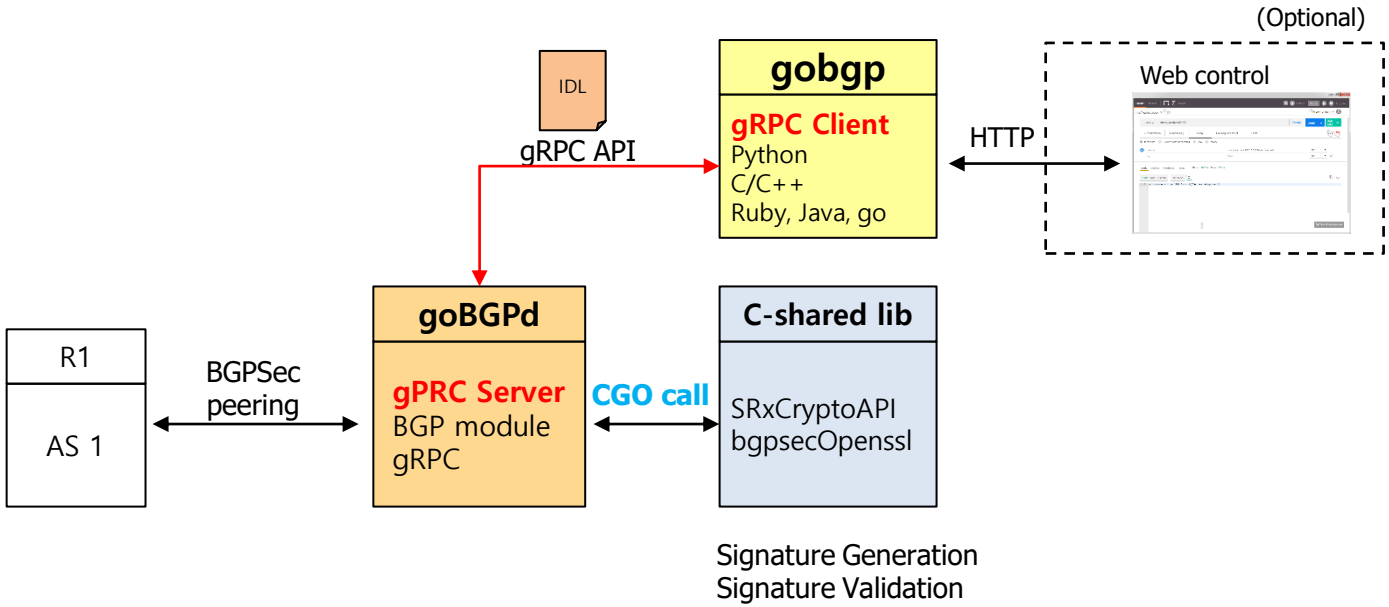
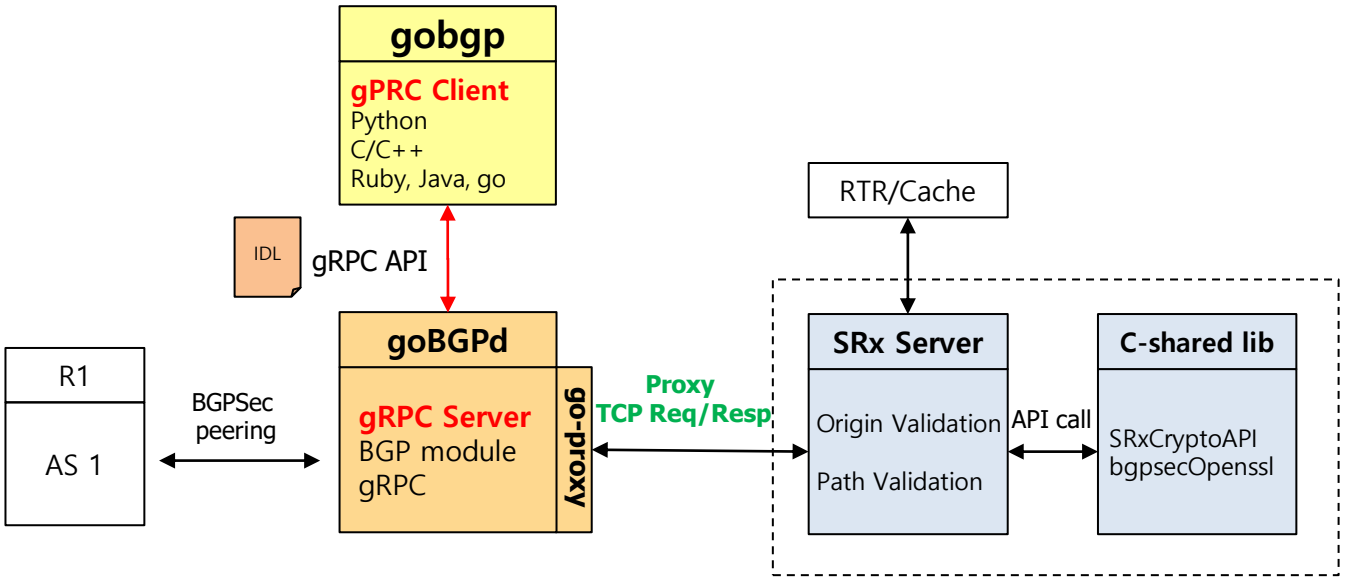
GoBGP: Inter-operation Scenarios (1)
CGO with dynamic library



