Ruby, Ractor, QUIC

unasuke (Yusuke Nakamura)

RubyKaigi Takeout 2021

2021-09-11

Self introduction

- Name: unasuke (Yusuke Nakamura)
- Work: freelance Web app developer
  - OOParts (Cloud Gaming Service) backend dev
- Itamae gem maintainer, Kaigi on Rails staff
- GitHub https://github.com/unasuke
- Mastodon https://mstdn.unasuke.com/@unasuke
- Twitter https://twitter.com/yu_suke1994

Next generation Web

widespreading Web usage

- Cloud Gaming
- Video meeting
- Streaming media

We want the Web to be faster!

Cloud Gaming requiement

"real-time bidirectional communication"

1. Send player's input to the game on the cloud machine
2. The game render the input result
3. Send result to the player's screen (web browser)

Repeat the above process as soon as possible for comfortable game play.

(Same as remote working collaboration)

Way to real-time bidirectional communication on the Web

- Now
  - WebRTC
  - WebSocket
- Future?
  - WebTransport
    - a new server-client protocol over the HTTP/3
    - use UDP
    - status: draft → https://datatracker.ietf.org/group/webtrans/documents/

HTTP/3

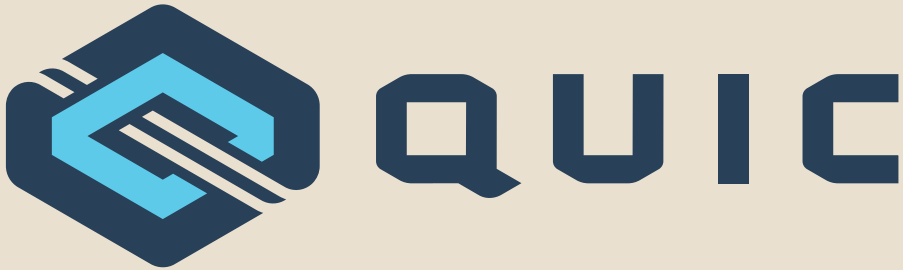
A new hyper text transfer protocol (draft)

> *This document defines HTTP/3, a mapping of HTTP semantics over the QUIC transport protocol*
>
> *https://datatracker.ietf.org/doc/draft-ietf-quic-http/*

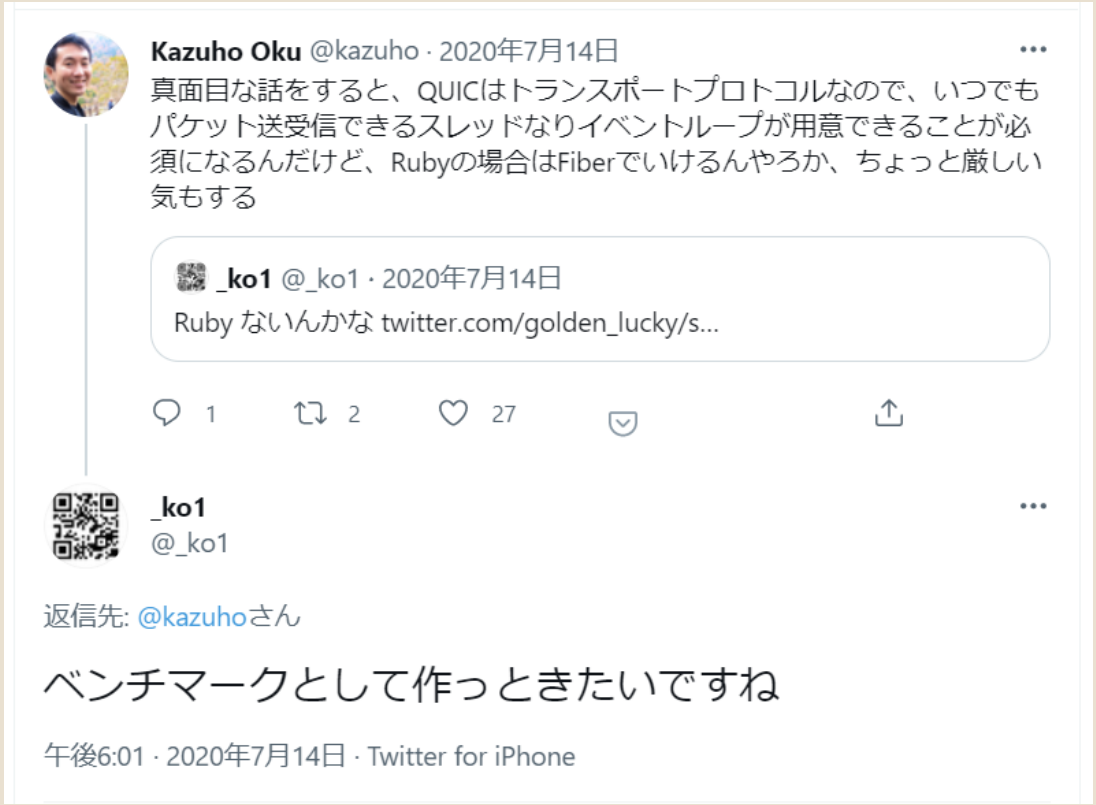
"QUIC" ? What is this?

QUIC

- standardized at 2021-05-27 by IETF
- More faster than HTTP/2
- Using UDP, not TCP



Tweet by ko1



Kazuho Oku @kazuho · 2020年7月14日

真面目な話をすると、QUICはトランスポートプロトコルなので、いつでもパケット送受信できるスレッドなりイベントループが用意できることが必須になるんだけど、Rubyの場合はFiberでいけるんやろか、ちょっと厳しい気もする

> **_ko1** @_ko1 · 2020年7月14日
> Ruby ないんかな twitter.com/golden_lucky/s...

