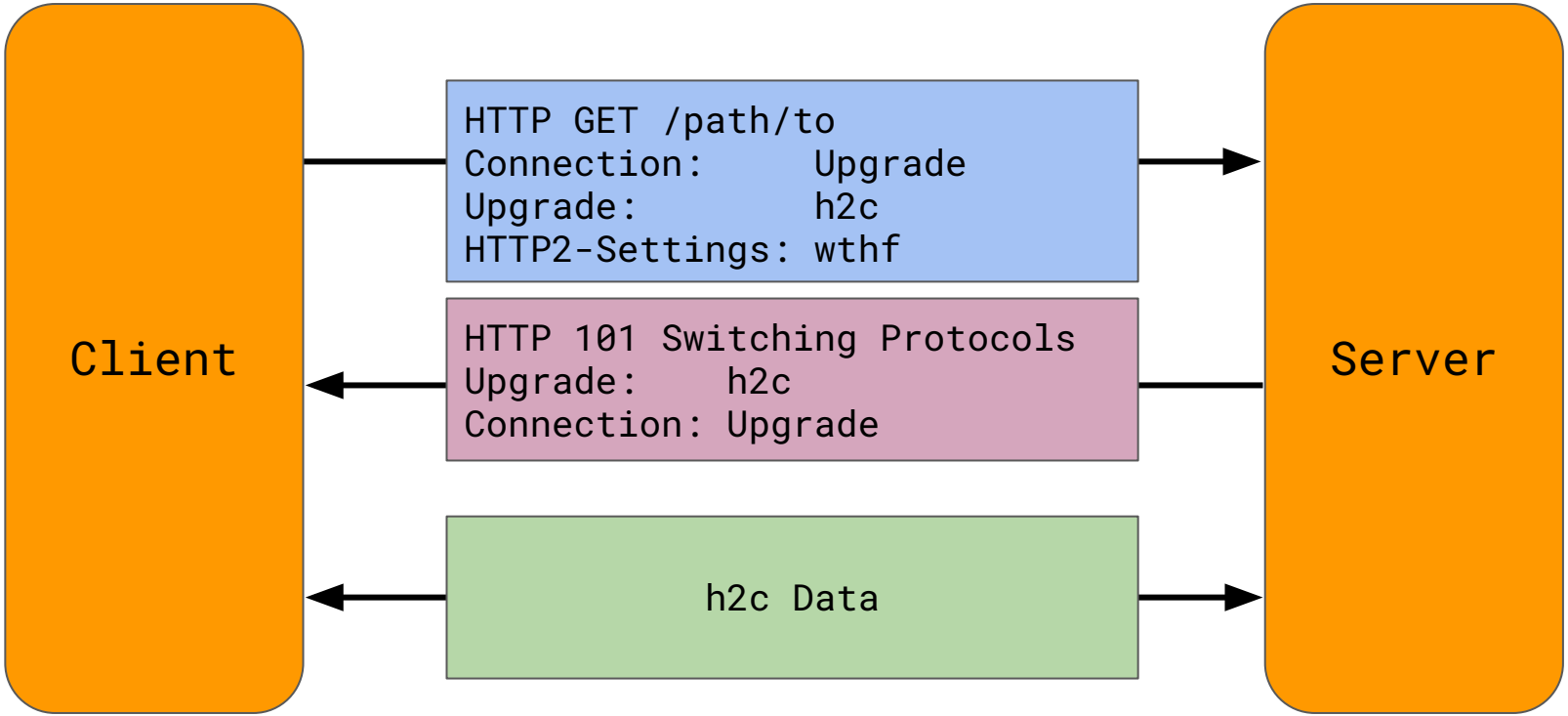
```
Connection: Upgrade, HTTP2-Settings\r\n
Upgrade: h2c\r\n
HTTP2-Settings: AAMAAABkAAQCAAAAAAIAAAAA\r\n
```