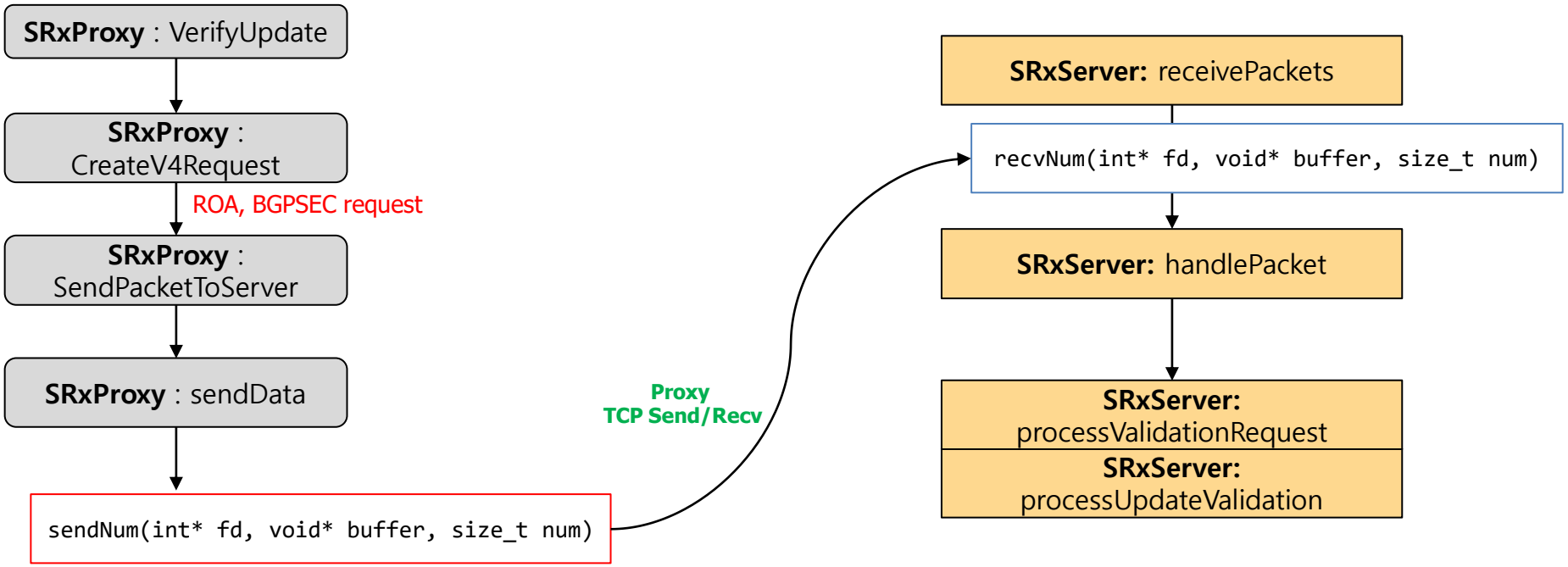
- **InterOps with CGO module**
  - Importing C/C++ binary shared library using CGO
    - Go language can incorporate C code directly or use libraries with cgo (https://golang.org/cmd/cgo/, http://golang.org/doc/articles/c_go_cgo.html)