1 2 27

_ko1
@_ko1

返信先: @kazuhoさん

ベンチマークとして作っときたいですね

午後6:01 · 2020年7月14日 · Twitter for iPhone

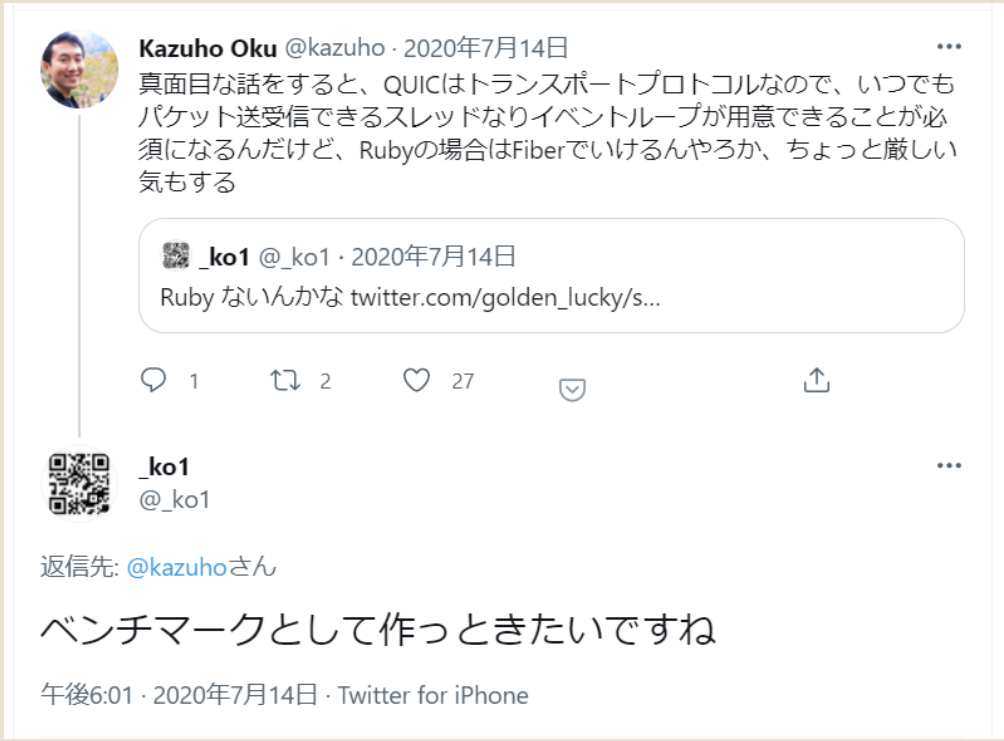
https://twitter.com/_ko1/status/1282963500583628800

Tweet by ko1

two people

- @_ko1 (Koichi Sasada)
  - Ruby core commiter
  - Ractor author
- @kazuho (Kazuho Oku)
  - author of h2o
  - QUIC RFC contributor

Sounds interesting!



What, Why QUIC

- HTTP uses TCP/IP
- TCP is reliable, but slow
  - Oriented data transfer
  - Retransmission of lost packets
  - Congestion control
- UDP is unreliable, but fast
  - realtime transfer e.g. audio

What, Why QUIC - TCP's problem

- Three way handshake
  - additional TLS handshake (at secure communication)
- Head-of-Line Blocking (HoLB)

> *QUIC is a new latency-reducing, reliable, and secure internet transport protocol*
>
> *https://www.fastly.com/blog/quic-is-now-rfc-9000*

Is there a benefit?

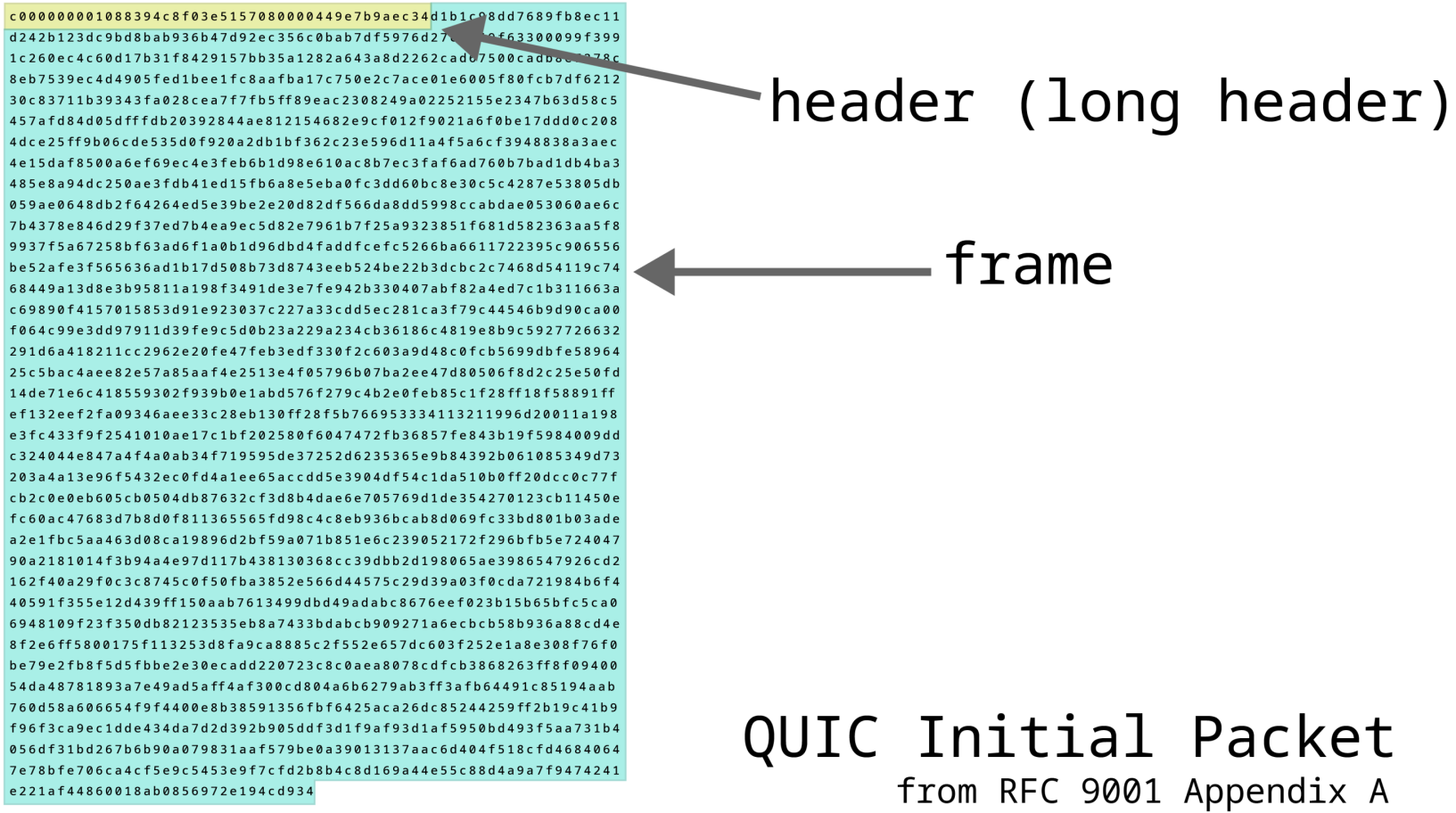
Why imprement QUIC on Ruby?

