Do Pure Ruby Dreams Encrypted Binary Protocol?

unasuke (Yusuke Nakamura)

RubyKaigi 2022
2022-09-09

Self introduction

- Name: unasuke (Yusuke Nakamura)
- Work: freelance Web app developer @ Tokyo
  - Rails app developer (mainly)
- Itamae gem maintainer, Kaigi on Rails Organizer
- GitHub https://github.com/unasuke
- Mastodon https://mstdn.unasuke.com/@unasuke
- Twitter https://twitter.com/yu_suke1994

At first, do PureRuby Dream of Encrypted Binary Protocol?

A: There is a harsh reality. 😭

A brief explanation of what QUIC is

- QUIC: standarized at 2021 by RFC 9000 etc
  - Using UDP for communication

TCP

low efficiency, high reliability

UDP

high efficiency, low reliability → faster than TCP

Did you remember my last year talk about QUIC?

https://slide.rabbit-shocker.org/authors/unasuke/rubykaigi-takeout-2021/

Ractor, I dropped out

It's too difficult.Because...

- Implementating communication protorol is very hard work
- debugging code using Ractor is very hard in now
  - debug.gem, I hope it will be a savior in the future...

Try to solve two problems in one time is difficult.

→ I gave up to use Ractor in first implementation 😇

It's a binary protocol

message like that

```
{ "message": "Hello!", "kind": "greet" }
```

parse it by ruby

```ruby
require "json"
request = JSON.parse('{ "message": "Hello!", "kind": "greet" }')
pp request["message"] # => "Hello!"
```

