

PyTAPS

Implementation Report

Max Franke
Philipp S. Tiesel
Theresa Enhardt
TAPS
IETF 104, March 2019, Prague

PyTAPS Principles

- Keep as close to the interface draft as possible
- Integrate with Python asyncio framework
- Build A Fully asynchronous API using coroutines

Implementation status

- What works so far
 - TCP, UDP, TLS (1.3)
 - Candidate selection for protocols
- What we want to do next:
 - Framers
 - Interface selection
 - Pooled connections
 - QUIC
 - Racing

Minimal Client

```
import asyncio
import PyTAPS as taps

class TestClient():
    async def handle_ready(self, connection):
        msgref = await self.connection.send_message("Hello\n")

    async def main(self):
        ep = taps.RemoteEndpoint()
        ep.with_address("localhost")
        ep.with_port(6666)
        tp = taps.TransportProperties()

        tp.prohibit("reliability")
        tp.ignore("congestion-control")
        tp.ignore("preserve-order")

        self.preconnection = taps.Preconnection(remote_endpoint=ep,
                                                transport_properties=tp)
        self.preconnection.on_ready(self.handle_ready)
        self.connection = await self.preconnection.initiate()

if __name__ == "__main__":
    client = TestClient()
    asyncio.get_event_loop().create_task(client.main())
    asyncio.get_event_loop().run_forever()
```

Minimal Server

```
import asyncio
import PyTAPS as taps

class TestServer():
    async def handle_connection_received(self, connection):
        connection.on_received(self.handle_received)
        await connection.receive()

    async def handle_received(self, data, context, connection):
        print(data)

    async def main(self):
        lp = taps.LocalEndpoint()
        lp.with_address("localhost")
        lp.with_port(6666)
        tp = taps.TransportProperties()

        tp.prohibit("reliability")
        tp.ignore("congestion-control")
        tp.ignore("preserve-order")

        self.preconnection = taps.Preconnection(local_endpoint=lp,
                                                transport_properties=tp)
        self.preconnection.on_connection_received(
            self.handle_connection_received)
        await self.preconnection.listen()

if __name__ == "__main__":
    server = TestServer()
    asyncio.get_event_loop().create_task(server.main())
    asyncio.get_event_loop().run_forever()
```

Things that work well

- TAPS API seems easy and *mostly* intuitive to use
- Ability to swap protocols without changing application code
- Minimize protocol specific code in the implementation

Things that we have noticed:

Defaults

- Recommended default properties too restrictive
 - Lack of specified properties indicates lack of constraints the app has for connections
 - Should have the highest amount of possible candidates to make sure a connection can be established
 - So why force us on TCP then?!

Things that we have noticed:

Hard to force a specific protocol

- Dependencies of transport properties make it unintuitive to change to a specific protocol
 - To change to UDP, application has to fiddle with at least 3 properties

```
tp.prohibit("reliability")  
tp.ignore("congestion-control")  
tp.ignore("preserve-order")
```

- Makes API harder to use for these cases

Things we have noticed:

Handling unsatisfiable configuration

- When to check for unsatisfiable configurations?
 - Check it every time a property gets set
 - possibly lots of errors
 - Check it only after initiate/listen has been called
 - works, but possibly too late
 - Address/Discuss in Implementation draft?
- Application can not distinguish between unsatisfiable configuration and network issues
 - Should have its own error type

Things we have noticed:

Properties

- Hard to tell which protocol satisfies what property
 - Does UDP satisfy the 0RTT property?
- How to treat optional features?
 - Just turn them on/off accordingly?
 - What happens if we can't turn them on/off reliably?
 - ❖ If multistreaming is prohibited should SCTP be excluded?

Things that we have noticed:

Miscellaneous

- Interaction between multiple receives and `min_incomplete_length`
- Having to call `receive` for each incoming messages can be inconvenient
- `msgref` is never clearly defined, only appears in callbacks in interface draft