1. For my study and interest
2. For Ractor
   - There are still few examples of Ractor
   - helps evaluation and improvement of Ractor
     - e.g. https://github.com/mame/optcarrot

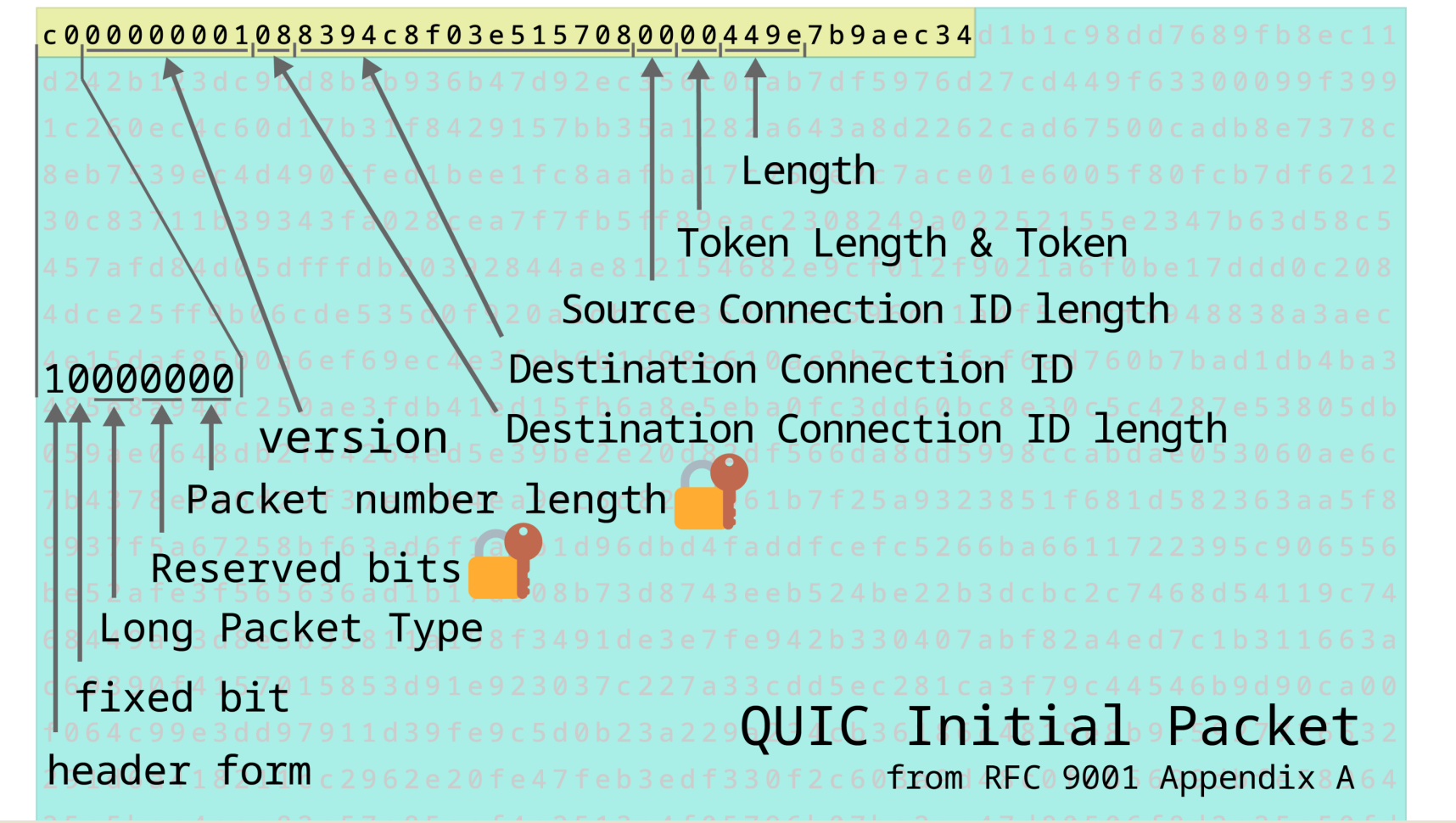
QUIC - Initial Packet (RFC9001 Appendix A)

```
c000000001088394c8f03e5157080000 449e7b9aec34d1b1c98dd7689fb8ec11
d242b123dc9bd8bab936b47d92ec356c 0bab7df5976d27cd449f63300099f399
1c260ec4c60d17b31f8429157bb35a12 82a643a8d2262cad67500cadb8e7378c
8eb7539ec4d4905fed1bee1fc8aafba1 7c750e2c7ace01e6005f80fcb7df6212
30c83711b39343fa028cea7f7fb5ff89 eac2308249a02252155e2347b63d58c5
457afd84d05dfffdb20392844ae81215 4682e9cf012f9021a6f0be17ddd0c208
4dce25ff9b06cde535d0f920a2db1bf3 62c23e596d11a4f5a6cf3948838a3aec
4e15daf8500a6ef69ec4e3feb6b1d98e 610ac8b7ec3faf6ad760b7bad1db4ba3
485e8a94dc250ae3fdb41ed15fb6a8e5 eba0fc3dd60bc8e30c5c4287e53805db
059ae0648db2f64264ed5e39be2e20d8 2df566da8dd5998ccabdae053060ae6c
7b4378e846d29f37ed7b4ea9ec5d82e7 961b7f25a9323851f681d582363aa5f8
9937f5a67258bf63ad6f1a0b1d96dbd4 faddfcefc5266ba6611722395c906556
be52afe3f565636ad1b17d508b73d874 3eeb524be22b3dcbc2c7468d54119c74
68449a13d8e3b95811a198f3491de3e7 fe942b330407abf82a4ed7c1b311663a
c69890f4157015853d91e923037c227a 33cdd5ec281ca3f79c44546b9d90ca00
f064c99e3dd97911d39fe9c5d0b23a22 9a234cb36186c4819e8b9c5927726632
291d6a418211cc2962e20fe47feb3edf 330f2c603a9d48c0fcb5699dbfe58964
