An HTTPS-based Transport for YANG Notifications

draft-ietf-netconf-https-notif-15

Collector Implementations

Team : Bharadwaja Meherrushi C, Siddharth Bhat, Hayyan Arshad, Vartika T Rao, Mohit P Tahiliani (NITK)
Draft authors : Mahesh Jethanandani, Kent Watsen

An HTTPS-based Transport for YANG Notifications

- A protocol for **sending asynchronous event notifications** similar to notifications defined in NETCONF Event Notifications [RFC5277], but over HTTPS.
- Interactions will occur between a **publisher** which must be a "server" (e.g., a NETCONF or RESTCONF server) and a **receiver** (which need not be a NETCONF or RESTCONF server).

- Two YANG modules are defined which **extend the data model defined in Subscription to YANG Notifications** [RFC8639], enabling the configuration of HTTPS-based receivers

Why HTTPS?

Using HTTPS, which is a secure form of HTTP Semantics [RFC9110], **maximizes transport-level interoperability**, while allowing for a **variety of encoding options** (currently supports JSON and XML as encoding options)

While the payload does not change between these versions of HTTP and HTTP/3 [RFC9114], the underlying transport does. Since NETCONF does not support QUIC: A UDP-Based Multiplexed and Secure Transport [RFC9000], support for HTTP/3 [RFC9114], is considered out of scope of this document.

Current implementation supports JSON and XML encoding as options (i.e content-type header)

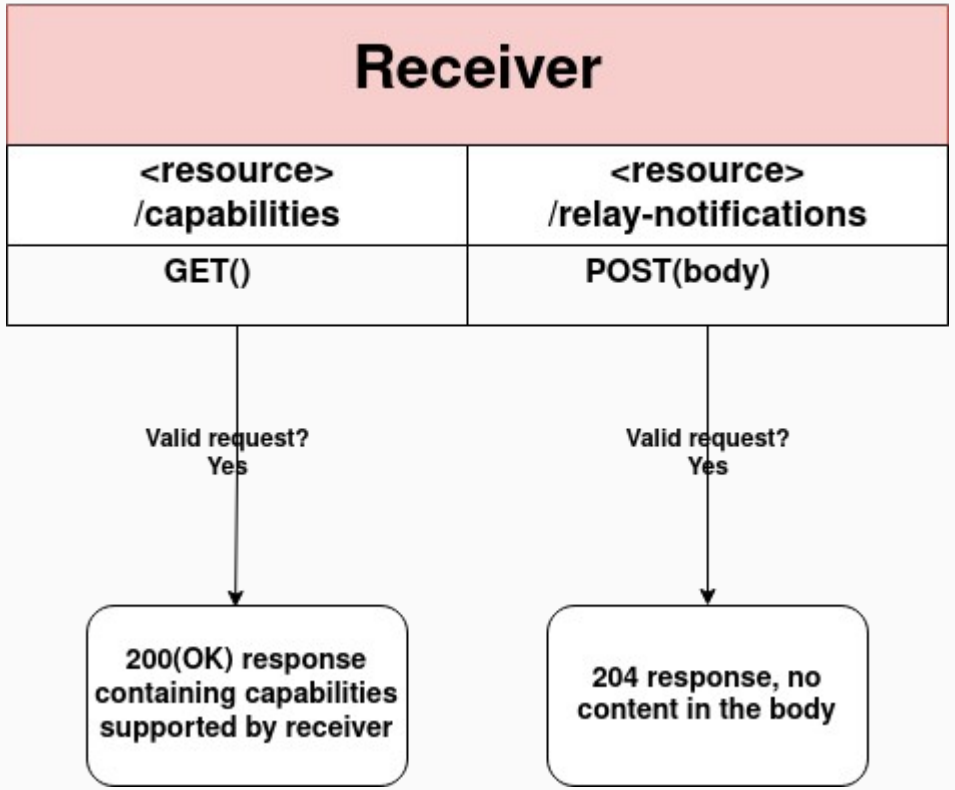
This can be extended to include other encoding options.