GoBGP: Inter-operation Scenarios (2)
proxy call with SRx Server



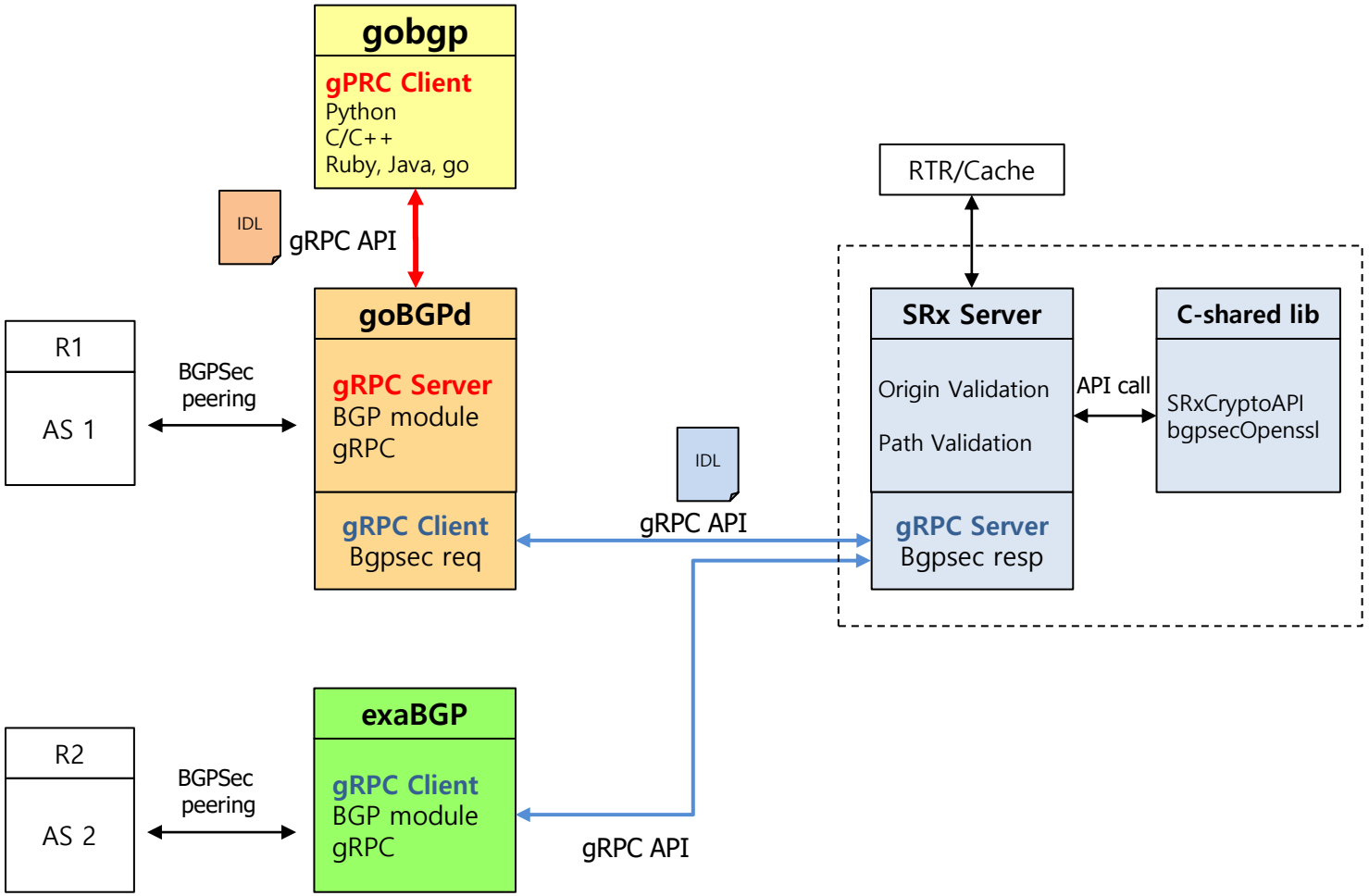
- **InterOps with go-proxy module into NIST SRx-server**
  - Requiring go-proxy module for go's TCP message exchange

