Breaking the 100 bits per second barrier with Matrix

An entirely new transport for Matrix for really terrible networks.

matthew@matrix.org
@matrixdotorg

Matrix is an open network for secure, decentralised real-time communication.

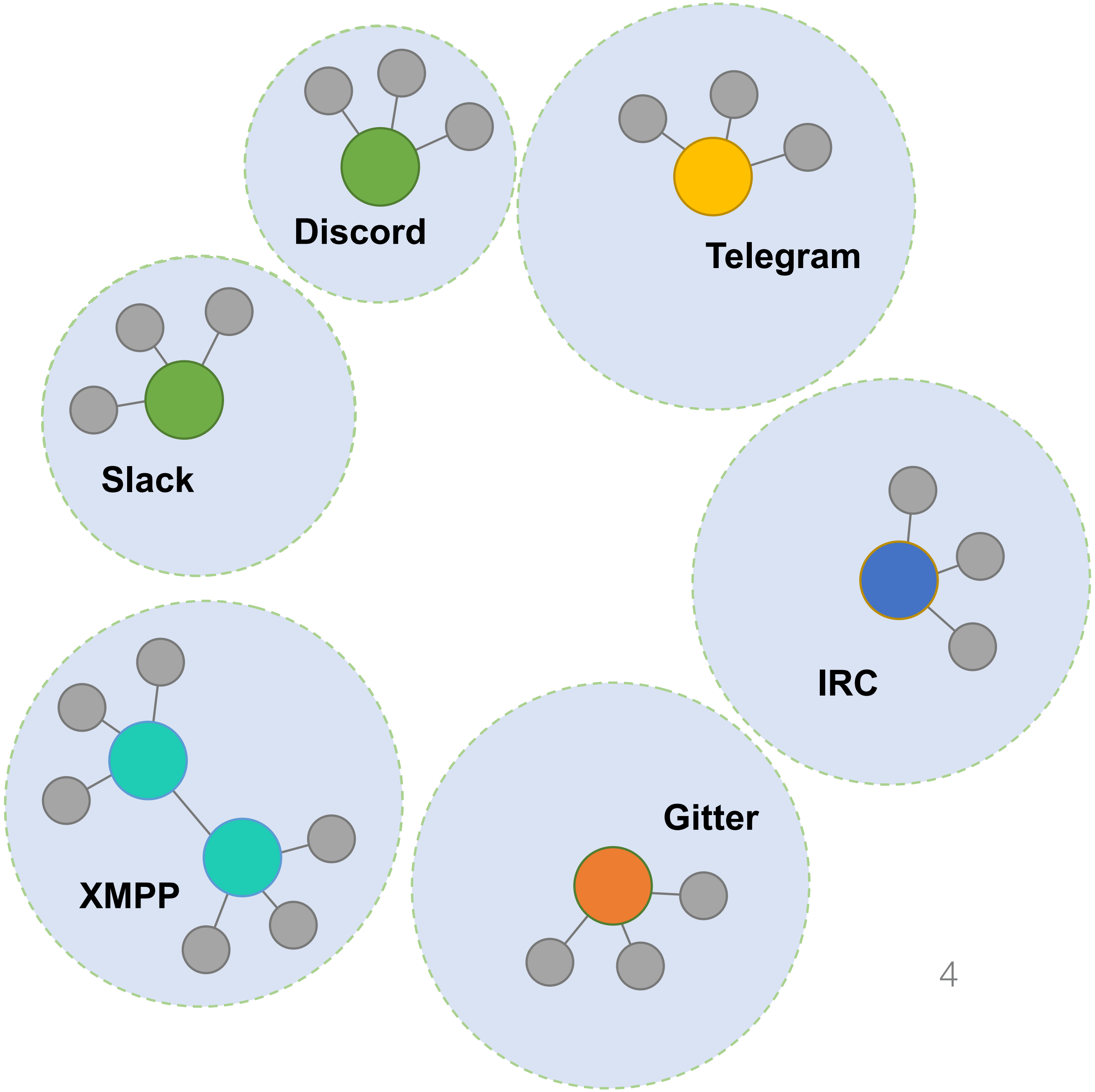
Interoperable chat

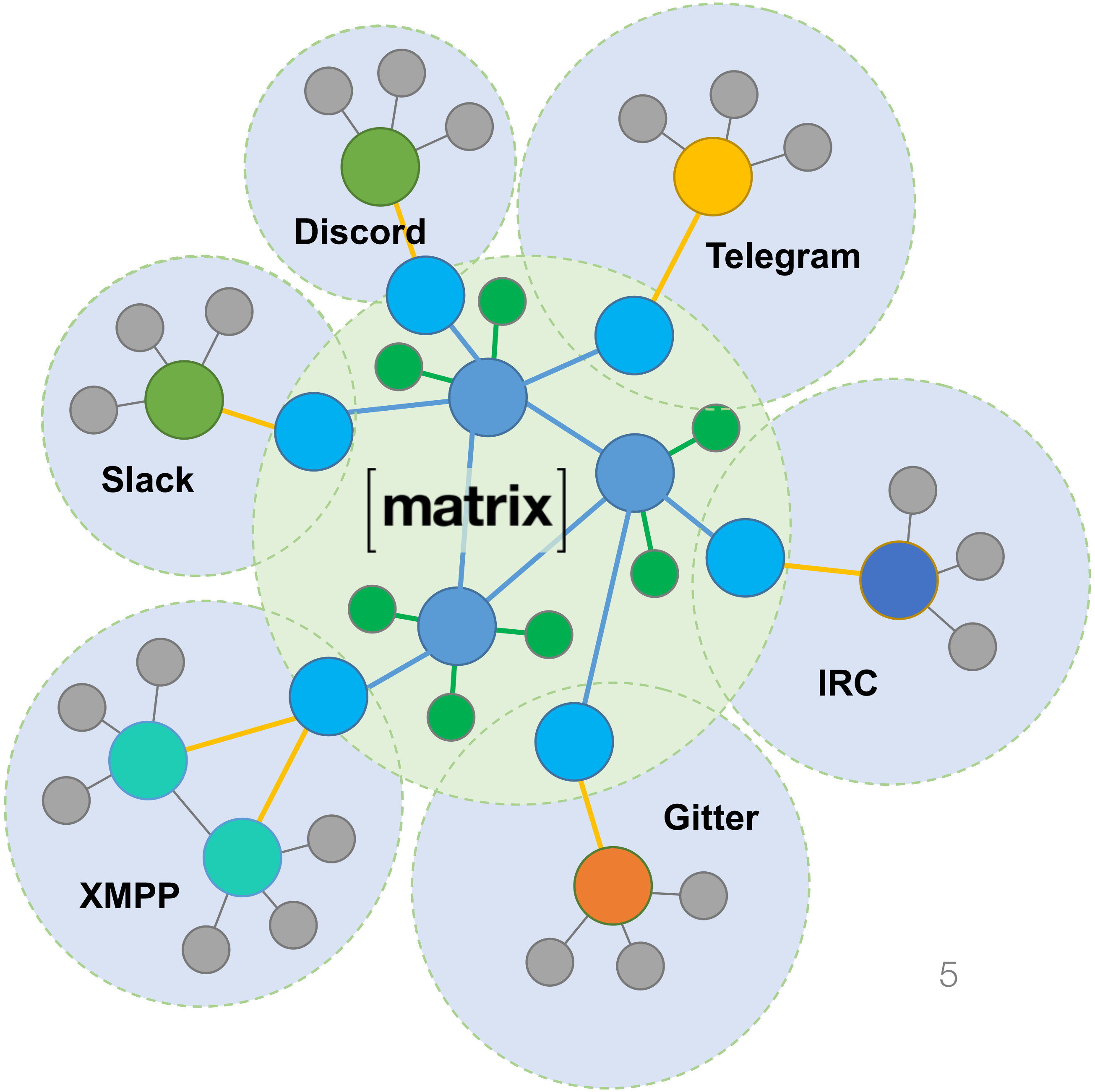
Interoperable VoIP

Open comms for VR/AR

Real-time IoT data fabric

Mission: to create a global decentralised encrypted comms network that provides an open platform for real-time communication.


Discord
Telegram
Slack
IRC
XMPP
Gitter


Discord
Telegram
Slack
[matrix]
IRC
XMPP
Gitter

[matrix]

No single party owns your conversations.