Это предложение поменять протокол

developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Upgrade

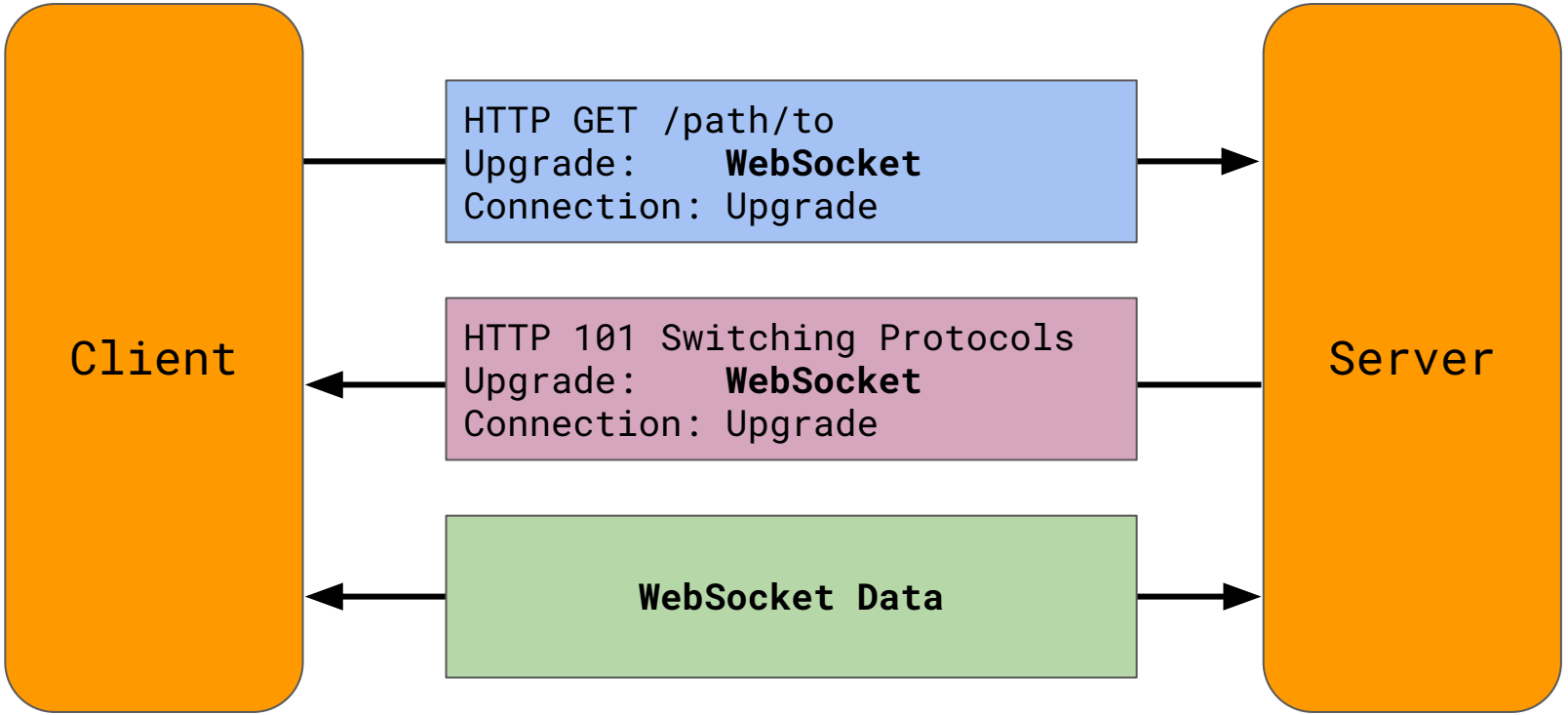

Client
HTTP GET /path/to
Connection: Upgrade
Upgrade: h2c
HTTP2-Settings: wthf
Server

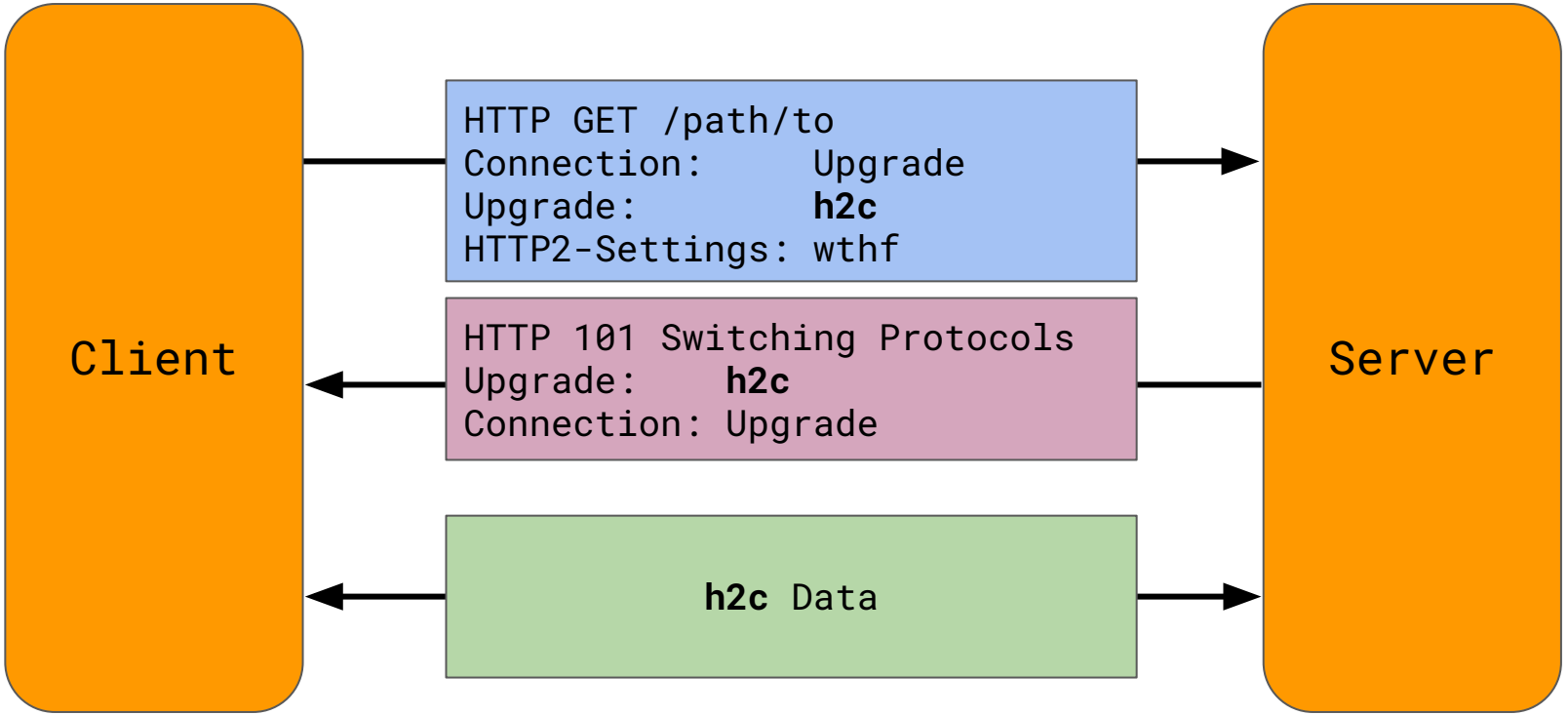

Client
HTTP GET /path/to
Connection: Upgrade
Upgrade: h2c
HTTP2-Settings: wthf
HTTP 101 Switching Protocols
Upgrade: h2c
Connection: Upgrade
Server


Client
HTTP GET /path/to
Connection: Upgrade
Upgrade: h2c
HTTP2-Settings: wthf
HTTP 101 Switching Protocols
Upgrade: h2c
Connection: Upgrade
h2c Data
Server

Что-то знакомое?

```
Connection: Upgrade
Upgrade: websocket
```


Client
HTTP GET /path/to
Upgrade: WebSocket
Connection: Upgrade
HTTP 101 Switching Protocols
Upgrade: WebSocket
Connection: Upgrade
WebSocket Data
Server


Client
HTTP GET /path/to
Connection: Upgrade
Upgrade: h2c
HTTP2-Settings: wthf
HTTP 101 Switching Protocols
Upgrade: h2c
Connection: Upgrade
h2c Data
Server

“To h2c, or not to h2c, that is the question”

“To h2c, or not to h2c, that is the question”

3.1. HTTP/2 Version Identification

The protocol defined in this document has two identifiers. Creating a connection based on either implies the use of the transport, framing, and message semantics described in this document.

* The string "h2" identifies the protocol where HTTP/2 uses Transport Layer Security (TLS); see Section 9.2. This identifier is used in the TLS Application-Layer Protocol Negotiation (ALPN) extension [TLS-ALPN] field and in any place where HTTP/2 over TLS is identified.

  The "h2" string is serialized into an ALPN protocol identifier as the two-octet sequence: 0x68, 0x32.