SRxProxy's TCP Send/Recv Internals



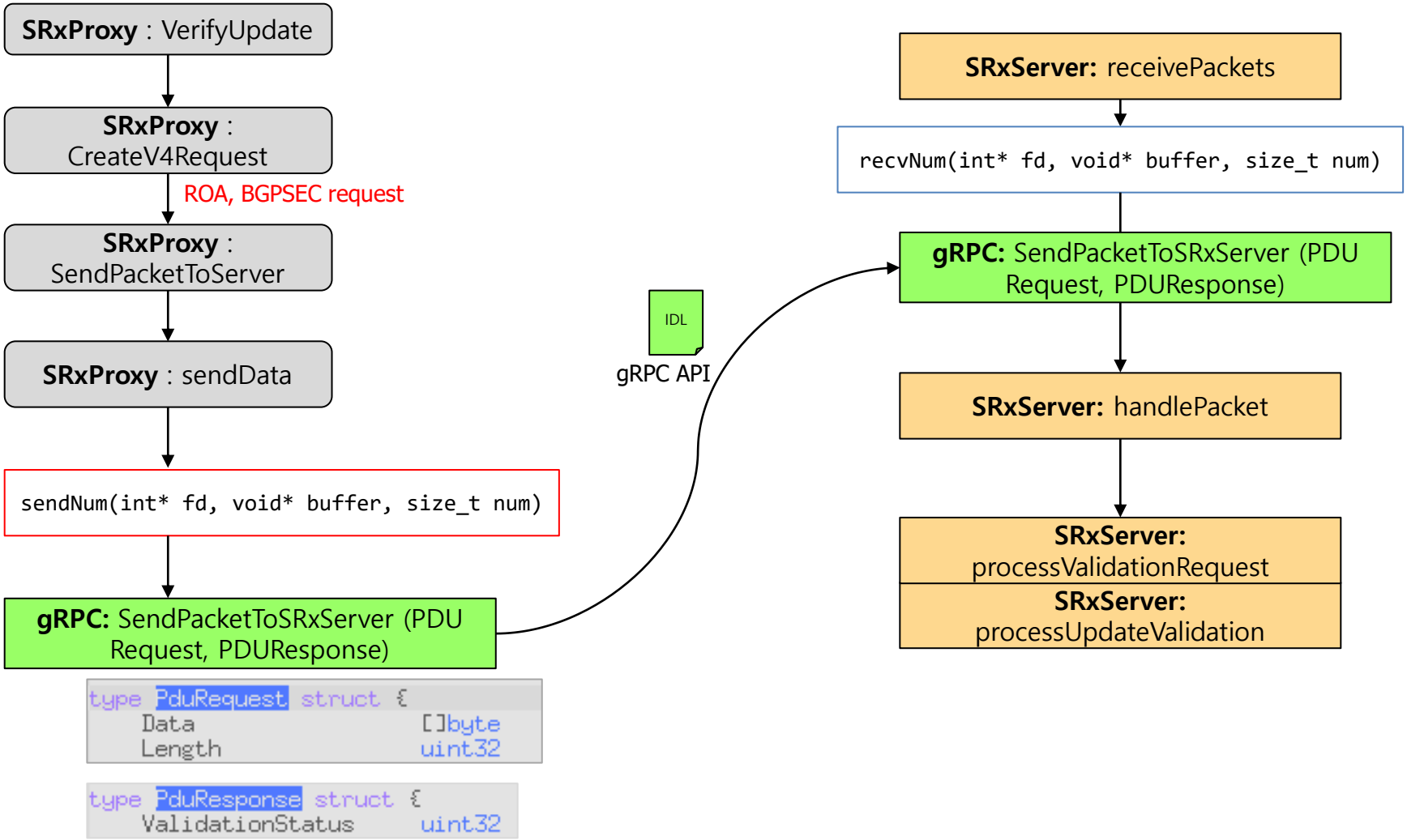
GoBGP: Inter-operation Scenarios (3)

gRPC with SRx Server



- **may use gRPC server for SRx Suite**
  - IDL needs be defined for go-proxy method with proto-buffer
  - language's serializing program

SRxProxy's gRPC exchange Internals



SRxProxy's gRPC exchange Example(1)

IDL **srxapi.proto** **gRPC IDL Definition**

```
syntax = "proto3";
package srxapi;

// Interface exported by the server.
service SRxApi {
  rpc SendPacketToSRxServer(PduRequest)
               returns (PduResponse) {}
}

message PduRequest {
  bytes data = 1;
  uint32 length = 2;
}

message PduResponse {
  uint32 validation_status = 1;
}
```