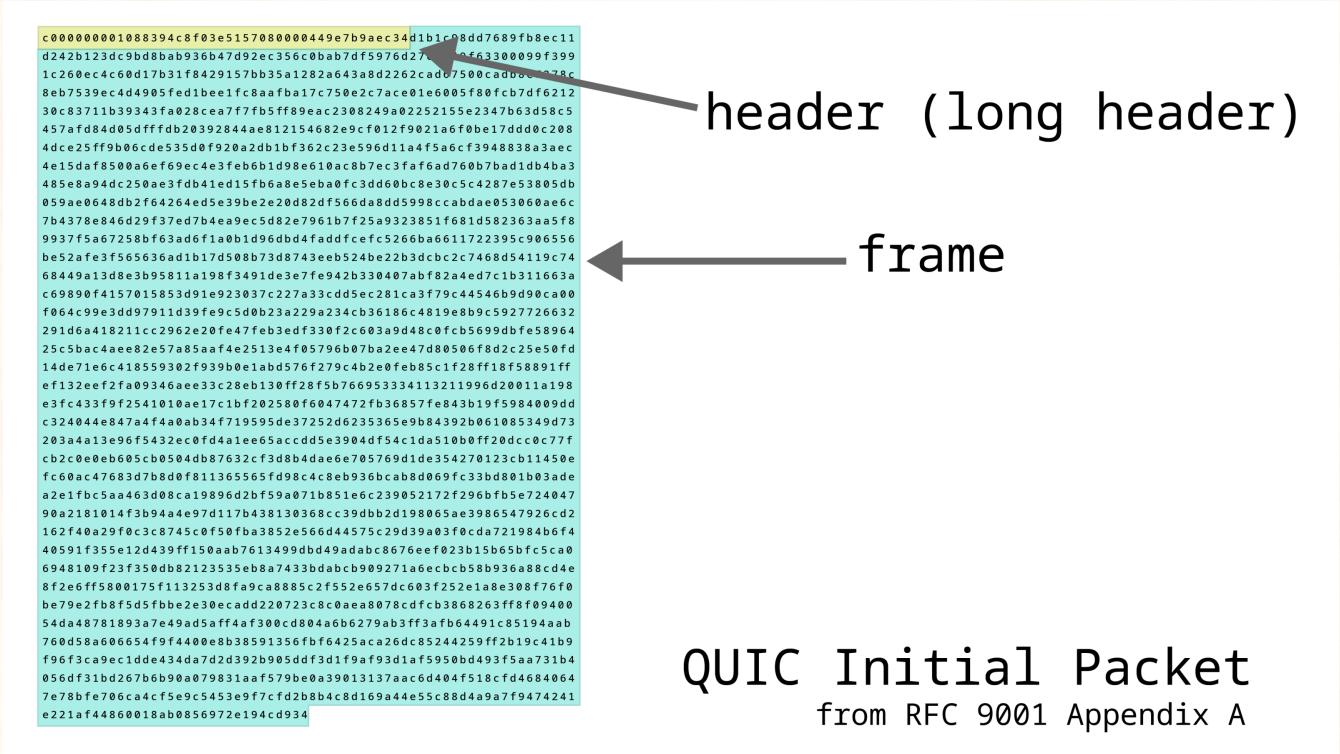
It's a binary protocol

QUIC message

```
c000000001088394c8f03e5157080000449e7b9aec34d1b1c98dd7689fb8ec11
d242b123dc9bd8bab936b47d92ec356c0bab7df5976d27cd449f63300099f399
1c260ec4c60d17b31f8429157bb35a1282a643a8d2262cad67500cadb8e7378c
8eb7539ec4d4905fed1bee1fc8aafba17c750e2c7ace01e6005f80fcb7df6212
30c83711b39343fa028cea7f7fb5ff89eac2308249a02252155e2347b63d58c5
457afd84d05dfffdb20392844ae812154682e9cf012f9021a6f0be17ddd0c208
4dce25ff9b06cde535d0f920a2db1bf362c23e596d11a4f5a6cf3948838a3aec
4e15daf8500a6ef69ec4e3feb6b1d98e610ac8b7ec3faf6ad760b7bad1db4ba3
...
```

