

# Ruby Implementation of QUIC: Progress and Challenges

unasuke (Yusuke Nakamura)

*RubyKaigi 2023*  
2023-05-12



**RubyKaigi 2023**

# Self introduction

---

- Name: unasuke (Yusuke Nakamura)
- Work: freelance Web app developer @ Japan
- Itamae gem maintainer, Kaigi on Rails Organizer
- GitHub <https://github.com/unasuke>
- Mastodon <https://mstdn.unasuke.com/@unasuke>
- Twitter [https://twitter.com/yu\\_suke1994](https://twitter.com/yu_suke1994)



# Ruby Association Grant

---

## Porting the QUIC protocol implementation from other languages and creating original implementation in Ruby

### Project Summary

The QUIC, an internet protocol standardized in 2021, is spreading rapidly, and implementations in various programming languages are growing. Some programming languages have multiple implementations, but Ruby has no publicly available implementation. This project aims to create a Ruby implementation of the QUIC protocol eventually. As an initial step, we will port aioquic, a Python implementation of QUIC, to Ruby to establish guidelines and knowledge for QUIC implementation.

### Applicant Name

unasuke (Yusuke Nakamura)

<https://www.ruby.or.jp/en/news/20221027>

# What is QUIC?

---

- UDP-based communication protocol
- HTTP/3 uses QUIC
  - Faster than HTTP/2 (TCP)



image from <https://github.com/quicwg/wg-materials>

# What is QUIC? - Diagram by Robin Marx

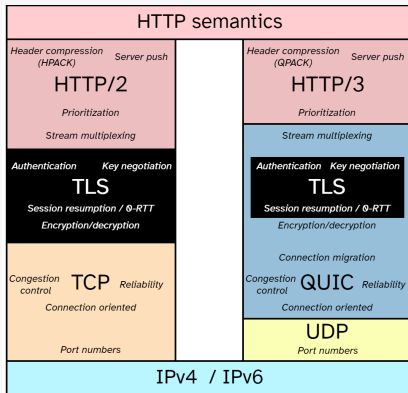


image from <https://github.com/rmarx/h3-protocol-stack>

# My talks

---

- RubyKaigi Takeout 2021
  - "Ruby, Ractor, QUIC"
- RubyKaigi 2022 (Tsu)
  - "Ruby, Ractor, QUIC"
- Now (2023, Matsumoto)

## Ruby, Ractor, QUIC

unasuke (Yusuke Nakamura)

*RubyKaigi Takeout 2021*

2021-09-11



RubyKaigi Takeout 2021



## Do Pure Ruby Dreams Encrypted Binary Protocol?

unasuke (Yusuke Nakamura)

*RubyKaigi 2022*

2022-09-09

# Implement QUIC from scratch

---

Have you ever created a Rails application?



# Implement QUIC from scratch

---

Have you ever implemented the QUIC protocol?





# Implement QUIC from scratch

---

It's too difficult!

- When create a Rails application
  - learn from "Rails Guides" and "Rails Tutorial"
- When create a QUIC implementation...?
  - RFCs are not a "implementation guide"

## Try to implement QUIC once

---

- To learn how to implement QUIC
  - Implement it once (how?)
    - Porting existing implementation!

## Topics

---

1. Porting from Python to Ruby
2. Rubyish QUIC implementation
3. Future of my implementation

# Porting from Python to Ruby

---

## To Ruby From Python

Python is another very nice general purpose programming language. Going from Python to Ruby, you'll find that there's a little bit more syntax to learn than with Python.

### Similarities

As with Python, in Ruby,...

- There's an interactive prompt (called `irb`).
- You can read docs on the command line (with the `ri` command instead of `pydoc`).
- There are no special line terminators (except the usual newline).
- String literals can span multiple lines like Python's triple-quoted strings

<https://www.ruby-lang.org/en/documentation/ruby-from-other-languages/to-ruby-from-python/>

## Porting from Python to Ruby

---

```
ary = [1, 2, 3]

print(f"length of the array is {len(ary)}")

for i in ary:
    if i % 2 == 0:
        print(i * 2)
    else:
        print(i)
```

Can you port this Python code to Ruby?