Conversations are shared over all participants.

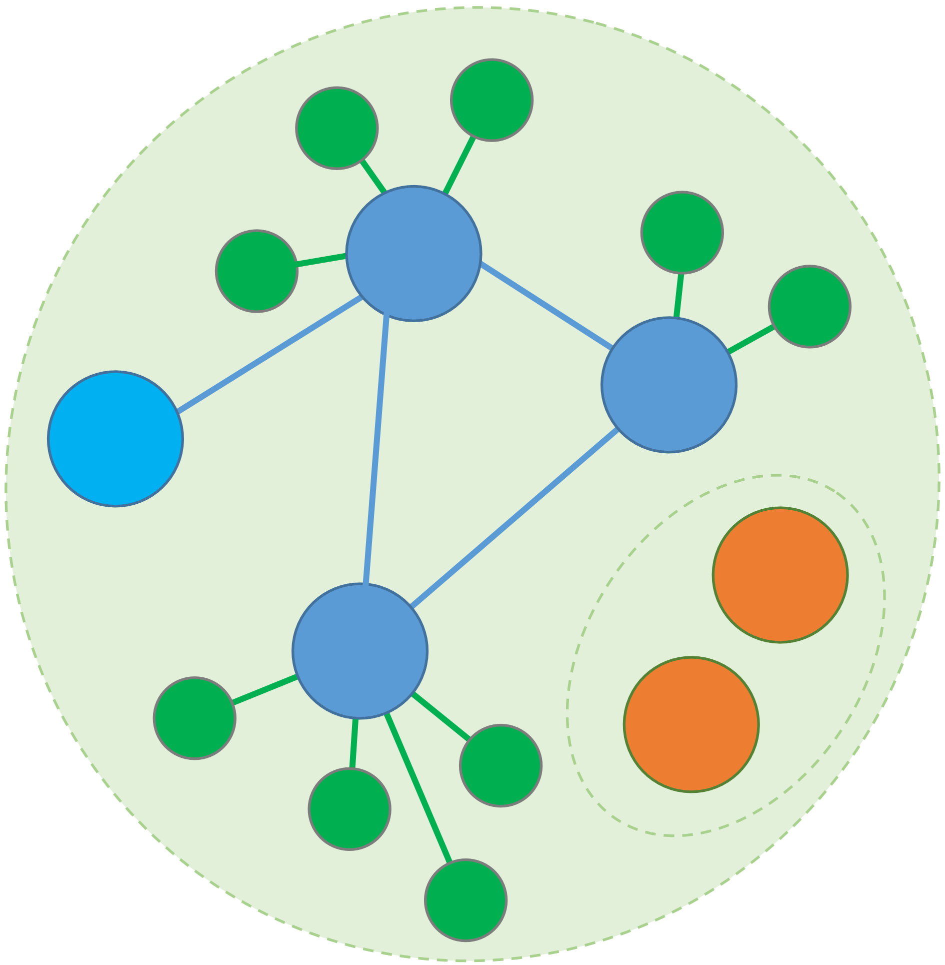
Matrix Architecture

🟢 Clients

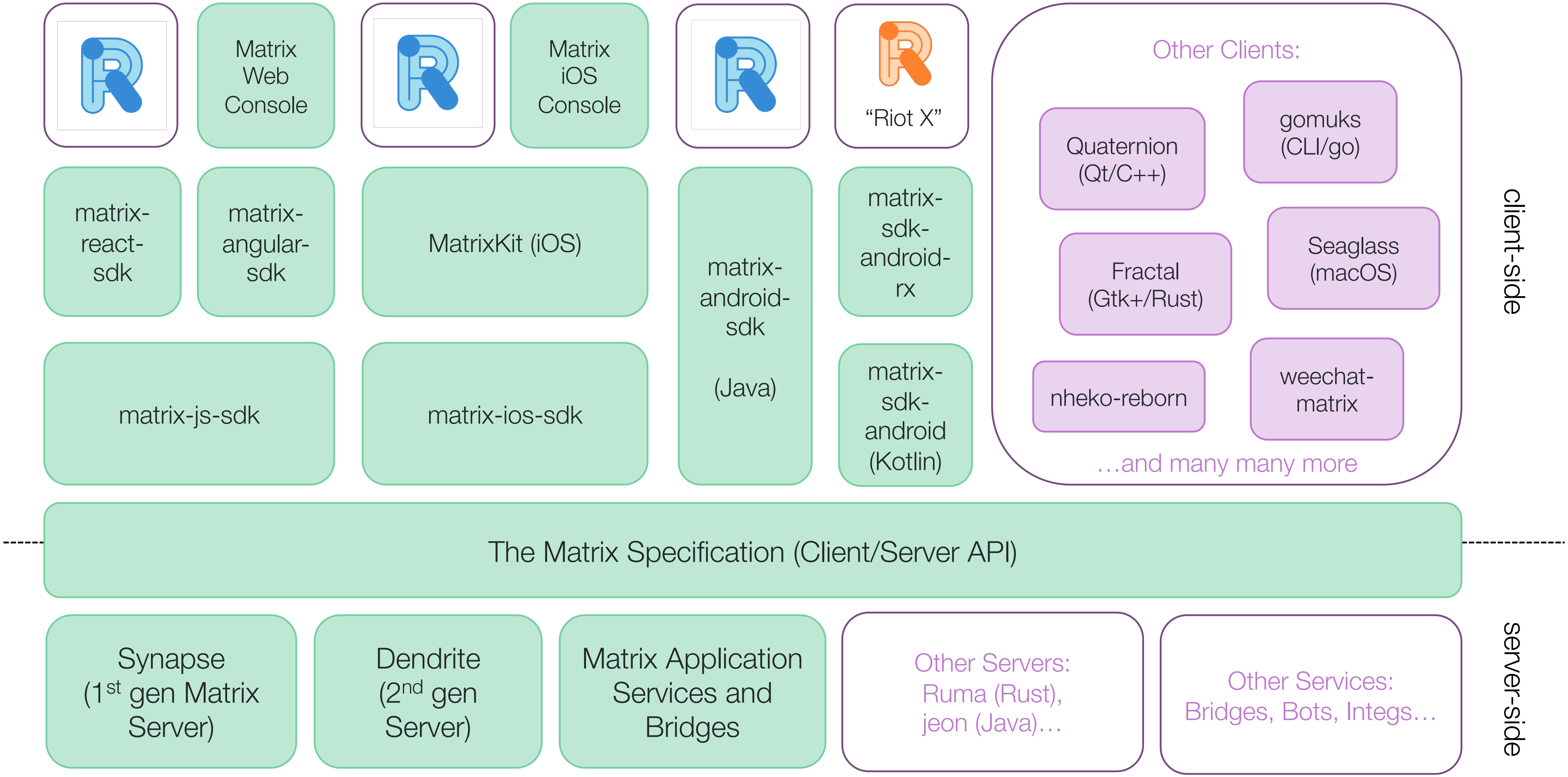
🔵 Home Servers

🔵 Application Servers

🟠 Identity Servers



Matrix Ecosystem



Matrix Web Console

Matrix iOS Console

"Riot X"

matrix-react-sdk

matrix-angular-sdk

MatrixKit (iOS)

matrix-android-sdk

(Java)

matrix-sdk-android-rx

matrix-js-sdk

matrix-ios-sdk

matrix-sdk-android (Kotlin)

Other Clients:

Quaternion (Qt/C++)

gomuks (CLI/go)

Fractal (Gtk+/Rust)

Seaglass (macOS)

nheko-reborn

weechat-matrix

…and many many more

client-side

The Matrix Specification (Client/Server API)

Synapse (1st gen Matrix Server)

Dendrite (2nd gen Server)

Matrix Application Services and Bridges

Other Servers: Ruma (Rust), jeon (Java)…

Other Services: Bridges, Bots, Integs…

server-side

Low Bandwidth Matrix

- Our target: 100bps.
- It takes 2 minutes to send an Ethernet packet (1500 MTU) at 100bps.
- Why would you do this?
- Connectivity can get pretty bad in life or death situations.
- If you are in appalling connectivity (e.g. the bottom of a crevasse) you want every bit to count.