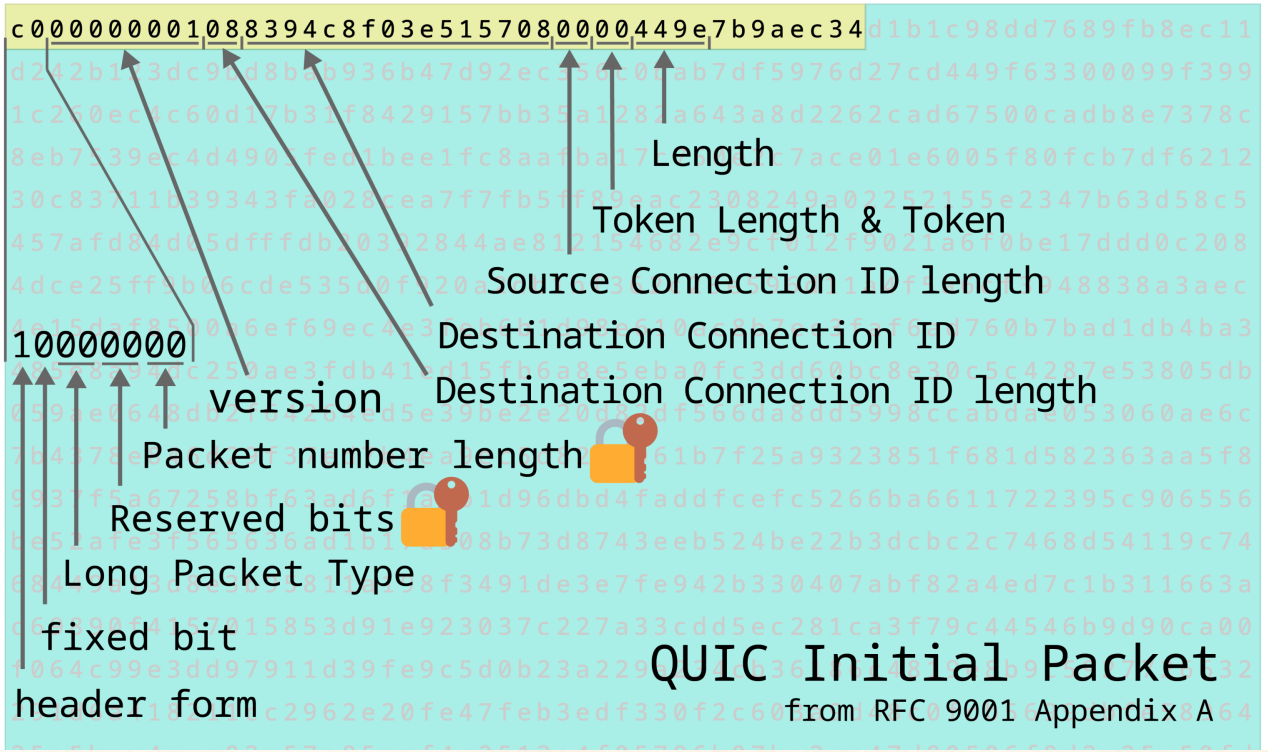
It's a binary protocol

Reading QUIC message (same as last year's slide)



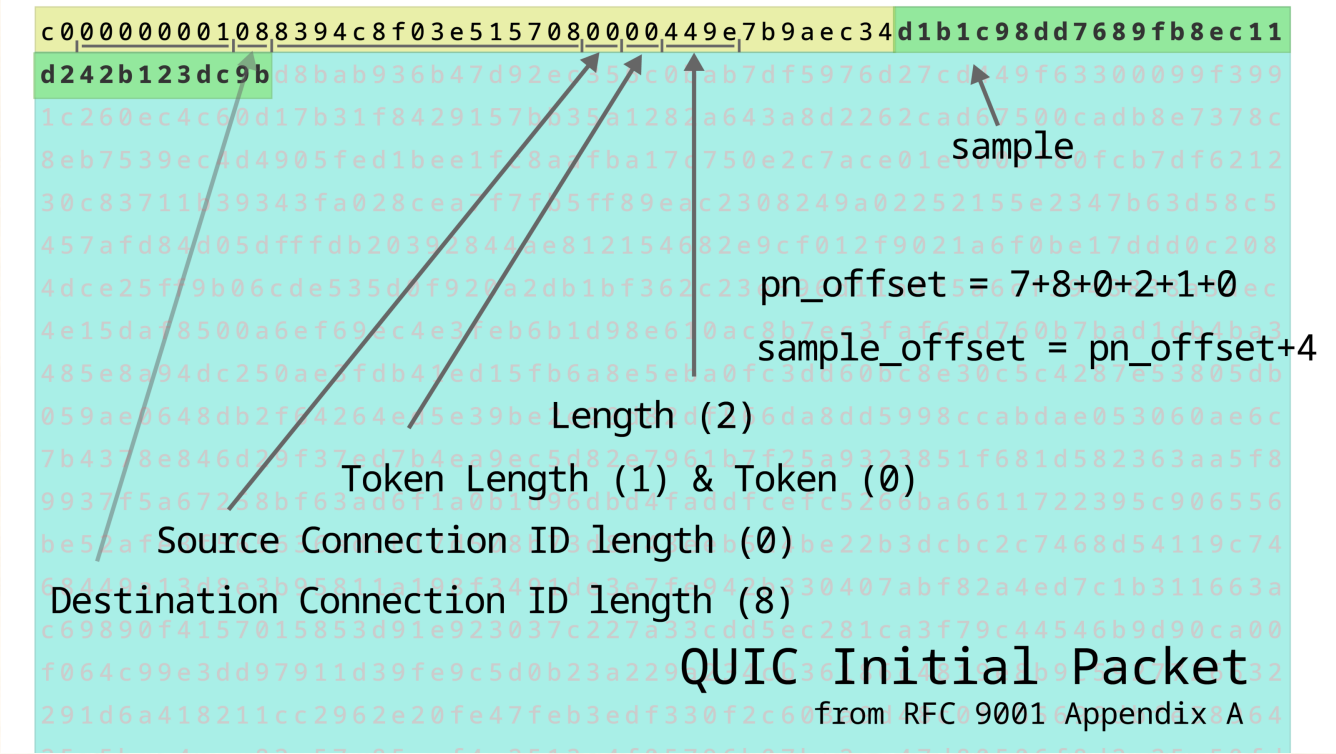
It's a binary protocol

Reading QUIC message (same as last year's slide)