# Porting from Python to Ruby - Code amount

> tokei

Language	Files	Lines	Code	Comments	Blanks
Autoconf	1	4	4	0	0
C	2	1025	751	128	146
CSS	1	10	9	0	1
HTML	2	63	63	0	0
JSON	1	3	3	0	0
Makefile	1	20	10	6	4
Python	55	22019	18483	1230	3205
ReStructuredText	8	463	300	0	163
SVG	1	72	72	0	0
Plain Text	2	5	0	5	0
TOML	1	2	2	0	0
Total	75	24585	19697	1369	3519

>

## Porting from Python to Ruby - How I ported it

---

- Keep the same structure as Python
  - Carelessly changed may cause getting stuck
- Avoid porting asynchronous processing parts
  - Difference of functionality
  - My lack of knowledge

# Porting from Python to Ruby - Built-in types, bytes

---

```
# aioquic src/aioquic/_buffer.pyi
class Buffer:
    def __init__(self, capacity: Optional[int] = 0, data: Optional[bytes] = None): ...
    @property
    def capacity(self) -> int: ...
    @property
    def data(self) -> bytes: ...
    def data_slice(self, start: int, end: int) -> bytes: ...
    # ...
```

[https://github.com/aioquic/aioquic/blob/main/src/aioquic/\\_buffer.pyi](https://github.com/aioquic/aioquic/blob/main/src/aioquic/_buffer.pyi)



## Porting from Python to Ruby - Built-in types, bytes

---

- bytes in Python
  - immutable (bytearray is not)
- String in Ruby
  - mutable
  - has Encoding (not only ASCII)

## Porting from Python to Ruby - Built-in types, bytes

---

Python returns 3

```
# Python  
len(b"\xe3\x81\x82") # => 3
```

Ruby returns 1 (return 3 if use String#bytesize)

```
# Ruby  
"\xe3\x81\x82".length # => 1 ("あ")  
"\xe3\x81\x82".bytesize # => 3
```

## Porting from Python to Ruby - Built-in types, enum

---

```
from enum import IntEnum
import pprint

class Color(IntEnum):
    RED = 1
    BLUE = 2
    GREEN = 3

color = Color.RED
print(color) # => 1
```

Python

## Porting from Python to Ruby - Built-in types, enum

---

```
class Color
  RED = 1
  BLUE = 2
  GREEN = 3
end

color = Color::RED
puts color # => 1
```

Ruby

## Porting from Python to Ruby - Built-in types, enum

---

```
from aioquic import tls
import pprint

ctx = tls.Context(is_client=True)
print(ctx._signature_algorithms[2]) # => 1025

pprint.pp(ctx._signature_algorithms[2])
# => <SignatureAlgorithm.RSA_PKCS1_SHA256: 1025>
```

"Oh! 1025 is the SignatureAlgorithm.RSA\_PKCS1\_SHA256 in TLS 1.3!"

## Porting from Python to Ruby - Built-in types, tuple

---

```
# example of initialize QUIC connection
import time
from aioquic.quic.connection import QuicConnection
from aioquic.quic.configuration import QuicConfiguration

conf = QuicConfiguration(is_client=True)
conn = QuicConnection(configuration=conf)
# ("1.2.3.4", 1234) is a tuple of IP address and port number
conn.connect(("1.2.3.4", 1234), now=time.time())
# ...snip...
```

Python

# Porting from Python to Ruby - Built-in types, tuple

---

```
# example of initialize QUIC connection (in Ruby)
require 'raioquic'
conf = Raioquic::Quic::QuicConfiguration.new(is_client: true)
conn =
  Raioquic::Quic::Connection::QuicConnection.new(configuration: conf)

# ["1.2.3.4", 1234] is an array of IP address and port number
conn.connect(addr: ["1.2.3.4", 1234], now: Time.now.to_f)
# ...snip...
```