25c5bac4aee82e57a85aaf4e2513e4f0 5796b07ba2ee47d80506f8d2c25e50fd
14de71e6c418559302f939b0e1abd576 f279c4b2e0feb85c1f28ff18f58891ff
ef132eef2fa09346aee33c28eb130ff2 8f5b766953334113211996d20011a198
e3fc433f9f2541010ae17c1bf202580f 6047472fb36857fe843b19f5984009dd
c324044e847a4f4a0ab34f719595de37 252d6235365e9b84392b061085349d73
203a4a13e96f5432ec0fd4a1ee65accd d5e3904df54c1da510b0ff20dcc0c77f
cb2c0e0eb605cb0504db87632cf3d8b4 dae6e705769d1de354270123cb11450e
fc60ac47683d7b8d0f811365565fd98c 4c8eb936bcab8d069fc33bd801b03ade
a2e1fbc5aa463d08ca19896d2bf59a07 1b851e6c239052172f296bfb5e724047
90a2181014f3b94a4e97d117b4381303 68cc39dbb2d198065ae3986547926cd2
162f40a29f0c3c8745c0f50fba3852e5 66d44575c29d39a03f0cda721984b6f4
40591f355e12d439ff150aab7613499d bd49adabc8676eef023b15b65bfc5ca0
6948109f23f350db82123535eb8a7433 bdabcb909271a6ecbcb58b936a88cd4e
8f2e6ff5800175f113253d8fa9ca8885 c2f552e657dc603f252e1a8e308f76f0
be79e2fb8f5d5fbbe2e30ecadd220723 c8c0aea8078cdfcb3868263ff8f09400
54da48781893a7e49ad5aff4af300cd8 04a6b6279ab3ff3afb64491c85194aab
760d58a606654f9f4400e8b38591356f bf6425aca26dc85244259ff2b19c41b9
f96f3ca9ec1dde434da7d2d392b905dd f3d1f9af93d1af5950bd493f5aa731b4
056df31bd267b6b90a079831aaf579be 0a39013137aac6d404f518cfd4684064
7e78bfe706ca4cf5e9c5453e9f7cfd2b 8b4c8d169a44e55c88d4a9a7f9474241
e221af44860018ab0856972e194cd934
```