XML-encoded notifications are encoded using the format defined by NETCONF Event Notifications [RFC5277] for XML.

JSON-encoded notifications are encoded the same as specified in Section 6.4 in RESTCONF [RFC8040] with some deviations

Overview of the collector

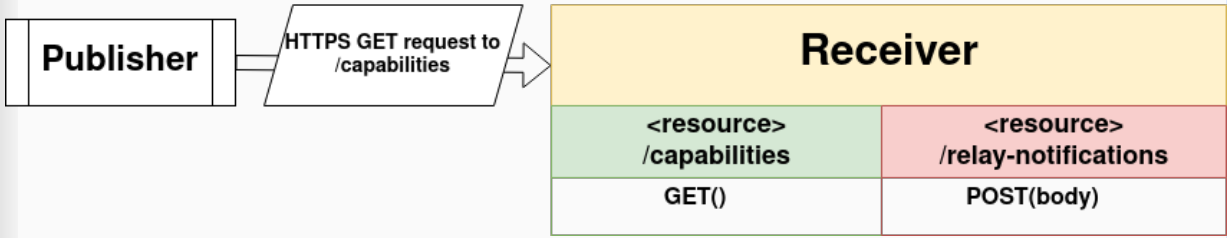
- Two HTTP based target resources are presented by the receiver. They are :
  - /capabilities - allows the publisher to discover what encoding the receiver supports.
  - /relay-notifications - allows the publisher to send one or more notifications to the receiver.



How does the protocol work?

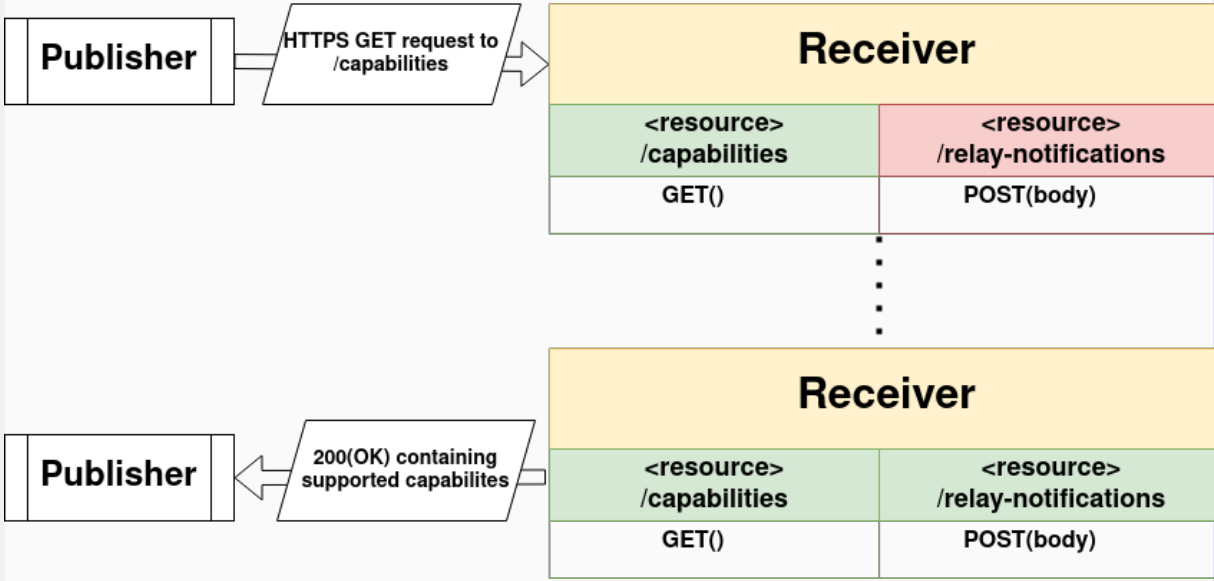
Working

- The publisher **MUST** query the targeted receiver(s) at the /capabilities resource initially, before sending any notifications (or if any error occurs).
- The “Accept” header must be set using application/json and/or application/xml



