RubyによるQUICプロトコルの他言語からの移植ならびに独自実装の作成

unasuke (Yusuke Nakamura)

Ruby Association Activity Report

2023-08-28

自己紹介

- Name: unasuke（Yusuke Nakamura）
- Work: フリーランス
- Kaigi on Rails オーガナイザー（10/27-28 開催）
- GitHub https://github.com/unasuke
- ActivityPub https://mstdn.unasuke.com/@unasuke
- X（Twitter）https://twitter.com/yu_suke1994
- https://unasuke.com

RubyによるQUICプロトコルの他言語からの移植ならびに独自実装の作成

- 目的1: aioquicをRubyに移植する
- 目的2: 移植の知見からRubyishなQUIC実装を作成する

QUICとは何か

- RFC 9000を含む複数のRFCによって標準化された通信プロトコル
- UDPの上にTCPのような信頼性のある通信を実現する
- TLSが組み込まれており、セキュアな通信がデフォルトとなっている

QUICとは何か - Protocol Stack

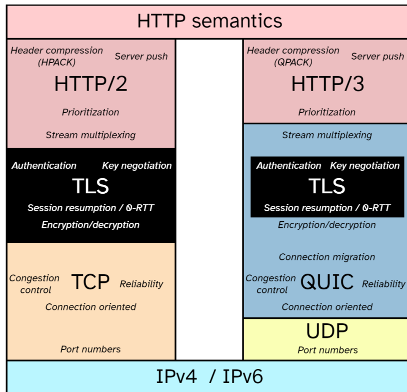

image from https://github.com/rmarx/h3-protocol-stack

aioquicとは何か

- https://github.com/aiortc/aioquic
- PythonによるQUIC実装
- async/await構文を使用している

既存実装

- https://github.com/quicwg/base-drafts/wiki/Implementations
  - IETF QUIC WG wikiにおけるQUICの実装一覧
- C, C++, Go, Rust, Haskell, Python, Java等
  - no Ruby

移植の動機

- QUICのRuby実装が存在しない
- Ractorの大規模採用例があるとよいのではないか

そもそも移植は可能なのか

```
> 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        22918        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
===============================================================================
>
```

aioquicの実装言語比率

aioquicの依存関係

- https://pypi.org/project/certifi/ 証明書を提供する
- https://pypi.org/project/pyOpenSSL/ OpenSSL
  - https://pypi.org/project/cryptography/
- https://pypi.org/project/pylsqpack/ QPACK
  - https://github.com/litespeedtech/ls-qpack/

aioquicの構造

```
$ tree src
src
├── aioquic
│   ├── about.py
│   ├── asyncio
│   │   ├── client.py
│   │   ├── __init__.py
│   │   ├── protocol.py
│   │   └── server.py
│   ├── _buffer.c
│   ├── buffer.py
│   ├── _crypto.c
│   ├── _crypto.pyi
│   ├── quic
│   │   ├── configuration.py
│   │   ├── connection.py
│   │   ├── crypto.py
│   │   ├── events.py
│   │   ├── __init__.py
│   │   ├── logger.py
│   │   ├── packet_builder.py
│   │   ├── packet.py
│   │   ├── rangeset.py
│   │   ├── recovery.py
│   │   ├── retry.py
│   │   └── stream.py
│   └── tls.py
...
```

buffer.c, buffer.pyi

- C言語によるBuffer領域の実装をPython Objectにしたもの
- mallocした領域に対する操作をPythonから行える
- → StringIOをwrapするClassを作成して移植

packet.py, packet_builder.py

- Packetそのものを表現したり構築するclass群
- → ほぼそのままClass及びStructに移植

crypto.py, _crypto.c, _crypto.pyi

- QUICのPacketそのものを暗号化するclass群
- OpenSSLのAPIを利用するC実装も含む
- openssl gem側でのAPIを調査してPure Rubyに移植

tls.py

- TLS 1.3の(ほぼ)Pure Python実装
  - 鬼門(1800行)
- →ほぼそのまま移植した

OpenSSL APIの差異

- PythonとRubyでOpenSSL APIをどのように抽象化するかが異なる
- Python側はほぼCと1対1
- Ruby側は扱いやすいように抽象化されている