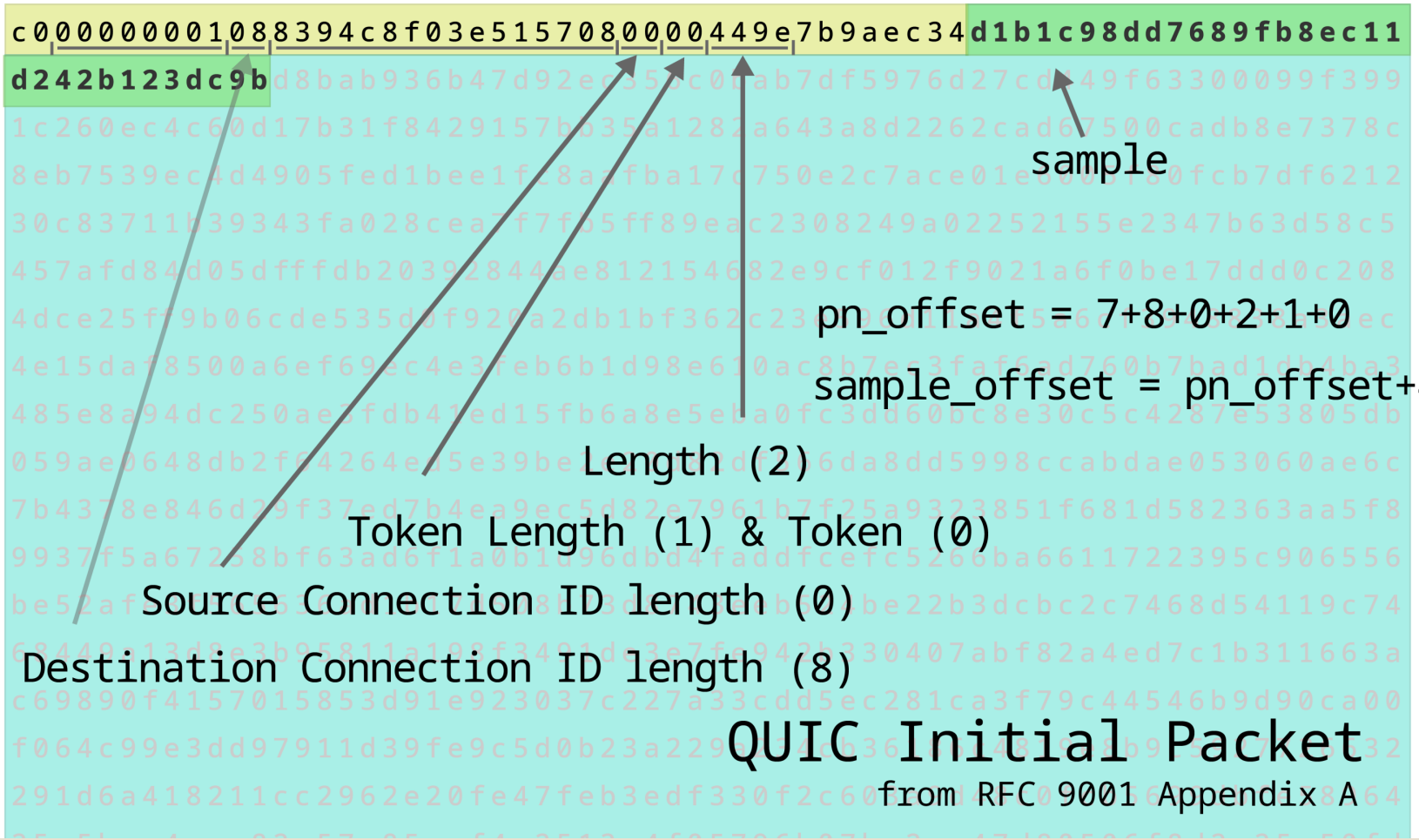
QUIC - reading Initial Packet



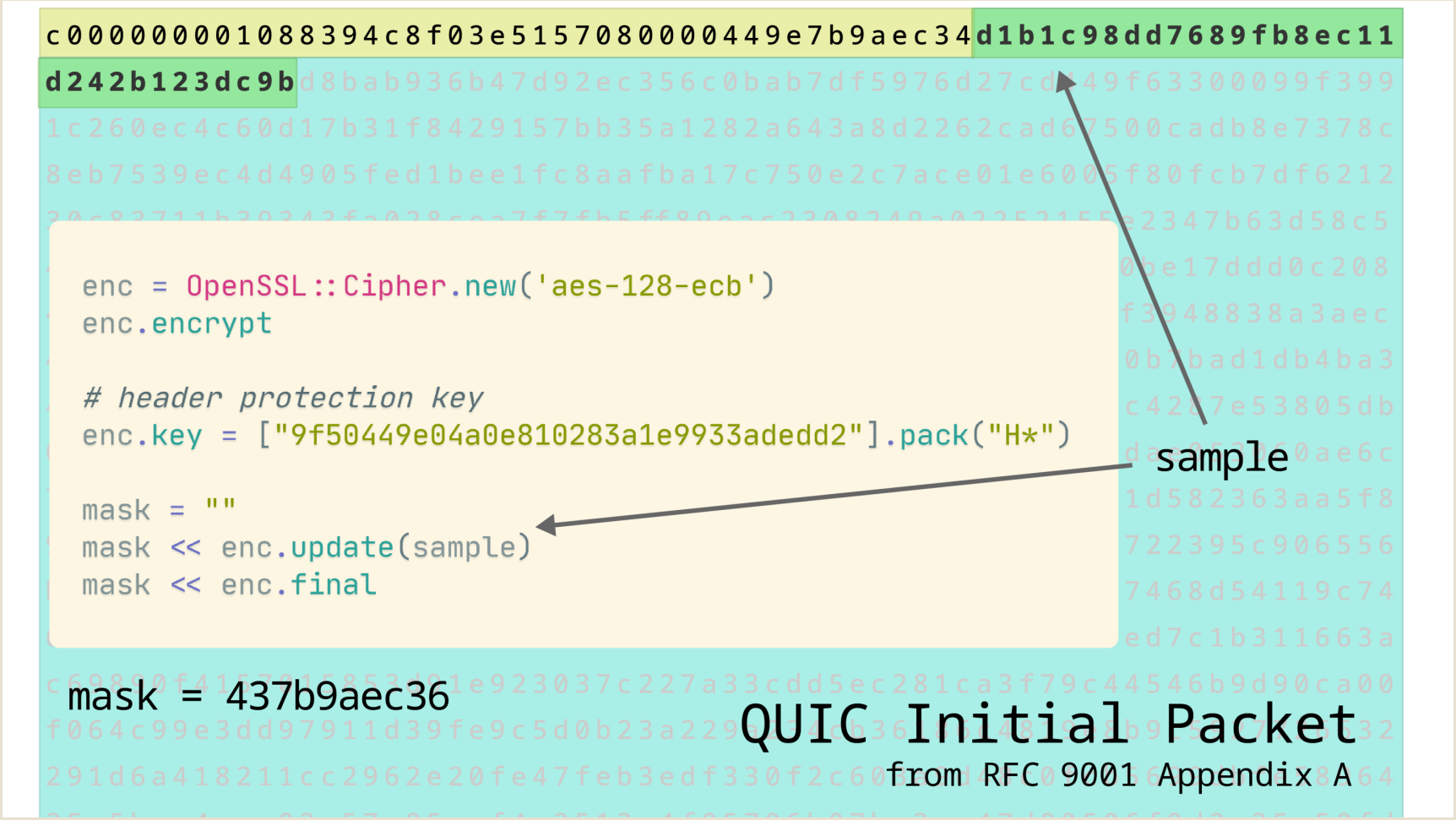
QUIC - reading Initial Packet



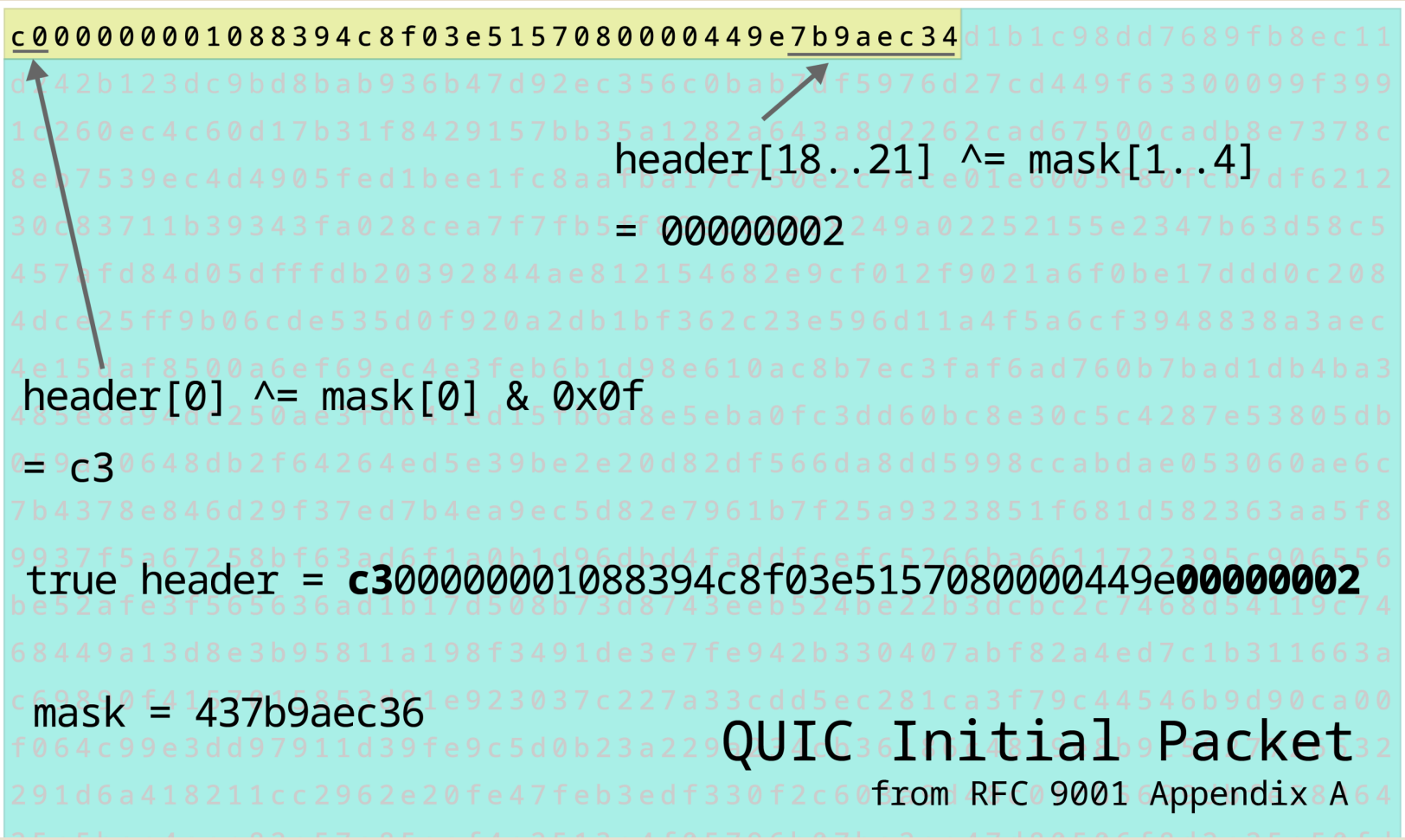
QUIC - reading Initial Packet



QUIC - reading Initial Packet



QUIC - reading Initial Packet



QUIC - reading Initial Packet

QUIC Initial Packet

from RFC 9001 Appendix A

n月刊ラムダノート Vol.2, No.1(2020)

#1 パケットの設計から見るQUIC（西田佳史）

https://www.lambdanote.com/collections/frontpage/products/nmonthly-vol-2-no-1-2020

QUIC - reading Initial Packet by Ruby

```ruby
class QUICInitialPacket < BinData::Record
  endian :big
  bit1 :header_form, asserted_value: 1
  bit1 :fixed_bit, asserted_value: 1
  bit2 :long_packet_type, asserted_value: 0
  bit2 :reserved_bit
  bit2 :packet_number_length
  bit32 :version
  bit8 :destination_connection_id_length
  bit :destination_connection_id, nbits: lambda { destination_connection_id_length * 8 }
  bit8 :source_connection_id_length
  bit :source_connection_id, nbits: lambda { source_connection_id_length * 8 }
  bit2 :token_two_most_significant_bits
  bit :token_length, nbits: lambda { tms(token_two_most_significant_bits) }
  string :token, read_length: lambda { token_length }
  bit2 :length_two_most_significant_bits