* The "h2c" string was previously used as a token for use in the HTTP Upgrade mechanism's Upgrade header field (Section 7.8 of [HTTP]). This usage was never widely deployed and is deprecated by this document. The same applies to the HTTP2-Settings header field, which was used with the upgrade to "h2c".

datatracker.ietf.org/doc/html/rfc9113#name-http-2-version-identificati

“To h2c, or not to h2c, that is the question”

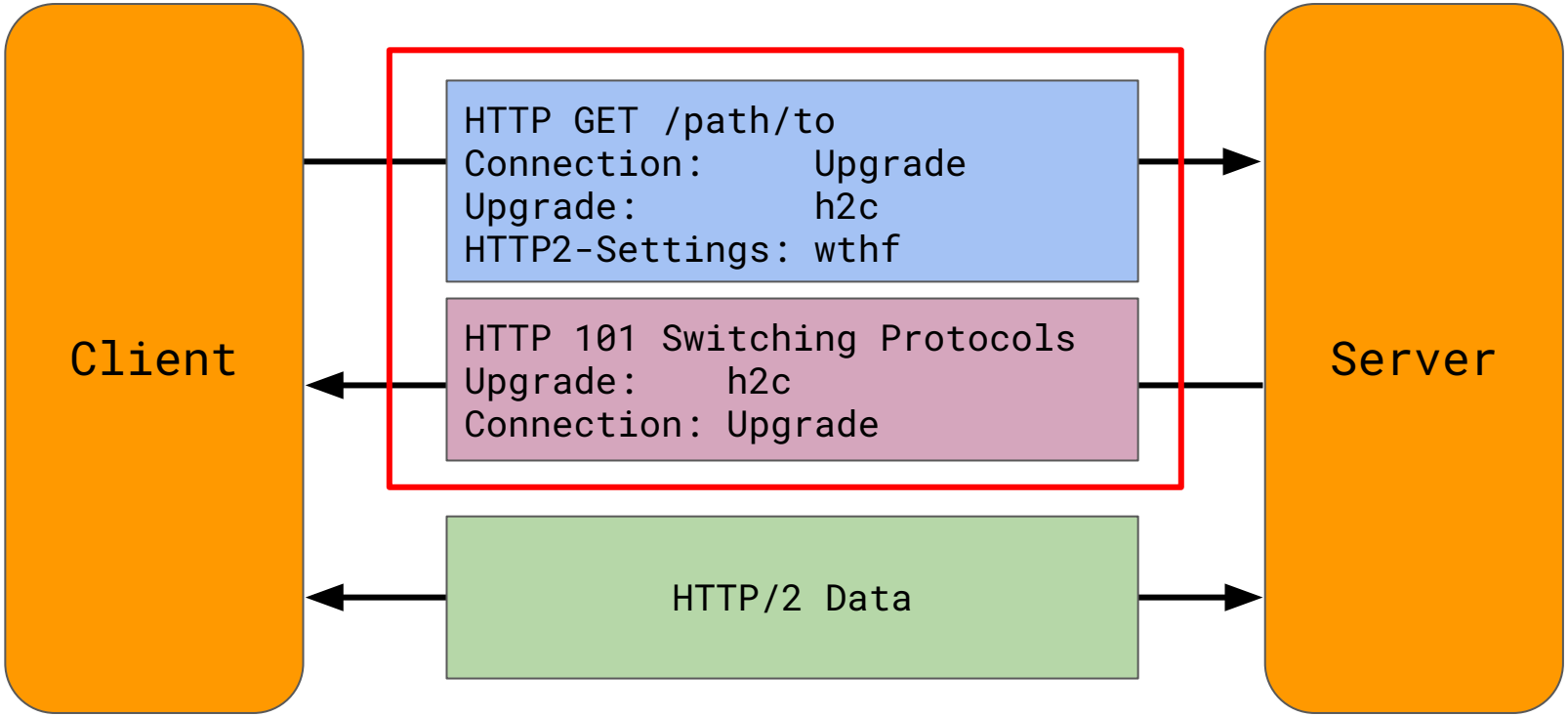
h2	HTTP/2 over HTTPS/TLS	default
h2c	HTTP/2 over HTTP	deprecated

“To h2c, or not to h2c, that is the question”

h2	HTTP/2 over HTTPS/TLS	default (ALPN)
h2c	HTTP/2 over HTTP	deprecated

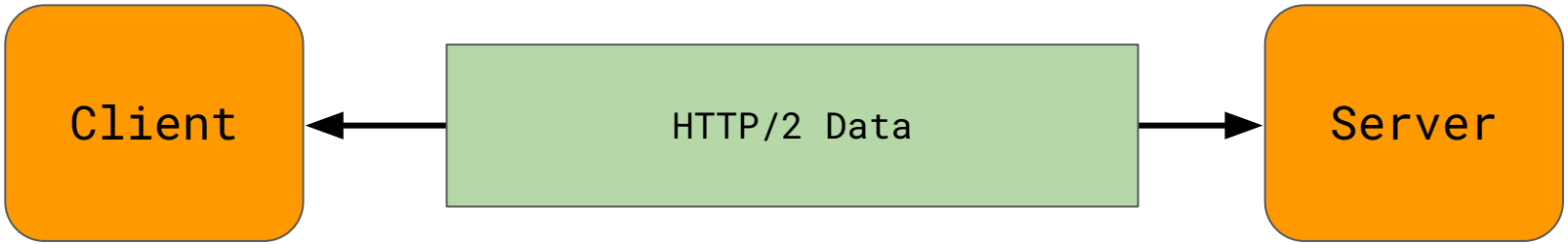
```
curl --http2 'http://127.0.0.1:8085/some/path'
```

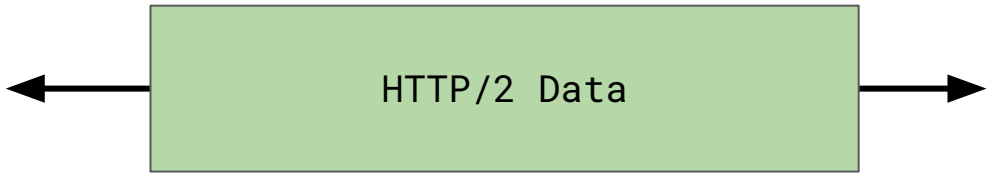
```
Connection: Upgrade, HTTP2-Settings\r\n
Upgrade: h2c\r\n
```