OpenSSL APIの差異 Python側

```python
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
```

OpenSSL APIの差異 Ruby側

```ruby
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
```

connection.py

- QUICのどのPacketを実際に取り扱う通信部分
  - 鬼門(3200行)
- →ほぼそのまま移植した(テストケース未完走)

最終報告時点の成果

```
> 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
===============================================================================
>
```

最終報告時点の成果

- https://github.com/unasuke/raioquic
  - 11000行
- aioquic, quic-goとの簡単な通信ができることを確認

やらなかったこと

- connection.rbのテストケース完走
- 実際にリクエストを受け付ける部分の移植
  - asyncioを使用している部分
- Rubyishな実装の作成

得られた知見

- PythonとRubyの言語の差
- QUIC及びTLS 1.3に対する理解

その後の活動

- RubyKaigi 2023登壇
  - https://rubykaigi.org/2023/presentations/yu_suke1994.html
- IETF Meetingに参加
  - 116 Yokohama
  - 117 San Francisco (remote)
- Rubyishな実装の作成
  - RubyKaigi 2023 follow up

Rubyishな実装の作成

- APIをRubyishにする
- Documentationをしっかりやる
- `IO::Buffer` を使用してみる

Documentationをしっかりやる（HKDF）

```
.hkdf_expand_label(secret, label, context, length) ⇒ String
```

permalink

Provide HKDF-Expand-Label function

```
HKDF-Expand-Label(Secret, Label, Context, Length) =
 HKDF-Expand(Secret, HkdfLabel, Length)

 Where HkdfLabel is specified as:

 struct {
    uint16 length = Length;
    opaque label<7..255> = "tls13 " + Label;
    opaque context<0..255> = Context;
 } HkdfLabel;
```

from RFC 8446

Parameters:

- secret (String) — (In RFC5869, this is *PRK*) a pseudorandom key of at least HashLen octets (usually, the output from the extract step)
- label (String) — An element of *Hkdflabel* (part of *info* in RFC 5869)
- context (String) — An element of *HkdfLabel* (part of *info* in RFC 5869)
- length (Integer) — length of output keying material in octets (<= 255*HashLen)

Returns:

- (String) — output keying material with label

See Also:

- https://www.rfc-editor.org/rfc/rfc8446.html#section-7.1
- https://www.rfc-editor.org/rfc/rfc5869.html#section-2.3
- OpenSSL::KDF.hkdf

IO::Buffer を使用してみる

```ruby
# Apply packet protection
# @see https://www.rfc-editor.org/rfc/rfc9001.html#section-5.4
def apply(plain_header, protected_payload)
  packet_number_length = (plain_header.get_value(:U8, 0) & 0x03) + 1
  packet_number_offset = plain_header.size - packet_number_length
  @mask = mask(
    protected_payload.slice(PACKET_NUMBER_LENGTH_MAX - packet_number_length).slice(0, SAMPLE_LENGTH)
  )

  buffer = IO::Buffer.for(plain_header.get_string + protected_payload.get_string).dup # make mutable
  if buffer.get_value(:U8, 0) & 0x80 != 0
    buffer.copy(buffer.slice(0, 1).xor!(IO::Buffer.for((@mask[0].unpack1("C*") & 0x0f).chr)))
  else
    buffer.set_value(:U8, 0, buffer.slice(0, 1).xor!(IO::Buffer.for((@mask[0].unpack1("C*") & 0x1f)).chr))
  end
  buffer.copy(
    buffer.slice(
      packet_number_offset, packet_number_length).xor!(IO::Buffer.for(@mask[1..packet_number_length])
    ),
    packet_number_offset
  )
  buffer
end
```

まとめ

- 引き続きRubyishなQUIC実装の開発を進めていく
- IETF Meetingに参加して最新動向を追っていく
- 謝辞
  - Ruby Association様
  - 笹田耕一様(メンター)