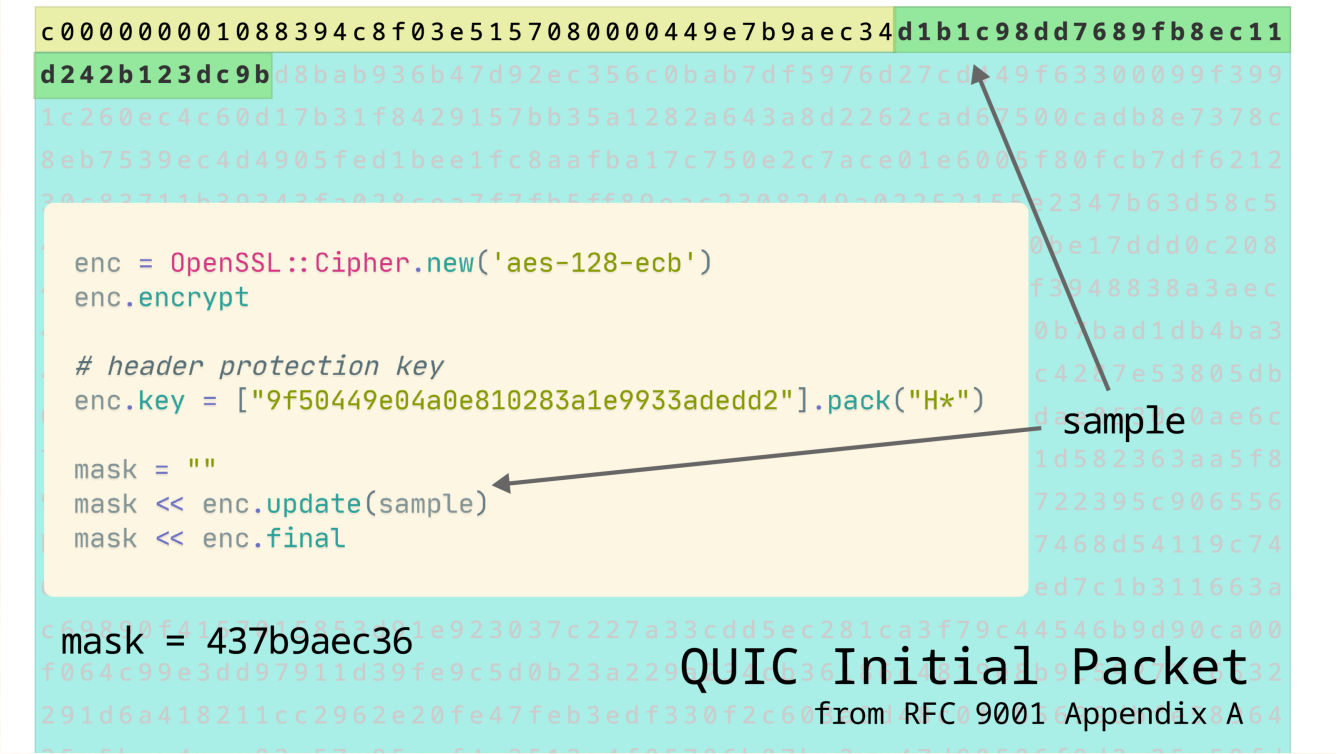
It's a binary protocol

Reading QUIC message (same as last year's slide)