Client
HTTP GET /path/to
Connection: Upgrade
Upgrade: h2c
HTTP2-Settings: wthf
HTTP 101 Switching Protocols
Upgrade: h2c
Connection: Upgrade
HTTP/2 Data
Server

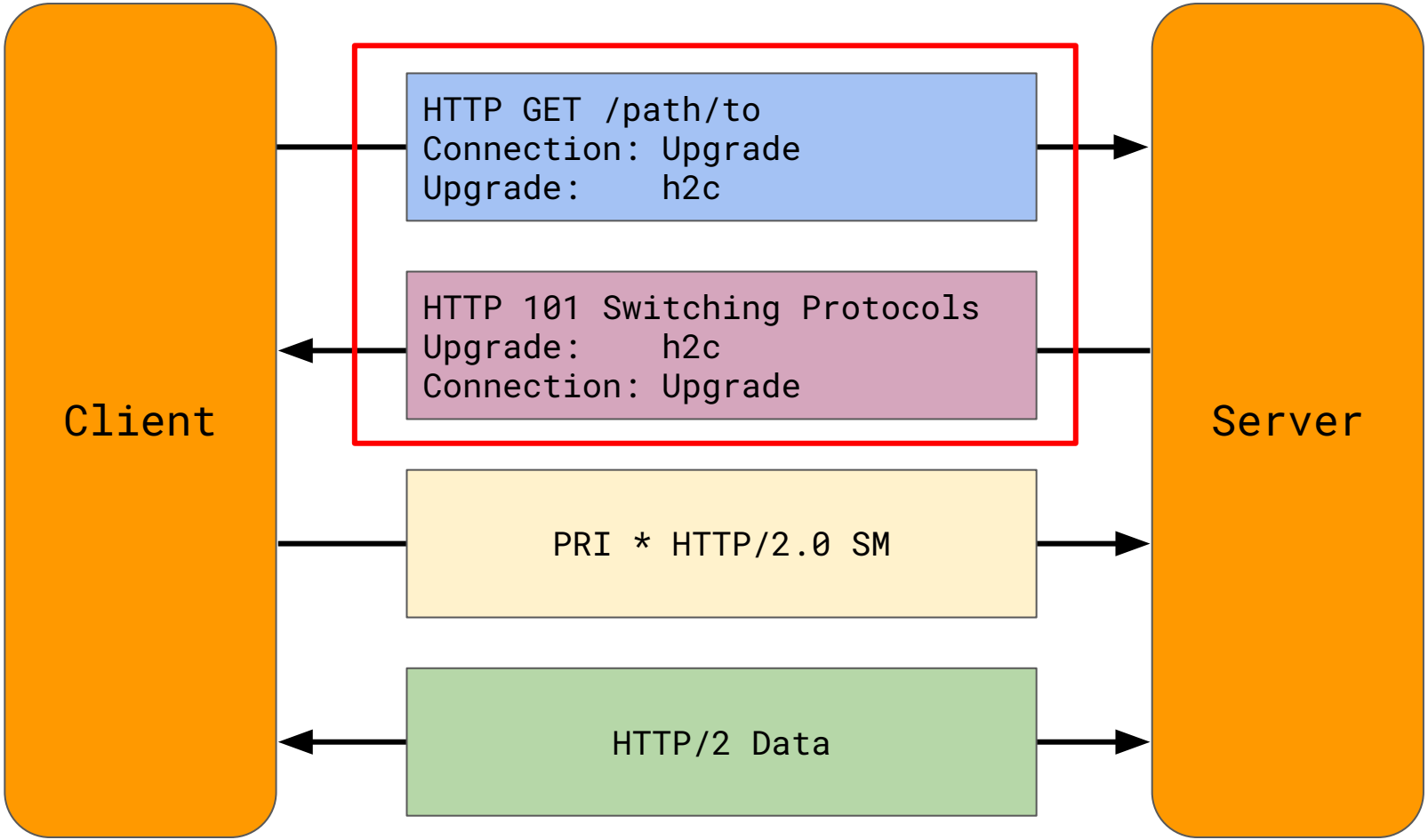
Режим работы “Мамой клянусь”

```
curl --http2-prior-knowledge 'http://127.0.0.1:8085/some/path'
```




HTTP/2 Data

```
▾ HyperText Transfer Protocol 2
  ▾ Stream: Magic
      Magic: PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n
▾ HyperText Transfer Protocol 2
  ▾ Stream: SETTINGS, Stream ID: 0, Length 18
      Length: 18
      Type: SETTINGS (4)
    ▸ Flags: 0x00
      0... .... .... .... .... .... .... .... = Reserved: 0x0
      .000 0000 0000 0000 0000 0000 0000 0000 = Stream Identifier: 0
    ▸ Settings - Max concurrent streams : 100
    ▸ Settings - Initial Windows size : 33554432
    ▸ Settings - Enable PUSH : 0
▾ HyperText Transfer Protocol 2
  ▾ Stream: WINDOW_UPDATE, Stream ID: 0, Length 4
      Length: 4
      Type: WINDOW_UPDATE (8)
```


Client
HTTP GET /path/to
Connection: Upgrade
Upgrade: h2c
HTTP 101 Switching Protocols
Upgrade: h2c
Connection: Upgrade
PRI * HTTP/2.0 SM
HTTP/2 Data
Server

```
> PRI * HTTP/2.0
>
> SM
>
```

Май 2013: **FOO** * HTTP/2.0\r\n\r\n**BA**\r\n\r\n

Июль 2013: **PRI** * HTTP/2.0\r\n\r\n**SM**\r\n\r\n

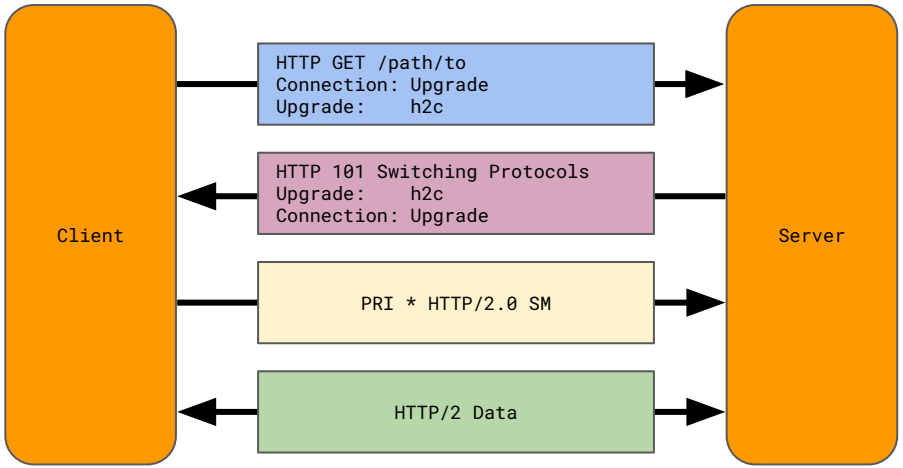
datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2-03#section-3.2
datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2-04#section-3.5

С мая 2013 Эдвард Сноуден начал передавать информацию о программе PRISM.