HTTP/1.1+TLS

- Matrix is intended to be transport agnostic
- We started with HTTPS+JSON for convenience
- Any web developer can trivially send a message:

```
curl 'https://matrix.org/_matrix/client/r0/rooms/!foo:matrix.org/send/m.room.message/1’
     -X PUT --data '{"msgtype":"m.text","body":"test"}'
```

- Typical HTTP/1.1+TLS/1.2 request to send “test”
  - 7,004 bytes (including Eth headers)
  - 8 round trips.
  - 10 minutes to set up & send a msg at 100bps
- Obviously it could be so much better…

HTTP/2

- So what about HTTP/2?
- Add --http2 to curl…
- Now 6,737 bytes – we saved ~300 bytes :/

HTTP/3

- So what about HTTP/3? (HTTP over QUIC)
- We’re now over UDP + TLS/1.3
- Still have to do a TLS certificate handshake
- => Roundtrips reduced to 6 in total
- => ~6,700 bytes to send the same message.
- QUIC requires bit-stuffing to mitigate amplification attacks
- Once established, 983 bytes to send again
- Not ideal :/

CoAP

- CoAP is Constrained Application Protocol (RFC7252).
- Very very bit-efficient transport for RPC over UDP.
- Designed for Constrained devices and environments (e.g. IOT)
- Maps almost directly to HTTP (but isn’t HTTP).
- Typically expects a request to fit inside a single packet
- 1 roundtrip!
- ~500 bytes! (so only 40s to send a message!)
- Now we’re getting somewhere

CoAP+DTLS

- CoAP’s recommended encryption is DTLS+PSK.
- According to https://tools.ietf.org/id/draft-mattsson-lwig-security-protocol-comparison-01.html this can be as low as 15 bytes of TLS overhead.
- However, very few CoAP stacks support DTLS (especially in Go)
- Also, Private Shared Keys can be a hassle to admin.

CoAP+Noise

- Instead, we hooked up Noise to go-coap (from go-ocf).
- Noise is a set of building blocks for cryptography protocols.
- We chose to use the Noise Pipe pattern (XX and IK handshakes)
    - XX handshake lets you mutually authenticate and establish the public key for your peer over 1 roundtrip, which you can then cache.
    - IK handshake lets you reestablish a secure channel with 0RTT (if you already know the public key of your peer).
- Handshake is 32 bytes per token (roughly), and then 16 bytes auth tag overhead per msg.

XX:

```
  -> e
  <- e, ee, s, es
  -> s, se + payload
```

IK:

```
  -> e, es, s, ss + payload
  <- e, ee, se + payload
```

CoAP+CBOR+Noise

- But what about the payload?
- JSON is fairly bulky
  - echo ‘{"msgtype":"m.text","body":"test"}’ | wc –c
  - 35 bytes
- Switch to CBOR?
  - echo '{"msgtype":"m.text","body":"test"}' | perl -MCBOR::XS -MJSON::XS -pe'$_=encode_cbor(decode_json($_))’ | wc -c
  - 26 bytes.
- CBOR is generally about 75% smaller.