Ruby

# Porting from Python to Ruby - Code style

---

```
# aioquic src/aioquic/quic/recovery.py (edited)
def on_ack_received(self, space, ack_rangeset, ack_delay, now) -> None:
    """
    Update metrics as the result of an ACK being received.
    """
    is_ack_eliciting = False
    largest_acked = ack_rangeset.bounds().stop - 1
    largest_newly_acked = None
    largest_sent_time = None

    if largest_acked > space.largest_acked_packet:
        space.largest_acked_packet = largest_acked

    for packet_number in sorted(space.sent_packets.keys()):
        if packet_number > largest_acked:
            break
        if packet_number in ack_rangeset:
            # ...snip...
            # trigger callbacks
            for handler, args in packet.delivery_handlers:
                handler(QuicDeliveryState.ACKED, *args) # <== HERE!!!
```

Python



# Porting from Python to Ruby - Code style

```
# raioquic lib/raioquic/quic/recovery.rb (edited)
def on_ack_received(space:, ack_rangeset:, ack_delay:, now:)
  # ..snip..
  # trigger callbacks
  packet.delivery_handlers&.each do |handler|
    # TODO: hmm...
    delivery = Quic::PacketBuilder::QuicDeliveryState::ACKED
    case handler[0]&.name
    when :on_data_delivery
      handler[0].call(delivery: delivery, start: handler[1][0], stop: handler[1][1])
    when :on_ack_delivery
      handler[0].call(delivery: delivery, space: handler[1][0], highest_acked: handler[1][1])
    when :on_new_connection_id_delivery
      handler[0].call(delivery: delivery, connection_id: handler[1][0])
    when :on_handshake_done_delivery, :on_reset_delivery, :on_stop_sending_delivery
      handler[0].call(delivery: delivery)
    when :on_ping_delivery
      handler[0].call(delivery: delivery, uids: handler[1][0])
    when :on_connection_limit_delivery
      handler[0].call(delivery: delivery, limit: handler[1][0])
    when :on_retire_connection_id_delivery
      handler[0].call(delivery: delivery, sequence_number: handler[1][0])
    else
      raise NotImplementedError, handler[0]
    end
  end
end
```

Ruby

# Porting from Python to Ruby - Code style

---

```
# aioquic src/aioquic/quic/connection.py (edited)
def _write_ping_frame(self, builder, uids = [], comment=""):
    builder.start_frame(
        QuicFrameType.PING,
        capacity=PING_FRAME_CAPACITY,
        handler=self._on_ping_delivery, # <=== callback function
        handler_args=(tuple(uids),), # <===== callback function args(tuple)
    )
    self._logger.debug(
        "Sending PING%s in packet %d",
        " (%s)" % comment if comment else "",
        builder.packet_number,
    )
```

callback register in Python

## Porting from Python to Ruby - Library API

---

Not only language themselves but also API differences.

Using OpenSSL functionality from...

- Python (aioquic)

- pyca/cryptography

- <https://github.com/pyca/cryptography>

- Ruby

- openssl gem

- <https://github.com/ruby/openssl>

(to implement TLS 1.3)

## Porting from Python to Ruby - Library API

---

1. looked for API calls to pyca/cryptography's
2. find which C API of OpenSSL it corresponds to
3. find how it is wrapped in the openssl gem
4. port it to a Ruby API call in the openssl gem