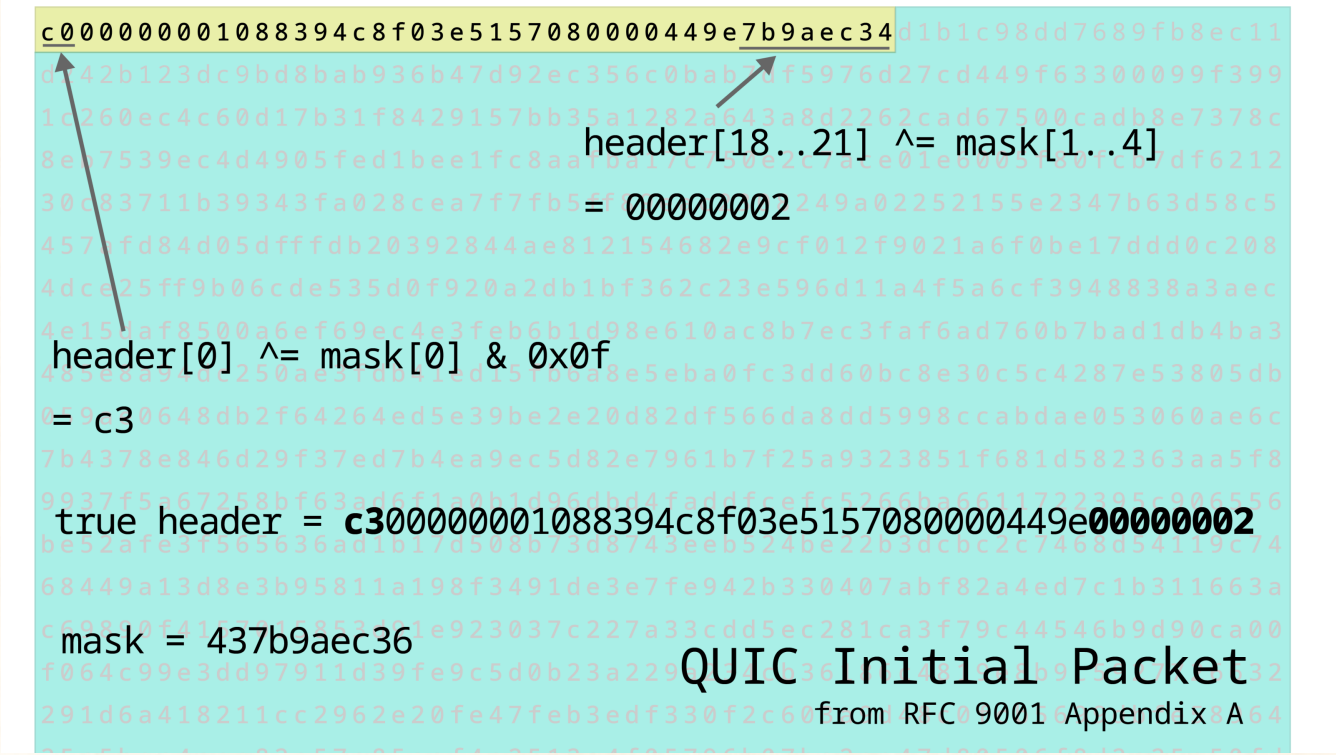
It's a binary protocol

Reading QUIC message (same as last year's slide)



It's a binary protocol

Reading QUIC message (same as last year's slide)



It's a binary protocol

Reading QUIC message (same as last year's slide)

QUIC Initial Packet

from RFC 9001 Appendix A

It's a binary protocol

Convert to bit-by-bit representation

```
"Hello".unpack1("B*")
# => "01001000011001010110110001101100011011111"
```

It's a binary protocol

Oops!

```ruby
data = "Hello".unpack1("B*")
# ...snip...
data.unpack1("B*") # unpack twice!
# => "00110000001100010011000000110000001100010011 0...."
```

I wasted a lot of time because of this mistake. 

How are the other language implementations

Look around some QUIC impletemtations

- kwik (Java)
- quic-go (Go)
- cloudflare/quiche (Rust)
- aioquic (Python)

kwik (Java) : QUIC impletemtations

```
void parsePacketNumberAndPayload(ByteBuffer buffer, byte flags, int
remainingLength, Keys serverSecrets, long largestPacketNumber, Logger
log) throws DecryptionException, InvalidPacketException {
```

https://github.com/ptrd/kwik/blob/d1c52e6ac3/src/main/java/net/luminis/quic/packet/QuicPacket.java#L89

quic-go (Go) : QUIC impletemtations

```
type unpackedPacket struct {
    packetNumber    protocol.PacketNumber // the decoded packet number
    hdr             *wire.ExtendedHeader
    encryptionLevel protocol.EncryptionLevel
    data            []byte
}
```

https://github.com/lucas-clemente/quic-go/blob/66f6fe0b711bc/packet_unpacker.go#L29-L34