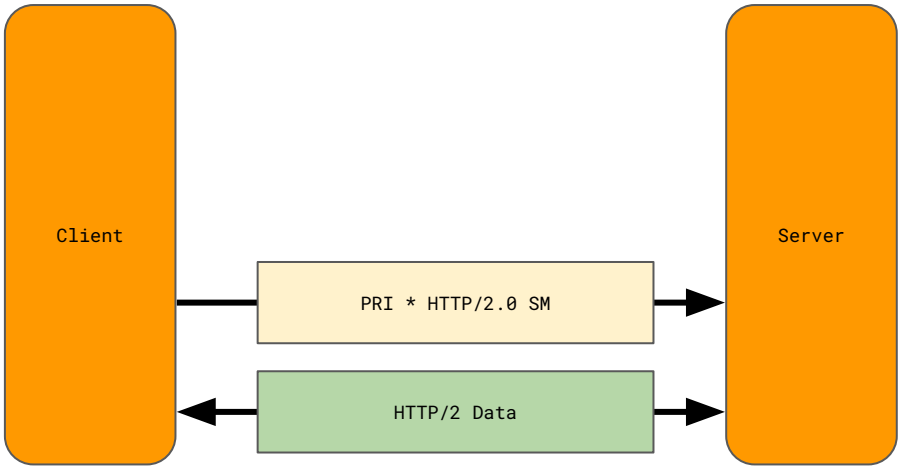
14 июня 2013 года в США ему были предъявлены обвинения в шпионаже и похищении государственной собственности.