  bit :length_length, nbits: lambda { tms(length_two_most_significant_bits) }
  bit :packet_number, nbits: lambda { (packet_number_length + 1) * 8 }
  string :payload, read_length: lambda { length_length - (packet_number_length + 1) }
end
```

QUIC - reading Initial Packet by Ruby

```ruby
class QUICInitialPacket < BinData::Record
  endian :big
  bit1 :header_form, asserted_value: 1
  bit1 :fixed_bit, asserted_value: 1
  bit2 :long_packet_type, asserted_value: 0
  bit2 :reserved_bit
  bit2 :packet_number_length
  bit32 :version
  bit8 :destination_connection_id_length
  bit :destination_connection_id, nbits: lambda { destination_connection_id_length * 8 }
  bit8 :source_connection_id_length
  bit :source_connection_id, nbits: lambda { source_connection_id_length * 8 }
  bit2 :token_two_most_significant_bits
  bit :token_length, nbits: lambda { tms(token_two_most_significant_bits) }
  string :token, read_length: lambda { token_length }
  bit2 :length_two_most_significant_bits
  bit :length_length, nbits: lambda { tms(length_two_most_significant_bits) }
  bit :packet_number, nbits: lambda { (packet_number_length + 1) * 8 }
  string :payload, read_length: lambda { length_length - (packet_number_length + 1) }
end
```