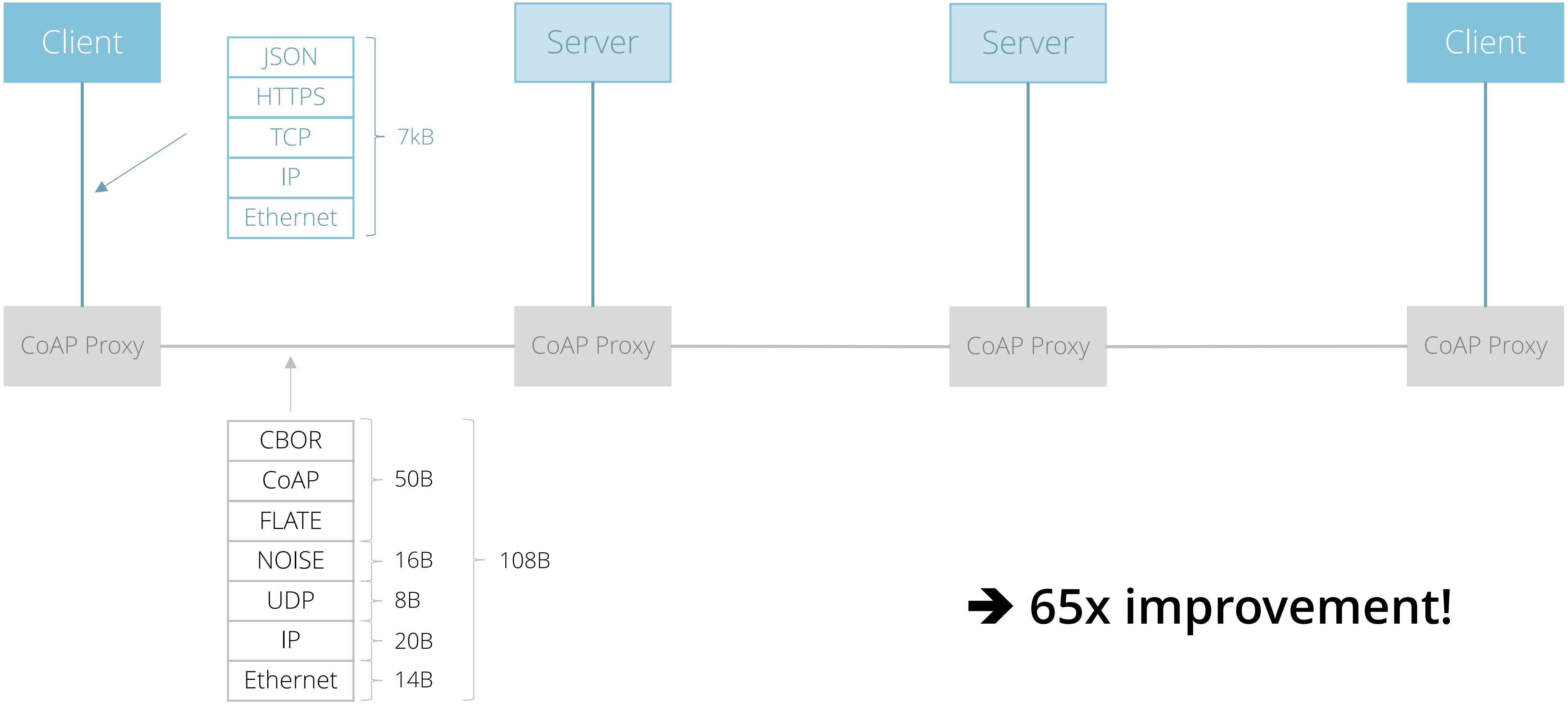
CoAP+CBOR+Deflate+Noise

- 75% isn't good enough.
- First let's improve the payload itself:
  - Map each HTTP URI to a numeric route ID for CoAP
  - Reduce the size of IDs (e.g. event IDs, room IDs, CoAP msg/token IDs)
- Manually mapping to shorter IDs gets a bit boring

⇒ Run everything through Deflate, with a preshared dictionary.

- Works excellently, but a bit questionable protocolwise.
- **~90 bytes (inc headers) + 16 bytes of crypto overhead.**
- **8 seconds to send a message at 100bps :D**

coap-proxy architecture



JSON	7kB
HTTPS	
TCP	
IP	
Ethernet	

CBOR	50B	108B
CoAP		
FLATE		
NOISE	16B	
UDP	8B	
IP	20B	
Ethernet	14B	

➔ 65x improvement!

Demo!

When can we use it?!

- Need to work a bit more on CoAP retry semantics.
- Need to ensure querystrings are < 255 bytes
- Need to ensure overlapping requests to the same endpoint don't get entangled.
- Need to sanitycheck blockwise CoAP + retries. A single missing block shouldn't kill the whole response.
- QUIC has loads of work on congestion control; CoAP doesn't.
- Need to decide what to do about Deflate.

Likely to be used in P2P Matrix experiments in future!

Code will be released on https://gitlab.matrix.org shortly.

We need help!!

DON'T USE PROPRIETARY SERVICES FOR YOUR CHAT.

- Run a server, or use a provider like modular.im
- Build bridges and bots to your services!
- Don't reinvent the wheel, use Matrix!
- Follow @matrixdotorg or @matrix@mastodon.matrix.org and spread the word!

matrix

Thank you!

@matthew:matrix.org
matthew@matrix.org
https://matrix.org
@matrixdotorg