ru.wikipedia.org/wiki/Сноуден,_Эдвард

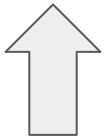
HTTP/2 Upgrade



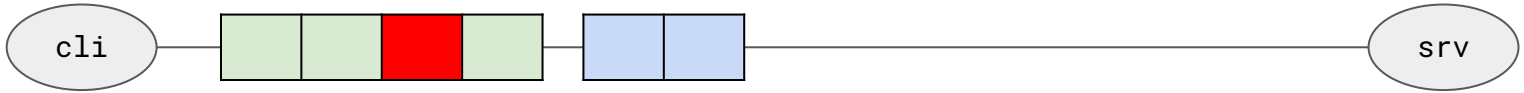
HTTP/2 Prior-knowledge



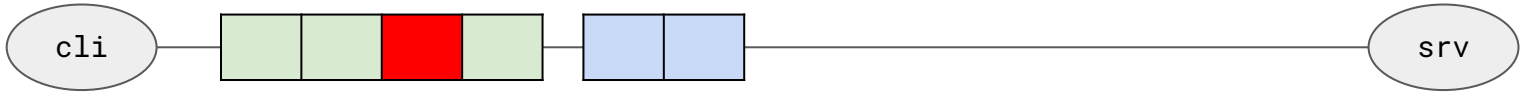
TCP -> HTTP/1 -> HTTP/2



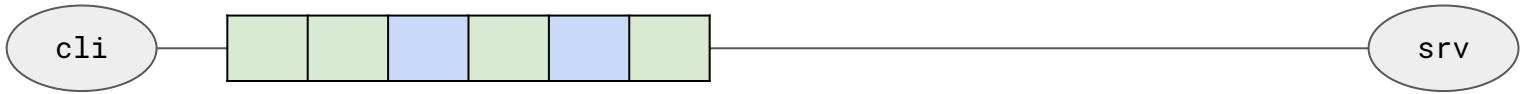
HTTP/1



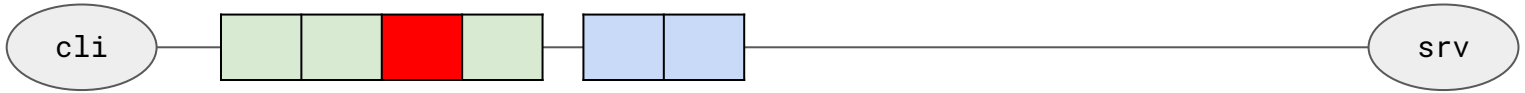
HTTP/1



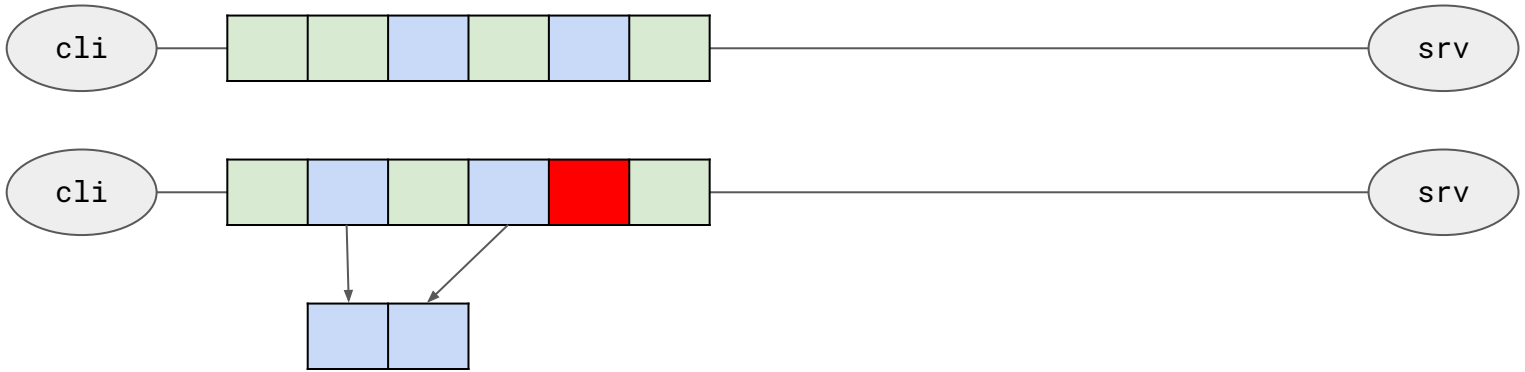
HTTP/2



HTTP/1

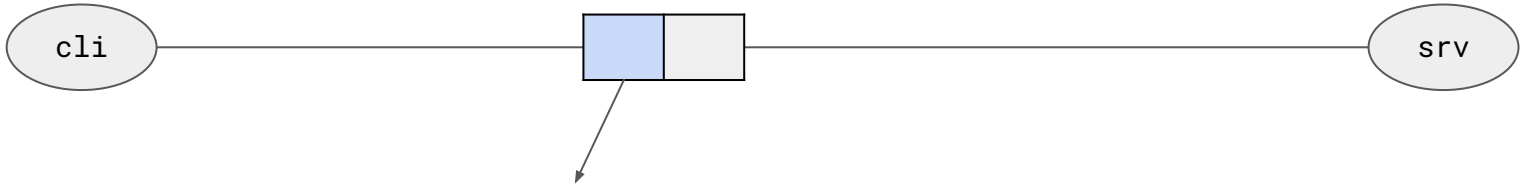


HTTP/2



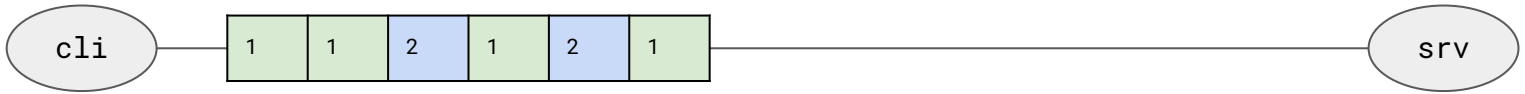

БОЛЬШЕ КОФЕ
БОГУ КОФЕ!

HTTP/2




```
Frame struct {
    PayloadLength uint32
    Type          FrameType
    Flags         FrameFlags
    StreamId      uint32

    Payload       []byte
}

FrameType uint8

FrameFlags uint8
```

HTTP/2




```
Frame struct {
    PayloadLength uint32
    Type          FrameType
    Flags         FrameFlags
    StreamId      uint32

    Payload       []byte
}
```

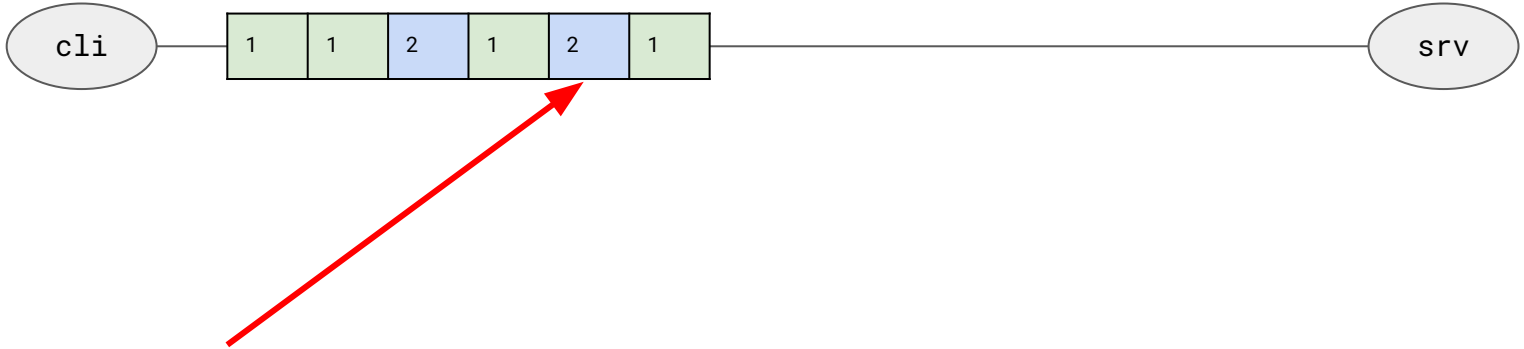
```
FrameHdr struct {
    PayloadLength uint32
    Type          FrameType
    Flags         FrameFlags
    StreamId      uint32
}
```

```
FrameHdr struct {
    PayloadLength uint32
    Type          FrameType
    Flags         FrameFlags
    StreamId      uint32
}

DataFrame struct {
   FrameHdr
   Data []byte
}

HeadersFrame struct {
   FrameHdr
   Exclusive           bool
   StreamIdDepend      uint32
   Weight              uint8
   HeaderBlockFragment []hpack.HeaderField
}
```

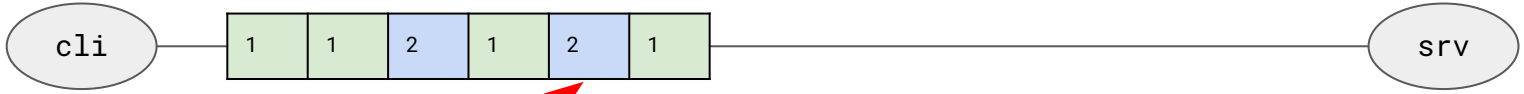
```
SettingsFrame struct {
   FrameHdr
   Params []SettingsFrameParam
}

PingFrame struct {
   FrameHdr
   Data uint64
}

GoawayFrame struct {
   FrameHdr
   LastStreamId uint32
   ErrorCode    ErrorCode
   DebugData    []byte
}

WindowUpdateFrame struct {
   FrameHdr
   WindowSizeIncrement uint32
}
```

datatracker.ietf.org/doc/html/rfc7540#section-6

HTTP/2



HTTP/2



FlagEndStream

```
FrameHdr struct {
    PayloadLength uint32
    Type          FrameType
    Flags         FrameFlags
    StreamId      uint32
}
```

Протокол	Запрос	Ответ
HTTP/1	POST /path HTTP/1 Header1: value1 Header2: value2 body1	200 OK Header3: value3 Header4: value4 body2

Протокол	Запрос	Ответ
HTTP/1	POST /path HTTP/1 Header1: value1 Header2: value2 body1	200 OK Header3: value3 Header4: value4 body2
HTTP/2	[]HeadersFrame []DataFrame	[]HeadersFrame []DataFrame

Протокол	Запрос	Ответ
HTTP/1	POST /path HTTP/1 Header1: value1 Header2: value2 body1	200 OK Header3: value3 Header4: value4 body2
HTTP/2	[]HeadersFrame []DataFrame	[]HeadersFrame []DataFrame