# Porting from Python to Ruby - Library API example

---

```
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives.serialization import Encoding, PublicFormat

peer1_ec_private_key = ec.generate_private_key(ec.SECP256R1)
peer1_ec_public_key = peer1_ec_private_key.public_key()
encoded_peer1_pubkey = peer1_ec_public_key.public_bytes(Encoding.X962, PublicFormat.UncompressedPoint)

peer2_ec_private_key = ec.generate_private_key(ec.SECP256R1)
peer2_ec_public_key = peer2_ec_private_key.public_key()
encoded_peer2_pubkey = peer2_ec_public_key.public_bytes(Encoding.X962, PublicFormat.UncompressedPoint)

decoded_peer2_public_key = ec.EllipticCurvePublicKey.from_encoded_point(ec.SECP256R1(), encoded_peer2_pubkey)
peer1_shared_key = \
    peer1_ec_private_key.exchange(ec.ECDH(), decoded_peer2_public_key)

decoded_peer1_public_key = ec.EllipticCurvePublicKey.from_encoded_point(ec.SECP256R1(), encoded_peer1_pubkey)
peer2_shared_key = \
    peer2_ec_private_key.exchange(ec.ECDH(), decoded_peer1_public_key)

print(peer1_shared_key)
print(peer2_shared_key)
print(peer1_shared_key == peer2_shared_key) # => True
```

# Porting from Python to Ruby - Library API example

---

```
require 'openssl'

peer1_ec = OpenSSL::PKey::EC.generate("prime256v1")
peer1_ec_public_key = peer1_ec.public_key
encoded_peer1_public_key = peer1_ec_public_key.to_octet_string(:uncompressed)

peer2_ec = OpenSSL::PKey::EC.generate("prime256v1")
peer2_ec_public_key = peer2_ec.public_key
encoded_peer2_public_key = peer2_ec_public_key.to_octet_string(:uncompressed)

group = OpenSSL::PKey::EC::Group.new("prime256v1")
decoded_peer2_public_key = OpenSSL::PKey::EC::Point.new(group, encoded_peer2_public_key)
peer1_shared_key = peer1_ec.dh_compute_key(decoded_peer2_public_key)

decoded_peer1_public_key = OpenSSL::PKey::EC::Point.new(group, encoded_peer1_public_key)
peer2_shared_key = peer2_ec.dh_compute_key(decoded_peer1_public_key)

pp peer1_shared_key
pp peer2_shared_key
pp peer1_shared_key == peer2_shared_key # => true
```

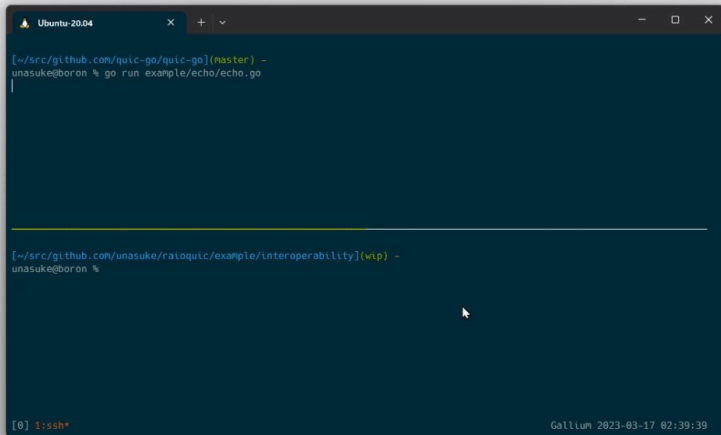
# Porting from Python to Ruby - Result

> tokei .

Language	Files	Lines	Code	Comments	Blanks
BASH	1	8	4	2	2
Go	1	61	46	4	11
Python	4	151	115	15	21
Rakefile	1	16	10	1	5
Ruby	42	14624	11463	1175	1986
Markdown	6	149	0	88	61
- Ruby	1	1	1	0	0
- Shell	1	3	3	0	0
(Total)		153	4	88	61
Total	55	15009	11638	1285	2086