cloudflare/quiche (Rust) : QUIC impletemtations

```rust
impl<'a> OctetsMut<'a> {
    /// Creates an `OctetsMut` from the given slice, without copying.
    ///
    /// Since there's no copy, the input slice needs to be mutable to allow
    /// modifications.
    pub fn with_slice(buf: &'a mut [u8]) -> Self {
        OctetsMut { buf, off: 0 }
    }
```

https://github.com/cloudflare/quiche/blob/3131c0d37/octets/src/lib.rs#L329-L336

aioquic (Python) : QUIC impletemtations

```python
@dataclass
class QuicStreamFrame:
    data: bytes = b""
    fin: bool = False
    offset: int = 0
```

https://github.com/aiortc/aioquic/blob/c758b4d936/src/aioquic/quic/packet.py#L477-L481

back to the Ruby

- Those languages have Byte specific class or types
  - but Ruby is not
    - but we can use String or array of Integer

I would like to see a binary protocol implementation by Ruby that already exists.

→ MessagePack!

msgpack/msgpack-ruby

```
MessagePack is an efficient binary serialization format.
```

https://msgpack.org

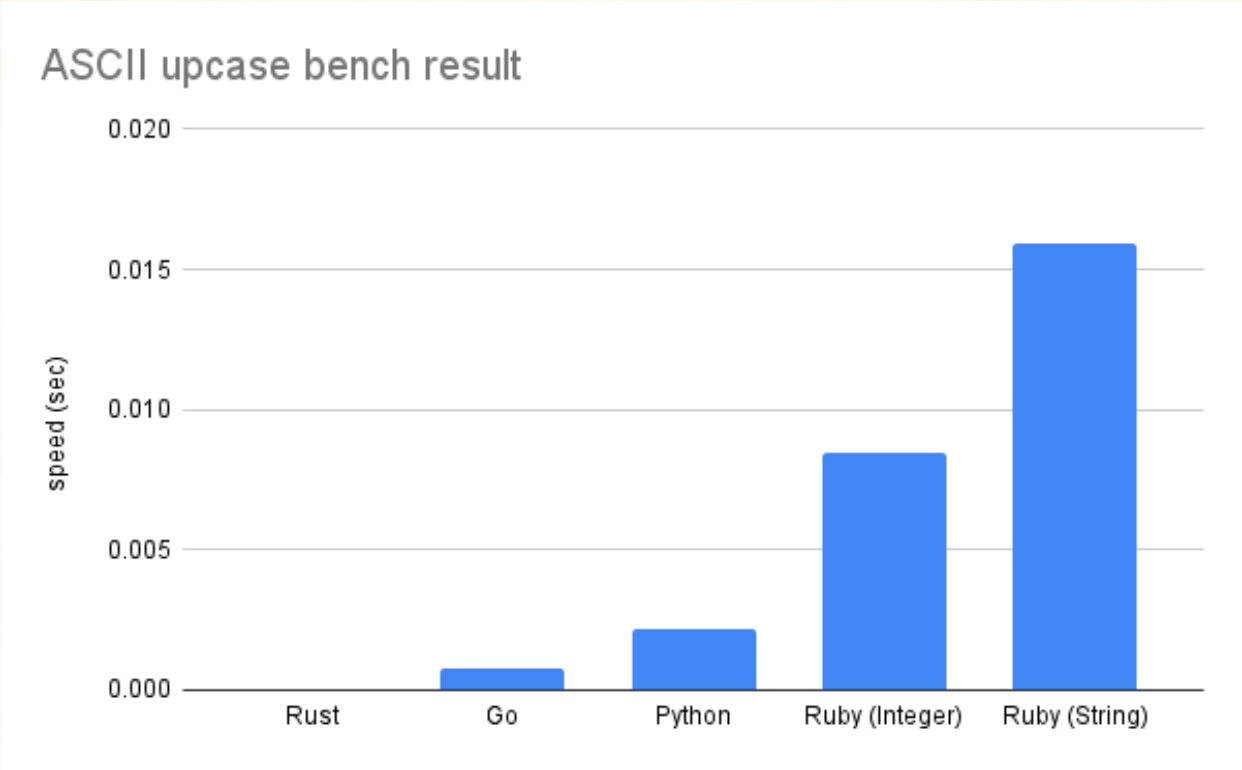
msgpack/msgpack-ruby

```
void _msgpack_unpacker_init(msgpack_unpacker_t* uk)
{
    msgpack_buffer_init(UNPACKER_BUFFER_(uk));


    uk->head_byte = HEAD_BYTE_REQUIRED;


    uk->last_object = Qnil;
    uk->reading_raw = Qnil;


    uk->stack = _msgpack_unpacker_new_stack();
}
```