Protocol Buffer compile

srxapi.pb.{cc, go} **gRPC protocol buffer file generated**

```
// For semantics around ctx use and closing/ending streaming RPCs, please refer to https
type SRxApiClient interface {
    SendPacketToSRxServer(ctx context.Context, in *PduRequest, opts ...grpc.CallOption)
}

type sRxApiClient struct {
    cc *grpc.ClientConn
}

func NewSRxApiClient(cc *grpc.ClientConn) SRxApiClient {
    return &sRxApiClient{cc}
}

func (c *sRxApiClient) SendPacketToSRxServer(ctx context.Context, in *PduRequest, opts
    out := new(PduResponse)
    err := c.cc.Invoke(ctx, "/srxapi.SRxApi/SendPacketToSRxServer", in, out, opts...)
    if err != nil {
        return nil, err
    }
    return out, nil
}

// SRxApiServer is the server API for SRxApi service.
type SRxApiServer interface {
    SendPacketToSRxServer(context.Context, *PduRequest) (*PduResponse, error)
}
```

srxapi_server.{cc, go}

Server driver (C++, go, python etc)

Making binary library

srxapi_server.so BIN

Dynamic Call

Server Application (SRx Server, Quagga etc)

Applications those who DO NOT have gRPC ability

srxapi_client.{cc,go}

Client driver (C++, go, python etc)

Making binary library

BIN **srxapi_client.so**

Dynamic Call

Client Application (Quagga, BIRD, ExaBGP etc)

Applications those who DO NOT have gRPC ability

gRPC message transfer

SRxProxy's gRPC exchange Example(2)

CImple_srxapi_server.c

```
#include <stdio.h>
#include "srxapi_server.h"
int main()
{
    printf("using srxapi_server library from C\n");
    Serve();
}
```

Compile with gRPC lib

BIN CImple_srxapi_server

```
vmware.005-antd [1918]{.../srxapi/server3-> ldd CImple_srxapi_server
        linux-vdso.so.1 =>  (0x00007ffce5b3d000)
        ./srxapi_server.so (0x00007fbc5595c000)
        libc.so.6 => /lib64/libc.so.6 (0x00007fbc5558f000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007fbc55373000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fbc5644a000)
```

gRPC server driver lib

Running server application

```
vmware.005-antd [1996]{.../srxapi/server3-> ./CImple_srxapi_server
using srxapi_server library from C
server: &{srxapi PduRequest{Data:[]uint8{0xab, 0xcd, 0xef}, Length:0x3,
recognized:[]uint8(nil), XXX_sizecache:0}
```

CImple_srxapi_client.c

```
#include <stdio.h>
#include "srxapi_client.h"
#include <stdlib.h>
int main () {
    printf(" Running C imple grpc client from C\n");

    char buff[10];
    buff[0] = 0xAB;
    buff[1] = 0xCD;
    buff[2] = 0xEF;
    GoSlice pdu = {(void*)buff, (GoInt)3, (GoInt)10};

    int32_t result = Run(pdu);
    printf(" validation result: %02x\n", result);
    return 0;}
```