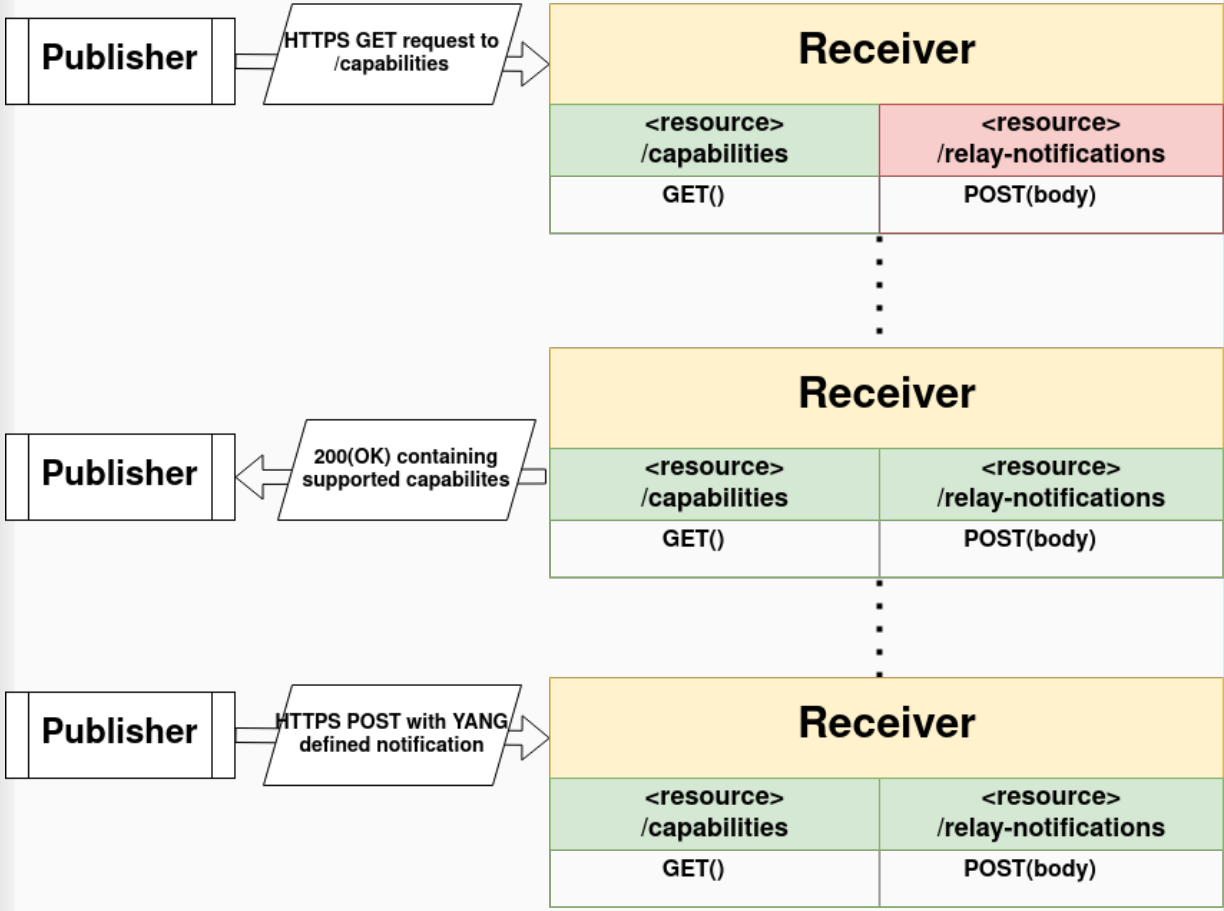
Working

- The receiver responds to the publisher with a 200(OK), the body **containing the capabilities supported by the receiver** in the appropriate encoding format.