Протокол	Запрос	Ответ
HTTP/1	:method: POST :path: /path Header1: value1 Header2: value2 body1	:status: 200 Header3: value3 Header4: value4 body2
HTTP/2	[]HeadersFrame []DataFrame	[]HeadersFrame []DataFrame

datatracker.ietf.org/doc/html/rfc9113#section-8.3-1

Версия	Запрос	
HTTP/1	POST /path HTTP/1 Content-Length: 12 {“hello”:42}	

HTTP/2	HeadersFrame{
	StreamId: 1
	Flags: FlagEndHeaders
	[:method: POST
	:path: /path
	Content-Length: 12
	]
	}
	DataFrame{
	StreamId: 1
	Flags: FlagEndStream
	PayloadLength: 12
	Data: {“hello”:42}
	}	

Версия	Запрос	Ответ
HTTP/1	POST /path HTTP/1 Content-Length: 12 {“hello”:42}	200 OK Content-Length: 0 Date: Wed, 07 Feb 2024 12:34:56 GMT

HTTP/2	HeadersFrame{
	StreamId: 1
	Flags: FlagEndHeaders
	[:method: POST
	:path: /path
	Content-Length: 12
	]
	}
	DataFrame{
	StreamId: 1
	Flags: FlagEndStream
	PayloadLength: 12
	Data: {“hello”:42}
	}
		HeadersFrame{
		StreamId: 1
		Flags: FlagEndHeaders |
		FlagEndStream
		[:status: 200
		Content-Length: 0
		Date: Wed,…
		]
		}

HTTP/2:

- Мультиплексирование
- Бинарный формат
- Не решает проблему TCP HoL blocking 😅

Запрос	Ответ

HeadersFrame{
StreamId: 1
Flags: FlagEndHeaders |
FlagEndStream
[:method: POST
:path: /path
Content-Length: 0
]
}
 	
HeadersFrame{
StreamId: 3
Flags: FlagEndHeaders |
FlagEndStream
[:method: POST
:path: /path
Content-Length: 0
]
}
	HeadersFrame{
	StreamId: 1
	Flags: FlagEndHeaders |
	FlagEndStream
	[:status: 200
	Content-Length: 0
	Date: Wed,…
	]
	}
	
	HeadersFrame{
	StreamId: 3
	Flags: FlagEndHeaders |
	FlagEndStream
	[:status: 200
	Content-Length: 0
	Date: Wed,…
	]
	}

HTTP/2 HPACK

Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206

httpwg.org/specs/rfc7541.html
blog.cloudflare.com/hpack-the-silent-killer-feature-of-http-2/

HTTP/2 HPACK

`method:  POST  => 3`

Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304

HTTP/2 HPACK

```
method:         POST    => 3
:path:           /path  => 4+"/path"
Content-Length: 0       => 28+"0"


:status:        200     => 8
Content-Length: 0       => 28+0
Date:           Wed,…   => 33+Wed…
```

Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304

HTTP/2 HPACK

```
method:         POST    => 3
:path:           /path  => 4+"/path"
Content-Length: 0       => 28+"0"


:status:        200     => 8
Content-Length: 0       => 28+0
Date:           Wed,…   => 33+Wed…


"POST / HTTP/1\r\n" - 15 байт
3, 4                -  2 байта
```

Index	Header Name	Header Value
1	:authority	
2	:method	GET
3	:method	POST
4	:path	/
5	:path	/index.html
6	:scheme	http
7	:scheme	https
8	:status	200
9	:status	204
10	:status	206
11	:status	304

HTTP/2 HPACK

```
import "golang.org/x/net/http2/hpack"
```

HTTP/2:

- Мультиплексирование
- Бинарный формат
- Не решает проблему TCP HoL blocking 😅
- Сжимает заголовки

```go
func Test_H2C_HTTP2PriorKnowledge(t *testing.T) {
   l, _ := net.Listen("tcp", ":0")
   defer l.Close()

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

   h2s := NewServer()

   h2s.Listen(l)
   go h2s.Serve(ctx)









}
```

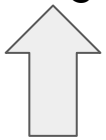
```go
func Test_H2C_HTTP2PriorKnowledge(t *testing.T) {
   l, _ := net.Listen("tcp", ":0")
   defer l.Close()

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

   h2s := NewServer()
   h2s.SetHTTPHandler(httpHandler)
   h2s.Listen(l)
   go h2s.Serve(ctx)









}
```

```go
func httpHandler(req *Request, resp *Response) error {
   response.Headers = map[string]string{
      "content-type": "text/plain; charset=utf-8",
   }

   response.Body = []byte(
       fmt.Sprintf("Hello, %v", request.URI),
   )

   response.Status = 200

   return nil
}
```

```go
func Test_H2C_HTTP2PriorKnowledge(t *testing.T) {
    l, _ := net.Listen("tcp", ":0")
    defer l.Close()

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

    h2s := NewServer()
    h2s.SetHTTPHandler(httpHandler)
    h2s.Listen(l)
    go h2s.Serve(ctx)



    var client http.Client

    uriPath := fmt.Sprintf("/hello?world=%d", rand.Int())
    uri := "http://" + l.Addr().String() + uriPath

    resp, _ := client.Get(uri)

    b, _ := io.ReadAll(resp.Body)

    require.Equal(t, 200, resp.StatusCode)
    require.Equal(t, `Hello, `+uriPath, string(b))
}
```

```go
func httpHandler(req *Request, resp *Response) error {
    response.Headers = map[string]string{
        "content-type": "text/plain; charset=utf-8",
    }

    response.Body = []byte(
        fmt.Sprintf("Hello, %v", request.URI),
    )

    response.Status = 200

    return nil
}
```

TCP -> HTTP/1 -> HTTP/2 -> gRPC



Как подключаться к серверу?

```
curl 'http://127.0.0.1:8085/some/path'
```

```
conn,    := grpc.Dial(l.Addr().String())
defer conn.Close()

client := grpcTestPb.NewGrpcTestServiceClient (conn)
resp, _ := client.Ping(ctx, &grpcTestPb.PingRequest{Val: 42}
```

HTTP/2 сервер

```
:method:        POST
:path:          /grpctest.v1.GrpcTestService/Ping
content-type:   application/grpc
te:             trailers
user-agent:     grpc-go/1.52.0

body: [0 0 0 0 2 8 42]
```

HTTP/2 сервер

```
:method:        POST
:path:          /grpctest.v1.GrpcTestService/Ping
content-type:   application/grpc
te:             trailers
user-agent:     grpc-go/1.52.0

body: [0 0 0 0 2 8 42]
```

```
package grpctest.v1;

service GrpcTestService {
    rpc Ping(PingRequest) returns (PingResponse);
}
```