Compile with gRPC lib

BIN CImple_srxapi_client

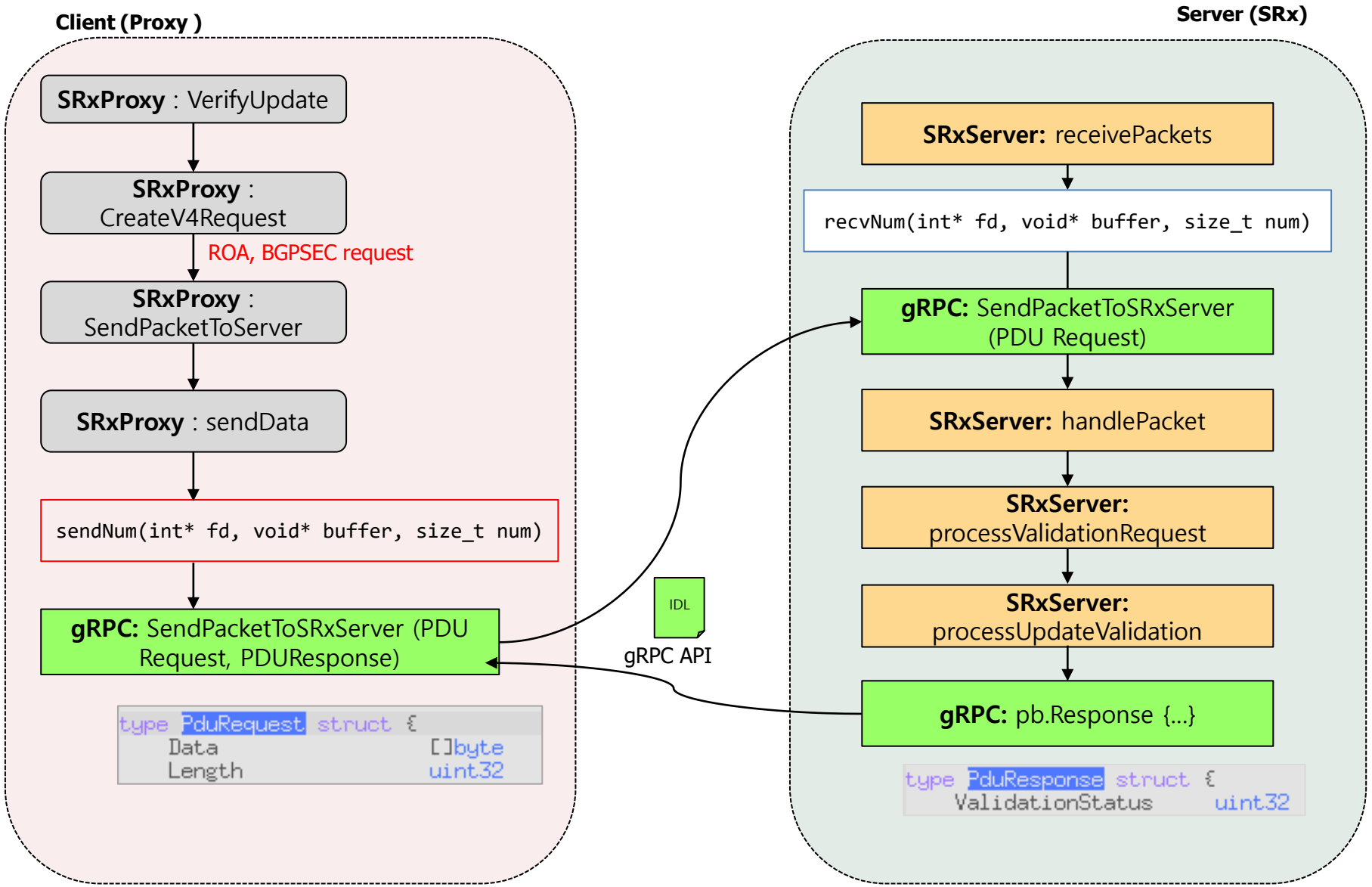
```
vmware.005-antd [1996]{.../srxapi/client3-> ldd CImple_srxapi_client
        linux-vdso.so.1 =>  (0x00007ffdf795b000)
        ./srxapi_client.so (0x00007f29bef5f000)
        libc.so.6 => /lib64/libc.so.6 (0x00007f29beb92000)
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00007f29be976000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f29bfa09000)
```

gRPC client driver lib

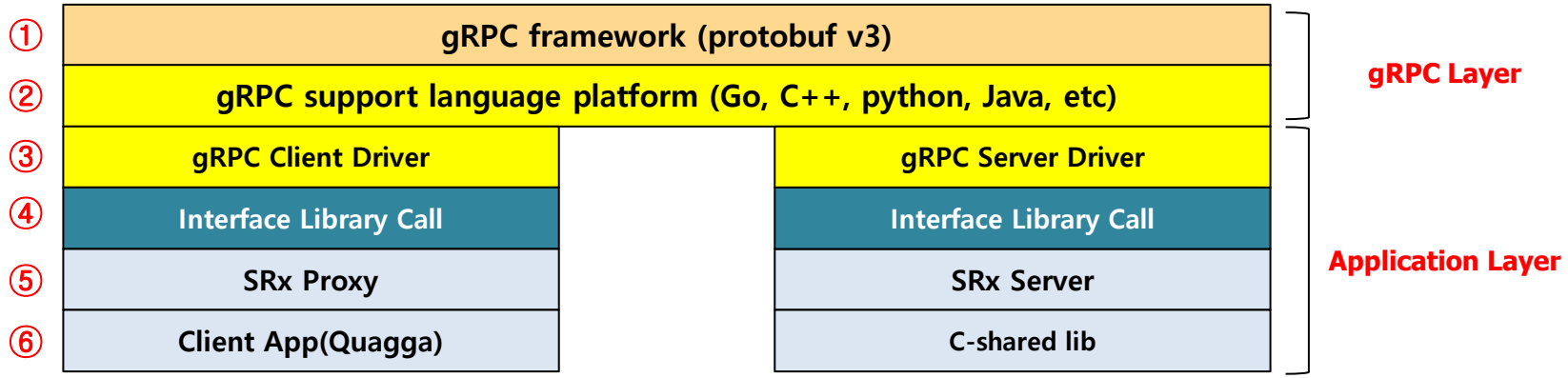
Running client Application

```
vmware.005-antd [1996]{.../srxapi/client3-> ./CImple_srxapi_client
 Running C imple grpc client from C
input data: []byte{0xab, 0xcd, 0xef}
status: 0x7
 validation result: 07
```