It causes Ractor::IsolationError when use in not main Ractor

QUIC - reading Initial Packet by Ruby

```
irb(main):008:0> pp raw_packet
"\xC0\x00\x00\x00\x01\b\x83\x94\xC8\xF0>QW\b\x00\x00D\x9E{\x9A\xEC4\xD1\xB1\xC9\x8D\xD7h\x9F\xB8\xEC\x11\xD2B\xB1#\xDC\x9B
\xD8\xBA\xB96\xB4}\x92\xEC5l\v\xAB}\xF5\x97m'\xCDD\x9Fc0\x00\x99\xF3\x99\x1C&\x0E\xC4\xC6\r\x17\xB3\x1F\x84)\x15{\xB3Z\x12
\x82\xA6C\xA8\xD2&,\xADgP\f\xAD\xB8\xE77\x8C\x8E\xB7S\x9E\xC4\xD4\x90_\xED\e\xEE\x1F\xC8\xAA\xFB\xA1¦u\x0E,z\xCE\x01\xE6\x
00_\x80\xFC\xB7\xDFb\x120\xC87\x11\xB3\x93C\xFA\x02\x8C\xEA\x7F\x7F\xB5\xFF\x89\xEA\xC20\x82I\xA0\"R\x15^#G\xB6=X\xC5Ez\xF
D\x84\xD0]\xFF\xFD\xB2\x03\x92\x84J\xE8\x12\x15F\x82\xE9\xCF\x01/\x90!\xA6\xF0\xBE\x17\xDD\xD0\xC2\bM\xCE%\xFF\x9B\x06\xCD
\xE55\xD0\xF9 \xA2\xDB\e\xF3b\xC2>Ym\x11\xA4\xF5\xA6\xCF9H\x83\x8A:\xECN\x15\xDA\xF8P\n" +
"n\xF6\x9E\xC4\xE3\xFE\xB6\xB1\xD9\x8Ea\n" +
"\xC8\xB7\xEC?\xAFj\xD7`\xB7\xBA\xD1\xDBK\xA3H^\x8A\x94\xDC%\n" +
"\xE3\xFD\xB4\x1E\xD1_\xB6\xA8\xE5\xEB\xA0\xFC=\xD6\v\xC8\xE3\f\\B\x87\xE58\x05\xDB\x05\x9A\xE0d\x8D\xB2\xF6Bd\xED^9\xBE.
\xD8-\xF5f\xDA\x8D\xD5\x99\x8C\xCA\xBD\xAE\x050`\xAEl{Cx\xE8F\xD2\x9F7\xED{N\xA9\xEC]\x82\xE7\x96\e\x7F%\xA928Q\xF6\x81\xD
5\x826:\xA5\xF8\x997\xF5\xA6rX\xBFc\xADo\x1A\v\x1D\x96\xDB\xD4\xFA\xDD\xFC\xEF\xC5&k\xA6a\x17\"9\\\x90eV\xBER\xAF\xE3\xF5e
cj\xD1\xB1}P\x8Bs\xD8t>\xEBRK\xE2+=\xCB\xC2\xC7F\x8DT\x11\x9CthD\x9A\x13\xD8\xE3\xB9X\x11\xA1\x98\xF3I\x1D\xE3\xE7\xFE\x94
+3\x04\a\xAB\xF8*N\xD7\xC1\xB3\x11f:\xC6\x98\x90\xF4\x15p\x15\x85=\x91\xE9#\x03¦\"z3\xCD\xD5\xEC(\x1C\xA3\xF7\x9CDTk\x9D\x
90\xCA\x00\xF0d\xC9\x9E=\xD9y\x11\xD3\x9F\xE9\xC5\xD0\xB2:\"\x9A#L\xB3a\x86\xC4\x81\x9E\x8B\x9CY'rf2)\x1DjA\x82\x11\xCC)b\
```

I cannot read this, and ruby cannot read it bit by bit.

QUIC - reading Initial Packet by Ruby

How to read each "bit" in Ruby?

```
"A" ← 0x41 (ASCII-8BIT)
"A" ← 0b01000001 (ASCII-8BIT)
"A" → ? → "01000001"
```

QUIC - reading Initial Packet by Ruby

Use "Array#pack", "String#unpack1"

```
"A".unpack1("B*") # => "01000001"
["01000001"].pack("B*") # => "A"
```

Now, we can each "bit" by Ruby!

Handling packet by Ractor

How do we handling UDP packet by Ractor?

Handling packet by Ractor - think about QPACK

- QPACK (Header Compression for HTTP/3)
  - Static Table
    - "content-encoding gzip" (values that appear many times)
  - Dynamic Table
    - per client-server
    - e.g. user-agent

→ Create ractor per client that has a dynamic table state...🤔

Handling packet by Ractor - UDP echo server by Ractor

- 3 classes
  - Server class
  - Router class
  - Connection class

UDP echo server by Ractor - Server class

```ruby
class Server
  def initialize
    @socket = UDPSocket.new
    @router = Router.new.ractor
  end

  def run
    @socket.bind("0.0.0.0", 8080)
    Ractor.new(@socket, @router) do |socket, router|
      loop do
        begin
          raw_packet, addr = socket.recvmsg_nonblock(2000)
        rescue IO::WaitReadable
          retry
        end
        router.send [raw_packet, addr.to_sockaddr, socket] if raw_packet
      end
    end
    puts "Start server"
    sleep
  rescue Interrupt
    puts "Stop server"
    exit
  end
end
```

UDP echo server by Ractor - Router class

```
class Router
  def initialize
    @ractor = Ractor.new() do
      loop do
        packet, addr, socket = Ractor.receive
        Connection.new.ractor.send([packet, addr, socket])
      end
    end
  end

  def ractor
    @ractor
  end
end
```

UDP echo server by Ractor - Connection class

```ruby
class Connection
  def initialize
    @ractor = Ractor.new() do
      packet, addr, socket = Ractor.receive
      socket.send(packet, 0, addr)
    end
  end

  def ractor
    @ractor
  end
end
```

benchmarking by udpbench

https://github.com/unasuke/udpbench

Improvised UDP benchmark tool written Go.

Send-receive UDP packet that contain UUIDv4 from goroutines.