HTTP/2 body

```
body: [0 0 0 0 2 8 42]
```

HTTP/2 body

```
body: [0 0 0 0 2 8 42]
```

```
type GRPCPayload struct {
    CompressedFlag byte
    MessageLength  uint32
    Message        []byte
}
```

```
proto message: [8 42]    // grpcTestPb.PingRequest{Val: 42}
```

gRPC payload

19:45 – 20:30

ДОКЛАД

gRPC Middleware в Go как способ модифицировать все запросы в одном месте

gRPC Middleware в Go: что это и зачем; проксирование gRPC-запросов: аутентификация, логирование, валидация и фильтрация; модификация запросов: можем ли и зачем? Проблемы поддержки кода; модификация…

Александр Шакмаев
Cloud.ru

RU

Libraries || Tools

13:00 – 13:45

ДОКЛАД

protobuf в Go

Расскажу, почему мы в компании используем `protobuf`, приведу наглядные примеры использования. Поговорим про наиболее типичные проблемы и боли, вызванные нарушением рекомендаций по обновлению…

Владислав Сидоров
Ozon

RU

Libraries || Tools

Go gRPC	client := grpcTestPb.NewGrpcTestServiceClient(conn) resp, _ := client.Ping(ctx, &grpcTestPb.PingRequest{Val: 42})

Go gRPC	`client := grpcTestPb.NewGrpcTestServiceClient(conn)`
	
	`resp, _ := client.Ping(ctx, &grpcTestPb.PingRequest{Val: 42})`

HTTP/2	`HeadersFrame{`
	`  StreamId: 1`
	`  Flags: FlagEndHeaders`
	`  PayloadLength: 80`
	`  [ :method: POST`
	`  :path: /grpctest.v1.FastGrpcTestService/Ping`
	`  content-type: application/grpc`
	`  user-agent: grpc-go/1.52.0`
	`  te: trailers`
	`  ]`
	`}`
	`DataFrame{`
	`  StreamId: 1`
	`  Flags: FlagEndStream`
	`  PayloadLength: 7`
	`  Data: [0 0 0 0 2 8 42]`
	`}`

Go gRPC	

```
var resp grpcTestPb.PingResponse
resp.Val = req.Val * 2
_, _ = proto.Marshal(&resp)
```

Go gRPC	`var resp grpcTestPb.PingResponse`
	`resp.Val = req.Val * 2`
	`_, _ = proto.Marshal(&resp)`

HTTP/2	`HeadersFrame{`
	`  StreamId: 1`
	`  Flags:  FlagEndHeaders`
	`  [ :status:  200`
	`  content-type:  application/grpc+proto`
	`  ]`
	`}`
	`DataFrame{`
	`  StreamId: 1`
	`  Flags:  0`
	`  Data:  [0 0 0 0 2 8 84]`
	`}`
	`HeadersFrame{`
	`  StreamId: 1`
	`  Flags:  FlagEndStream | FlagEndHeaders`
	`  [ grpc-status: 0 // google.golang.org/grpc/codes.OK`
	`  ]`
	`}`

redbubble.com/i/art-print/Golang-Gopher-Developer-and-His-Soul-by-clgtart/95257905.1G4ZT

А что же gRPC стримы?

А что же gRPC стримы?

```
service GrpcTestService {
  rpc Sub(stream SubRequest) returns (stream SubResponse);
}

message SubRequest {
  int64 pubInterval = 1;
}

message SubResponse {
  int64 currentTime = 1;
}
```

```
_ = subCli.Send(&grpcTestPb.SubRequest{PubInterval: int64(time.Second)})

for {
   msg, _ := subCli.Recv()
   log.Println(time.Unix(0, msg.CurrentTime))
}
```

Go gRPC	client := grpcTestPb.NewGrpcTestServiceClient(conn) subCli, _ := client.Sub(ctx)

Go gRPC	`client := grpcTestPb.NewGrpcTestServiceClient(conn)`
	
	`subCli, _ := client.Sub(ctx)`

HTTP/2	`HeadersFrame{`
	`  StreamId: 1`
	`  Flags:  FlagEndHeaders`
	`  [  :method:  POST`
	`  :path:  /grpctest.v1.GrpcTestService/Sub`
	`  content-type:  application/grpc`
	`  user-agent:  grpc-go/1.52.0`
	`  te:  trailers`
	`  ]`
	`}`
	`А и всё :)`

Go gRPC	`_ = subCli.Send(&grpcTestPb.SubRequest{ PubInterval: int64(time.Second), })`

Go gRPC	`_ = subCli.Send(&grpcTestPb.SubRequest{`
	`  PubInterval: int64(time.Second),`
	`})`

HTTP/2	`DataFrame{`
	`  StreamId:  1`
	`  Flags:  0`
	`  Data:  [0 0 0 0 6 8 128 148 235 220 3]`
	`}`

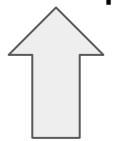
HTTP/2 ответ

```
HeadersFrame{
    StreamId: 1
    Flags:    FlagEndHeaders
    [  :status:        200
       content-type:   application/grpc+proto
    ]
}
DataFrame{
    StreamId:      1
    Flags:         0
    Data:          [0 0 0 0 10 8 174 198 172 253 184 219 242 217 23]
}
DataFrame{
    StreamId:      1
    Flags:         0
    Data:          [0 0 0 0 10 8 228 160 177 218 188 219 242 217 23]
}
...
HeadersFrame{
    StreamId: 1
    Flags:    FlagEndStream | FlagEndHeaders
    [  grpc-status: 0
    ]
}
```

gRPC:

- Unary call - обычный HTTP/2 запрос (:path + body payload)
- gRPC стримы работают поверх HTTP/2 стримов
- Никак не понять, что это streaming запрос
- protobuf опционален

TCP -> HTTP/1 -> HTTP/2 -> gRPC -> Application

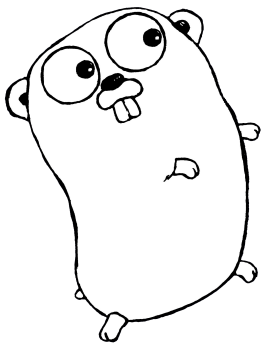


Выводы

```
panic: runtime error: invalid memory address or nil pointer dereference
```

Вопросы?

github.com/atercattus/tinygrpc



@atercattus