SRxProxy's gRPC exchange Internals



Layers of gRPC-enabled SRx Suite



① gRPC serialized traffic exchanges (protocol buffer v3)

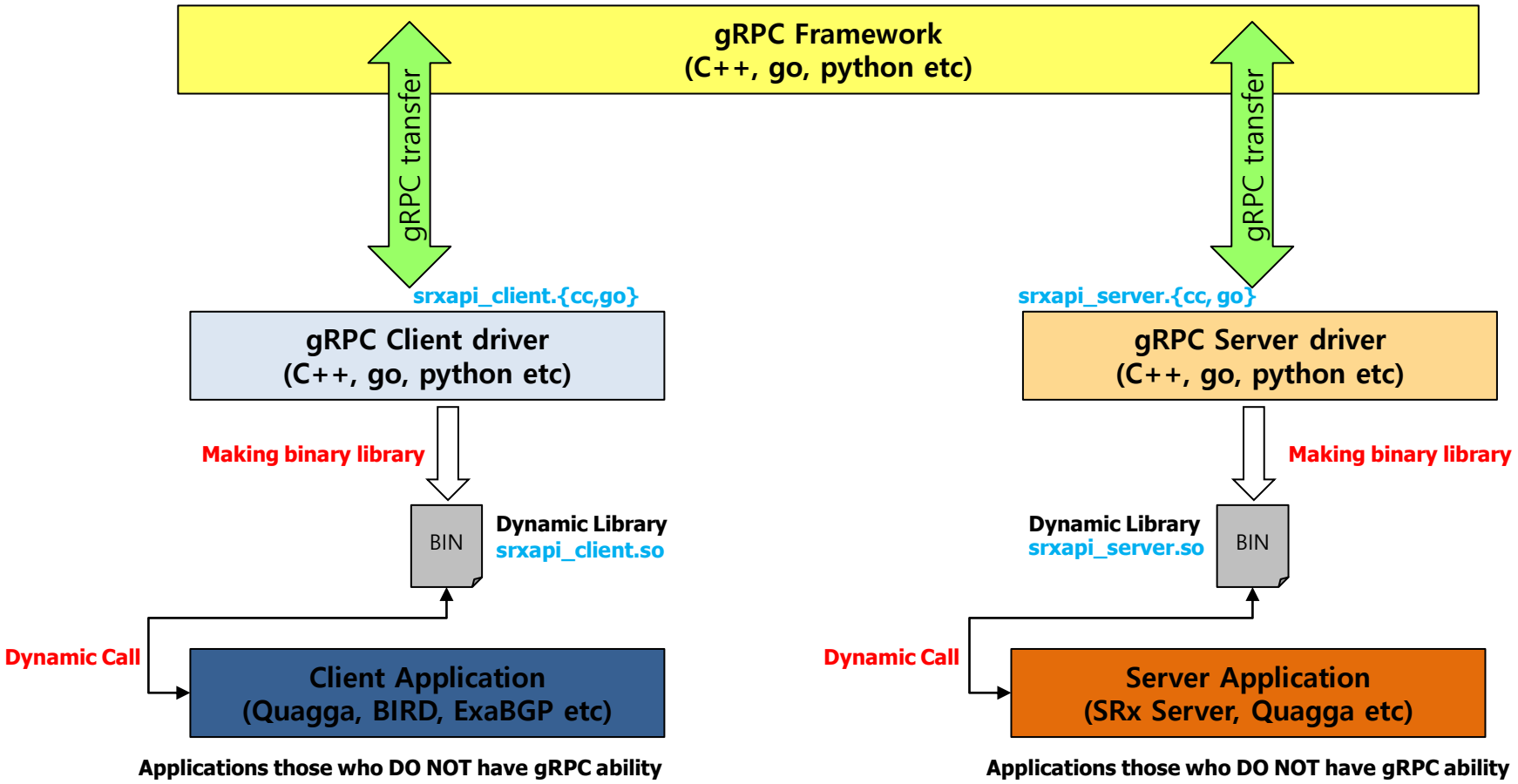
② Any gRPC support language (Go, C++, Java, python …) platform, provided by plugins on each language