> █

# Porting from Python to Ruby - Demo



A terminal window titled "Ubuntu-20.04" with standard window controls. The terminal shows two separate command-line sessions. The first session is in a Go directory, where the command `go run example/echo/echo.go` is entered. The second session is in a Ruby directory, where the command `ruby example/interoperability.rb` is entered. A horizontal yellow line separates the two sessions. The bottom status bar shows "[0] 1:ssh\*" on the left and "Gallium 2023-03-17 02:39:39" on the right.

```
Ubuntu-20.04 X + v - □ X

[~/src/github.com/quic-go/quic-go](master) -
unasuke@boron % go run example/echo/echo.go
|

[~/src/github.com/unasuke/raioquic/example/interoperability](wip) -
unasuke@boron %

[0] 1:ssh* Gallium 2023-03-17 02:39:39
```



## Porting from Python to Ruby - Insights

---

1. QUIC IS VERY DIFFICULT
2. TLS 1.3 IS ALSO VERY DIFFICULT
3. "writing once" empowers me

# Porting from Python to Ruby - "Raioquic"

---

unasuke/**raioquic**



1

Contributor



0

Issues



13

Stars



0

Forks



<https://github.com/unasuke/raioquic>

# Rubyish QUIC implementation

---

```
require "rалоquic"
require "socket"

HOST = ["0.0.0.0", 4433]

conf = Raloquic::Quic::QuicConfiguration.new(is_client: true)
conf.load_verify_locations(cafile: "localhost-unasuke-dev.crt")
client = Raloquic::Quic::Connection::QuicConnection.new(configuration:
  conf)
client.connect(addr: HOST, now: Time.now.to_f)
s = UDPSocket.new
s.connect("0.0.0.0", 4433)

# handshake
3.times do
  client.datagrams_to_send(now: Time.now.to_f).each do |data, addr|
    s.send(data, 0)
  end
  data, addr = s.recvfrom(65536)
  client.receive_datagram(data: data, addr: HOST, now: Time.now.to_f)
  while ev = client.next_event
    pp ev
  end
end

stream_id = client.get_next_available_stream_id

2.times do
  client.send_stream_data(stream_id: stream_id, data: "hello")
  client.datagrams_to_send(now: Time.now.to_f).each do |data, addr|
    s.send(data, 0)
  end
  data, addr = s.recvfrom(65536)
  client.receive_datagram(data: data, addr: HOST, now: Time.now.to_f)
  while ev = client.next_event
    pp ev
  end
end
client.close
```

## Future of my implementation - How to make it Rubyish?

---

To make implementation Rubyish...

1. Use suitable features to Ruby (internal)

- tuple → class or dedicated struct or data
- bytes → `IO::Buffer`, not `String`

2. Adapt to existing API styles (public API)

## Future of my implementation - TLS in Ruby

---

```
require 'net/http'

url = URI.parse('https://www.example.com/index.html')
req = Net::HTTP.new(url.host, url.port)
req.use_ssl = true
res = req.get(url.path)
puts res.body
```

High level API (use Net::HTTP)

# Future of my implementation - TLS in Ruby

---

```
# https://docs.ruby-lang.org/ja/latest/class/OpenSSL=3a=3aSSL=3a=3aSSLSocket.html
require 'socket'
require 'openssl'

soc = TCPSocket.new('www.example.com', 443)
ssl = OpenSSL::SSL::SSLSocket.new(soc)
ssl.connect
ssl.post_connection_check('www.example.com')
raise "verification error" if ssl.verify_result != OpenSSL::X509::V_OK

ssl.write('hoge')
print ssl.peer_cert.to_text

ssl.close
soc.close
```

Low level API (use OpenSSL::SSL::SSLSocket)

## Future of my implementation - Faraday style

---

```
# Faraday like client API  
res = QuicClient.get("https://quic.nginx.org/")
```

Very high level API

## Future of my implementation - Net::HTTP style

---

```
# API similar to Net::HTTP
require 'net/http/quic'

url = URI.parse('https://cloudflare-quic.com/')
req = Net::HTTP::Quic.new(url.host, url.port)
req.use_ssl = true
res = req.get(url.path)
puts res.body
```

Net::HTTP style API



## Future of my implementation - Vaporware

---

- The APIs like I've been talking about, doesn't exist yet
  - next year...?
    - um...