https://github.com/msgpack/msgpack-ruby/blob/0775a9a6a5/ext/msgpack/unpacker.c#L75-L85

compare languages

benchmaek for bytes manipulation (AWS c5.large Ubuntu 22.04)



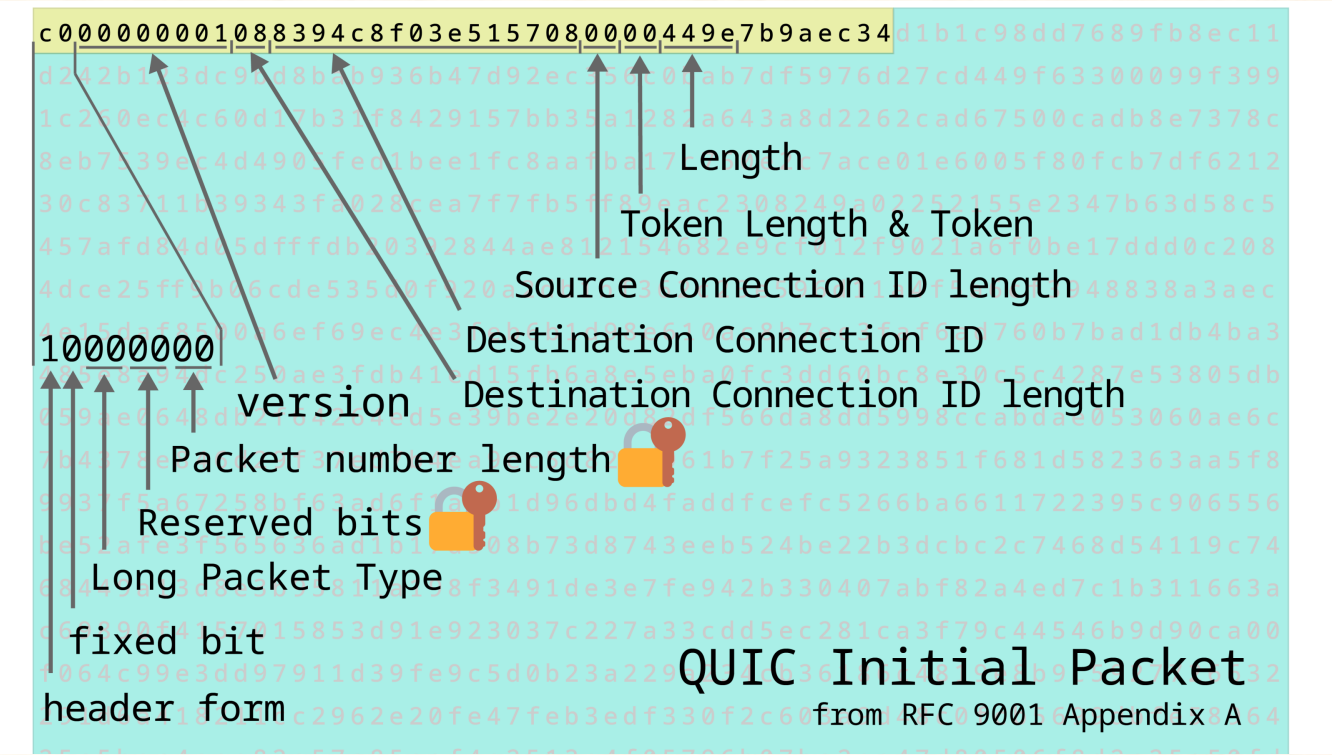
compare String and array of Integer

```
$ bundle exec ruby main.rb
Warming up --------------------------------------
  hello_world_upcase_string    848.029k i/s -    872.655k times in 1.029039s (1.18μs/i)
 hello_world_upcase_integer      1.268M i/s -      1.291M times in 1.018188s (788.58ns/i)
Calculating -------------------------------------
  hello_world_upcase_string    861.605k i/s -      2.544M times in 2.952730s (1.16μs/i)
 hello_world_upcase_integer      1.285M i/s -      3.804M times in 2.961585s (778.48ns/i)



Comparison:
 hello_world_upcase_integer :   1284551.4 i/s
  hello_world_upcase_string :    861605.1 i/s - 1.49x  slower
```