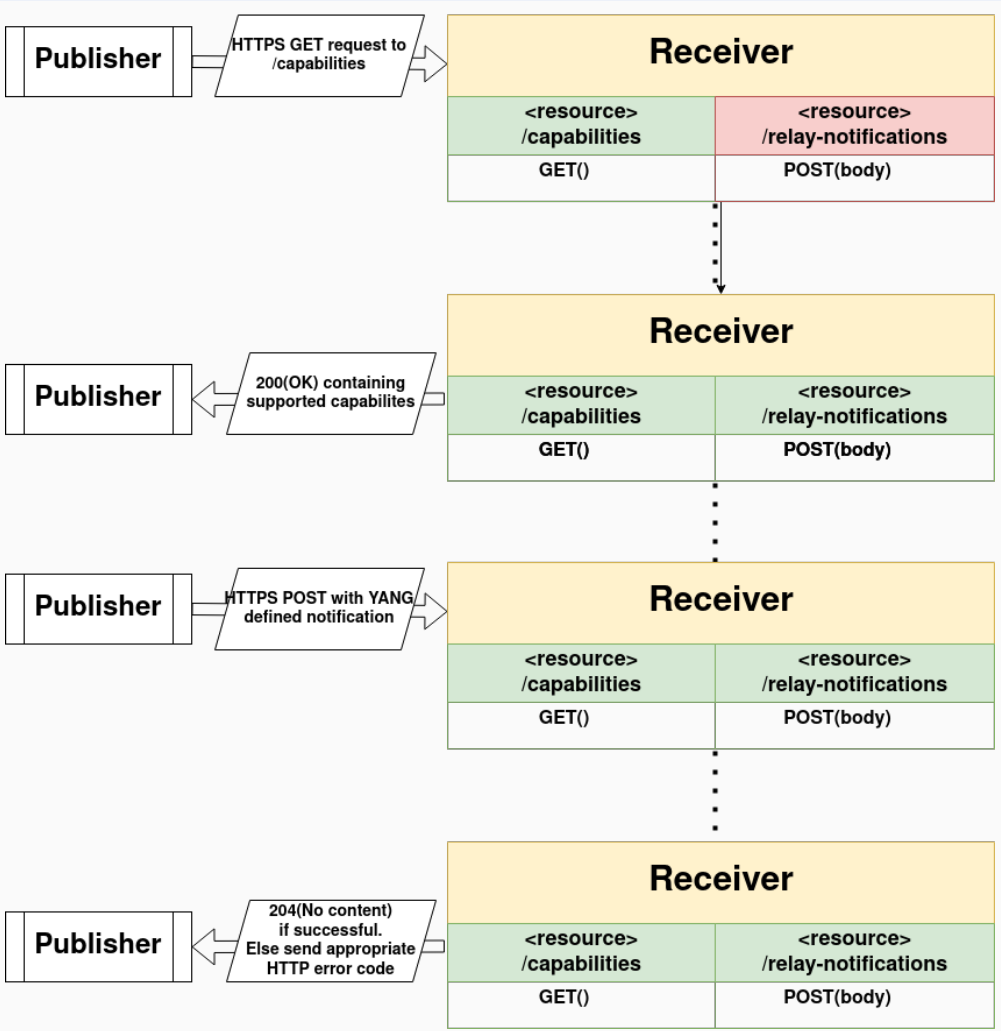
Working

- The publisher can now send notifications using a HTTPS POST request to the /relay-notifications resource.
- The “Content-Type” header set to either application/json or application/xml , depending upon what capabilities the receiver has(which is learnt from the response to the GET request to /capabilities)



Working

- The receiver can respond with a 204(No Content) in a successful scenario.
- In case of a corrupted or malformed event, the response should be an appropriate HTTP error response