## **Future of my implementation - Raise your hand time!**

---

Are you running some Rails apps on production env?



## Future of my implementation - Raise your hand time!

---

Are you running some Rails apps on production **without** reverse proxies or load balancers?



## Future of my implementation - Speed!

---

- Most Rails app is behind of web server (load balancer)
  - TLS termination is high cost
  - (asset delivery)

# Future of my implementation - QUIC diagram (again!)

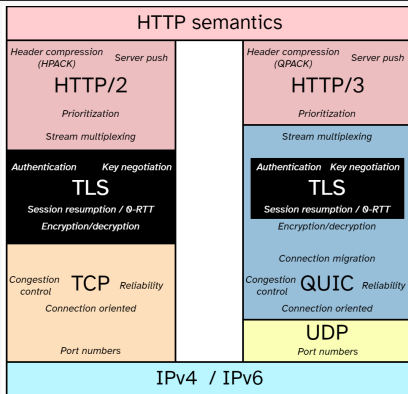



image from <https://github.com/rmarx/h3-protocol-stack>





## Future of my implementation - Faster language impls

---

- C or Rust or...
  - [socketry/protocol-quic](#) gem wraps ngtcp2
    - created by Samuel-san (ioquatix)  
"Unleashing the Power of Asynchronous HTTP with Ruby" in Day 3
  - [ngtcp2/ngtcp2](#) written by C

# Future of my implementation - ngtcp2 used by curl

 master [curl / docs / HTTP3.md](#) ↑ Top

Preview Code Blame 354 lines (246 loc) · 12.2 KB Raw    

## ngtcp2 version

Building curl with ngtcp2 involves 3 components: `ngtcp2` itself, `nghttp3` and a QUIC supporting TLS library. The supported TLS libraries are covered below.

For now, `ngtcp2` and `nghttp3` are still *experimental* which means their evolution bring breaking changes. Therefore, the proper version of both libraries need to be used when building curl. These are

- `ngtcp2` : v0.13.1
- `nghttp3` : v0.10.0

<https://github.com/curl/curl/blob/master/docs/HTTP3.md>

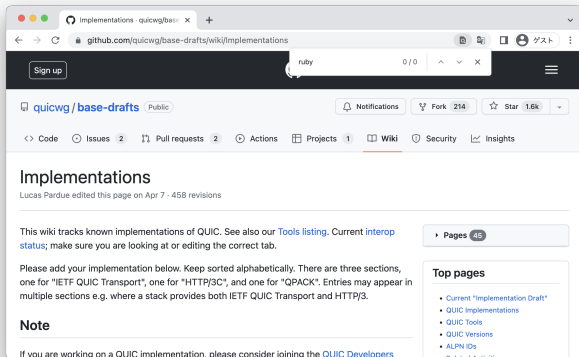
## Future of my implementation - Worth of Pure Ruby

---

- Research
  - Implementations that make it easy to change internal behavior are useful
- QUIC implementation itself
  - Helping QUIC implementation itself
    - e.g. Build data for test its behavior



# Future of my implementation - Motivation



<https://github.com/quicwg/base-drafts/wiki/Implementations>

# Summary

---

- Ported aioquic (Python) to Ruby
  - Differs from language features, library API makes porting hard
  - Ported impl could communicate other impls
- Creating Rubyish implementation
  - Uses Ruby's built-in features
  - Make it can apply existing idioms
- This may be where this implementation would be useful
  - Research
  - QUIC implementation itself

# Acknowledgments

---

- Ruby Association
  - For adopting my project
- Koichi Sasada-san
  - a mentor of the porting project
- Kuwayama-san
  - Author of `ttls1.3` gem
- Daisuke Aritomo a.k.a. osyoyu
  - Adviser of this talk