How to prevent mistake?

```
class UnpackedString < String
  def unpack1()
    raise RuntimeError
  end
end
```

It's an encrypted protocol



It's an encrypted protocol

compare String and array of Integer (re)

```
$ bundle exec ruby main.rb
Warming up --------------------------------------
  hello_world_upcase_string    848.029k i/s -    872.655k times in 1.029039s (1.18μs/i)
 hello_world_upcase_integer      1.268M i/s -      1.291M times in 1.018188s (788.58ns/i)
Calculating -------------------------------------
  hello_world_upcase_string    861.605k i/s -      2.544M times in 2.952730s (1.16μs/i)
 hello_world_upcase_integer      1.285M i/s -      3.804M times in 2.961585s (778.48ns/i)



Comparison:
 hello_world_upcase_integer :   1284551.4 i/s
  hello_world_upcase_string :    861605.1 i/s - 1.49x  slower
```

It's an encrypted protocol

leading "0" gone

```
"01001".to_i(16) # => 4097
"01001".to_i(16).to_s(16) # => "1001"
```

XOR between String

```ruby
def xor(a, b)
  a.unpack("C*").zip(b.unpack("C*")).map do |x, y|
    x ^ y
  end.pack("C*")
end
```

It's an encrypted protocol

```
data[0] = [(data[0].unpack1('H*').to_i(16) ^
  (mask[0].unpack1('H*').to_i(16) & 0x0f)).to_s(16)].pack("H*")



# https://www.rfc-editor.org/rfc/rfc9001#figure-6
pn_length = (data[0].unpack1('H*').to_i(16) & 0x03) + 1



packet_number =
  (data[pn_offset...pn_offset+pn_length].unpack1("H*").to_i(16) ^
    mask[1...1+pn_length].unpack1("H*").to_i(16)).to_s(16)



# fill zero because leading "0" gone
data[pn_offset...pn_offset+pn_length] =
  [("0" * (pn_length * 2 - packet_number.length)) + packet_number].pack("H*")
self.class.new(data, protected: false).tap {|p| p.parse}
```

XOR between byte (Python)

```
bytes([aa ^ bb for aa, bb in zip(a, b)])
```

https://programming-idioms.org/idiom/238/xor-byte-arrays/4146/python

What should we do?

If there is no constraint for "Pure Ruby"...

- Write extension library by system programming language
  - C or Rust
  - Rust is popular
    - https://github.com/rubygems/rubygems/pull/5613

      "Add support for bundle gem --rust command"

QUIC, Some headers and many frames

- two type of headers
  - long header, short header
- 20 types of frames
  - padding, ping, ack, etc...

QUIC, Some headers and many frames

```ruby
private def find_frame_type(frame)
  case [frame[0..7]].pack("B*")
  when "\x00"
    :padding
  when "\x01"
    :ping
  when "\x02"..."\x03"
    :ack
  # ......
```

Is it time for pattern matching? (This is a bad case. Too simple.)

But, I'm negative about introducing special class for bytes data

- It may break the existing code base
  - Ruby has 20+ years of history
  - "Bytes" is a very common noun, especially computer science
    - "bytes" gem was already taken 😠

So...

- Use another name of the String class
- Create own helper methods of bit operation
- Create extension library (if absolutely necessary)

Conclusion

- handling/manipulating binary data in Ruby is hard
  - than other languages that support bytes data as standard
  - bit operation is slow
- should convert data on encrypt/decrypt operations

Why did I choose the "Pure Ruby" way?

- To avoid problems coming from ractor or multithreading
  - Problems will appear near future...maybe...🤔