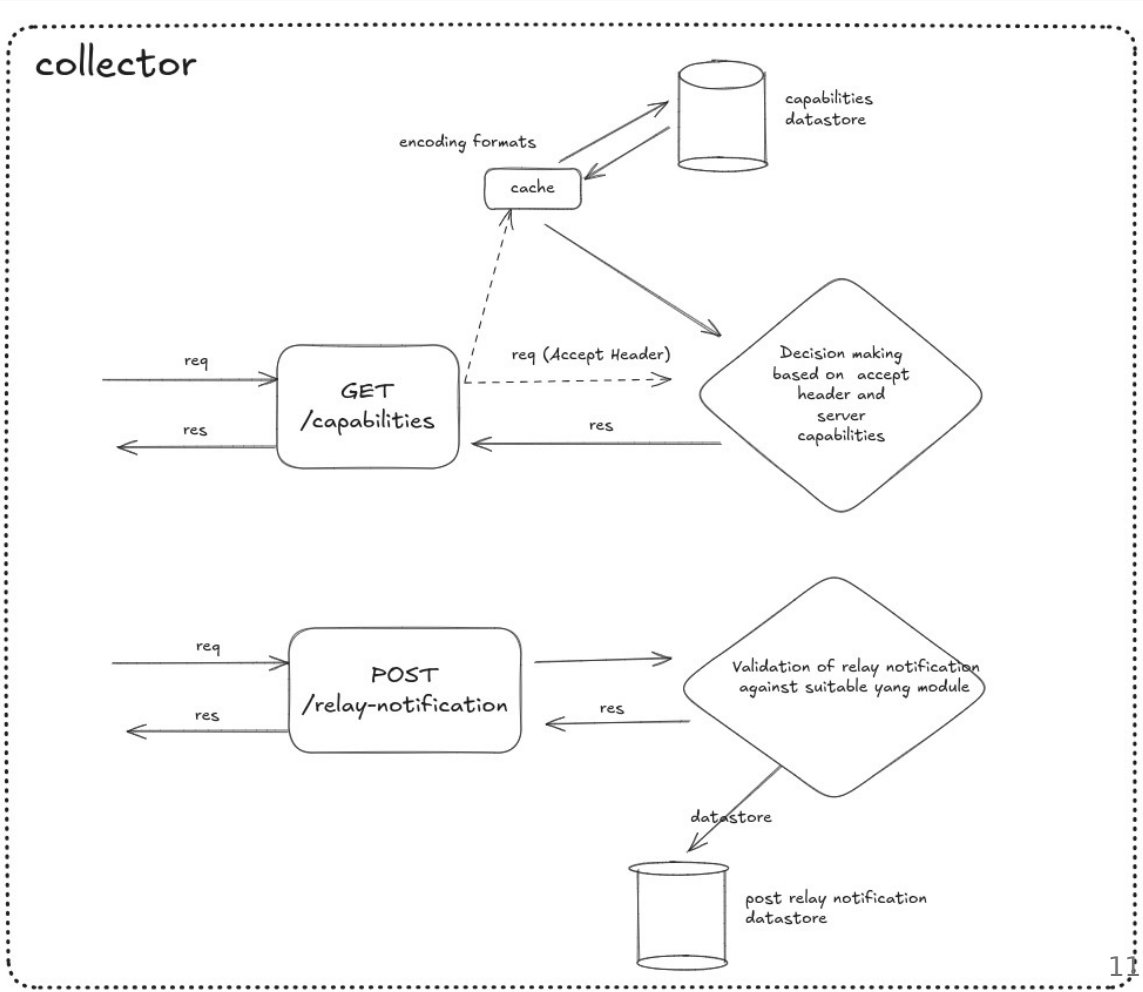


Implementation details?