Result of benchmark UDP echo server by Ractor

```
$ ./udpbench --count 100 --parallelism 100
Total request count : 10000
Total request time : 10m48.446641s
Time per packets : 64.844664ms
Failed count : 0
```

simple UDP echo server

```ruby
require 'socket'

socket = UDPSocket.new
socket.bind("0.0.0.0", 8080)

loop do
  begin
    raw_packet, addr = socket.recvmsg_nonblock(500)
  rescue IO::WaitReadable
    retry
  end
  if raw_packet
    socket.send(raw_packet, 0, addr)
  end
end
```

Result of benchmark simple UDP echo server

```
$ ./udpbench --count 100 --parallelism 100
Total request count : 10000
Total request time : 7.5696799s
Time per packets : 756.967µs
Failed count : 0
```

x85 faster than Ractor impl 🤷‍♂️❓

Let's decrypt QUIC packet in Ractor

```
#<Thread:0x000055b3e611fce8 run> terminated with exception (report_on_exception is t
rue):
#<Thread:0x000055b3e611f8b0 run> terminated with exception (report_on_exception is t
rue):
tmp/stresstest.rb:97:in `initialize'tmp/stresstest.rb:97:in `initialize': : ractor u
nsafe method called from not main ractorractor unsafe method called from not main ra
ctor ( (Ractor::UnsafeError)Ractor::UnsafeError
)
        from tmp/stresstest.rb:97:in `new'
        from tmp/stresstest.rb:97:in `new'
        from tmp/stresstest.rb:97:in `block in initialize'
        from tmp/stresstest.rb:97:in `block in initialize'
```

Let's decrypt QUIC packet in Ractor

```
#<Thread:0x000055b3e611fce8 run> terminated with exception (report_on_exception is t
rue):
#<Thread:0x000055b3e611f8b0 run> terminated with exception (report_on_exception is t
rue):
tmp/stresstest.rb:97:in `initialize'tmp/stresstest.rb:97:in `initialize': : ractor u
nsafe method called from not main ractorractor unsafe method called from not main ra
ctor ( (Ractor::UnsafeError)Ractor::UnsafeError
)
        from tmp/stresstest.rb:97:in `new'
        from tmp/stresstest.rb:97:in `new'
        from tmp/stresstest.rb:97:in `block in initialize'
        from tmp/stresstest.rb:97:in `block in initialize'
```

Failed by OpenSSL::Cipher.new 😵

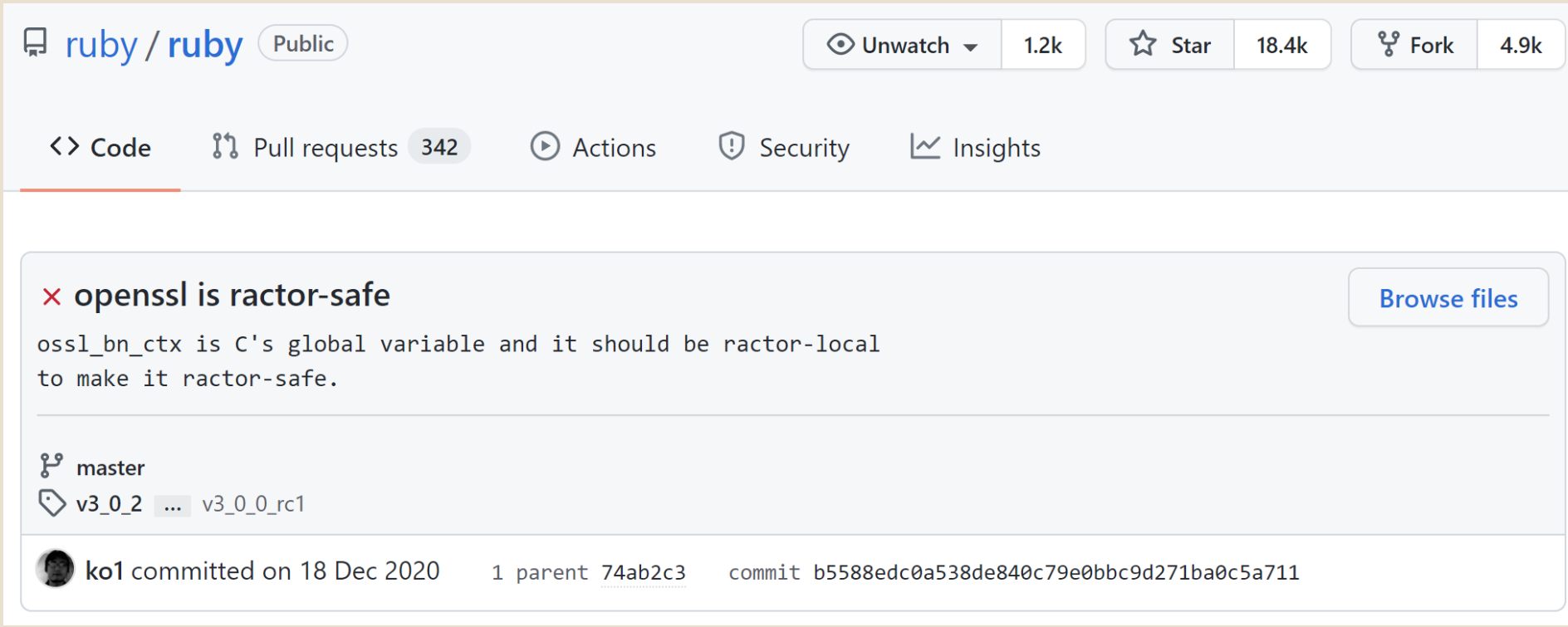
Can OpenSSL rb_ext_ractor_safe(true) ...?

Wait, I heard about yesterday...

> *'Standard libralies are already ractor-safe' by ko1*

I heard about that in "Ruby Committers vs the World" yesterday.

OpenSSL is `rb_ext_ractor_safe(true)`



Wow...

Conclusion

There are two problems

- Libraies that cannot use in not main Ractor
  - useful gem → imprement myself (e.g. bindata)
  - core lib → DEEP DIVE to fix this (e.g. openssl)
- Need many many code to imprement QUIC
  - Mozilla's Neqo : 50,000 LoC (Rust) https://github.com/mozilla/neqo
  - aioquic : 17,000 LoC (Python) https://github.com/aiortc/aioquic
  - Ruby : ?