Github Links :
C impl
Python Impl

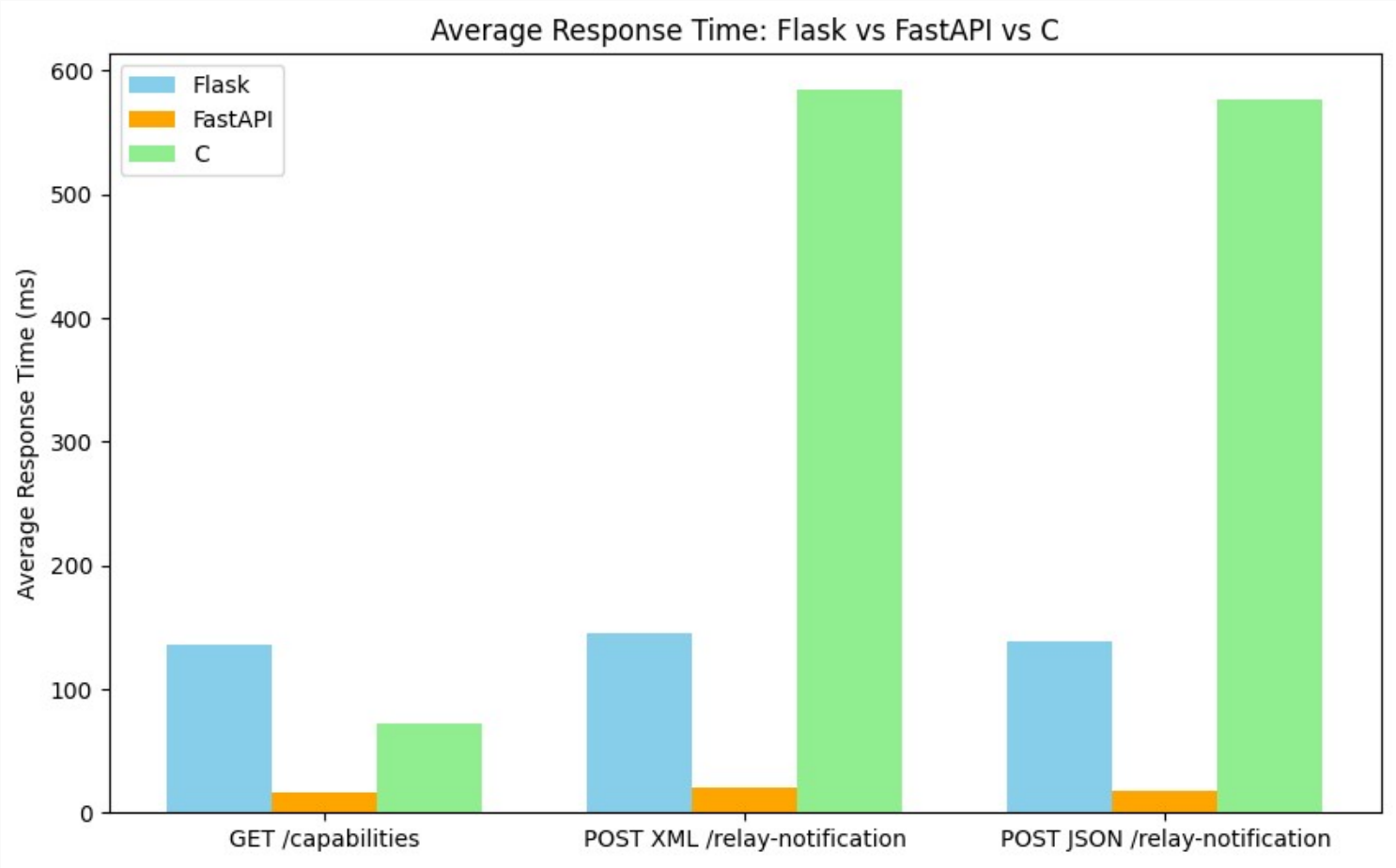
Implementation Overview

- We need a basic web server and a request handler and simple application logic and yang validation tool
- We have implementations in
- Python :
  - flask - synchronous
  - fast API asynchronous
  - We have used **Gunicorn** server to enable **multithreading**
  - Yangson for validation
- C *:
  - libmicrohttpd
  - libyang for validation

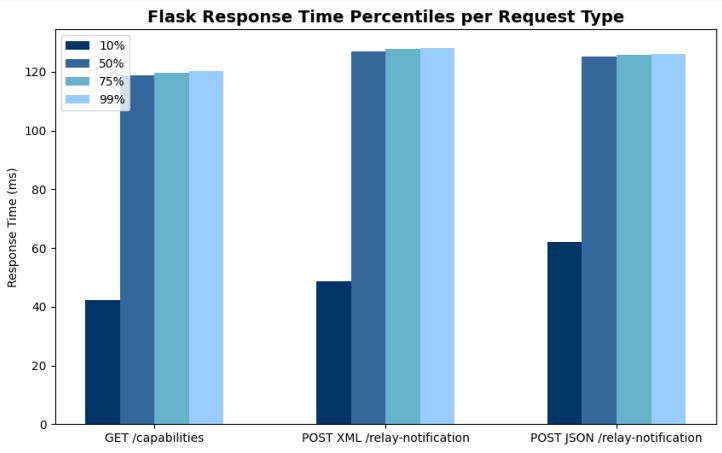


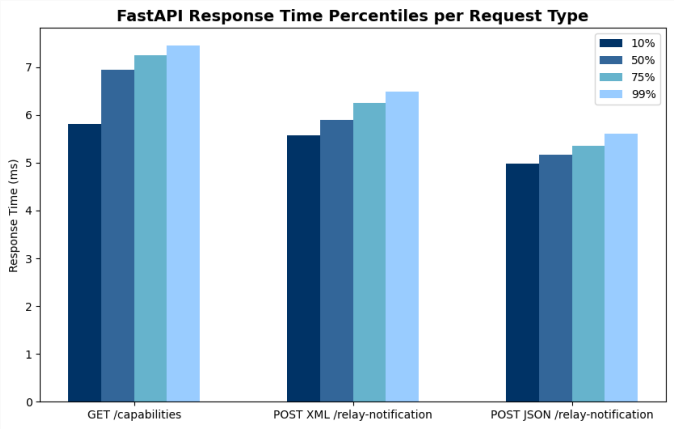
Results & Analysis of Collector Performance

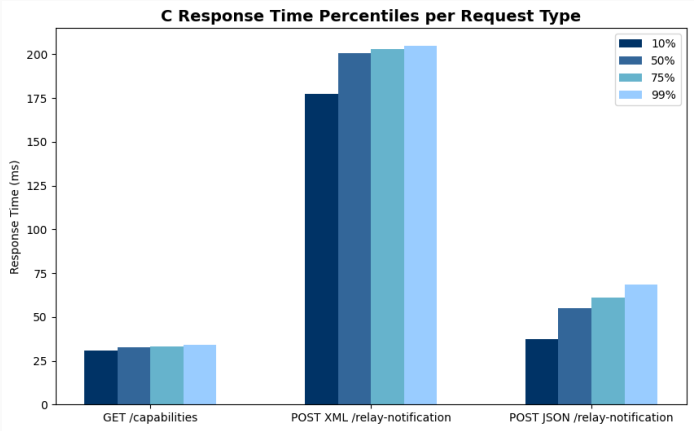
(Experimental Setup details : link)



A Comparison between average response time of servers implemented in Flask , FastAPI and C


Flask Response Time Percentiles per Request Type
10%
50%
75%
99%
Response Time (ms)
0
20
40
60
80
100
120
GET /capabilities
POST XML /relay-notification
POST JSON /relay-notification


FastAPI Response Time Percentiles per Request Type
10%
50%
75%
99%
Response Time (ms)
0
1
2
3
4
5
6
7
GET /capabilities
POST XML /relay-notification
POST JSON /relay-notification


C Response Time Percentiles per Request Type
10%
50%
75%
99%
Response Time (ms)
0
25
50
75
100
125
150
175
200
GET /capabilities
POST XML /relay-notification
POST JSON /relay-notification

Challenges and Future Work

Challenges Faced

Finding a Suitable YANG Verification Tool for different languages

Understanding YANG : Steep learning curve in comprehending YANG data modeling language.

Bridging the Gap Between Implementation and Theoretical descriptions: Challenges in translating theoretical RFC into practical, functional implementations.

Lots of possible variations in implementation and time spent in alternate implementations For eg : we got sidetracked into Data Storage at the /GET endpoint (pertaining to restconf standards)

Future Work

1. Extensive Testing
2. Integration with Grafana and Prometheus
3. Data Generation Using Linux Virtual Devices & Sysrepo

Thank you!

Any questions?

More Detailed slides : Link