③ Language dependent layer; MUST be same with upper layer and
Provided as a binary (library) and compiled with the lower layer's code

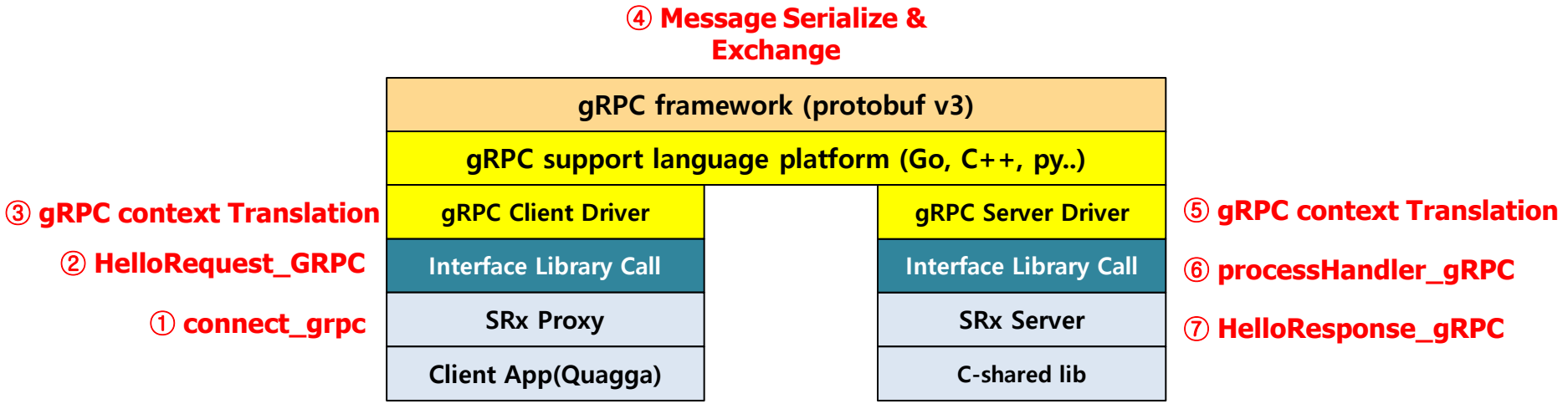
④ Library calls for connection between C-based SRxServer(or SRxProxy) and gRPC-based server(or client) driver
Also includes gRPC service functions; normaly callback functions and concurrent threads

⑤ ⑥ Regular application layer who doesn't have gRPC ability

Technical Analysis of gRPC-enabled Architecture for SRx Suite



gRPC message Exchange Example (1) - HelloRequest & Response



gRPC message Exchange Example (2) – Result Analysis (HelloRequest & Response)

[gRPC Client] –> [SRx Server]

SRx Server received gRPC Request of Hello Request from a grpc client

```
[SRxCryptoAPI - INFO] The internal key initialized storage holds (2 private and 6 public keys)!
WARNING [03/20/19 11:35.54] The loaded srx-crypto-api plug-in does not support remote control of LOGING configuration - Use API configuration to do
+ pthread grpc service started...
+ grpcServiceHandler : 0x63a160
+ grpcServiceHandler.CommandQueue   : 0x637380
+ grpcServiceHandler.CommandHandler : 0x637420
+ grpcServiceHandler.UpdateCache    : 0x636fe0
+ grpcServiceHandler.svrConnHandler : 0x637460
+ pthread grpc service thread stopped and going to joinable status
++ pthread grpc service thread started...
 setLogMode testing : input param: 3
server: * &srx_grpc.PduRequest{Data:[]uint8{0x0, 0x0, 0x2, 0x0, 0x0, 0x0, 0x0, 0x14, 0x0, 0x0, 0x0, 0x5, 0x0, 0x0, 0xfd, 0xed, 0x0, 0x0, 0x0, 0x3},
XX_sizecache:03
```

Requested ProxyID : 0x05

[SRx Server] → [gRPC client]

Successfully registered the client and Return a HelloResponse

```
[processHandshake_grpc] function called
++ grpcServiceHandler : 0x63a160
++ grpcServiceHandler.CommandQueue   : 0x637380
++ grpcServiceHandler.CommandHandler : 0x637420
++ grpcServiceHandler.UpdateCache    : 0x636fe0
++ grpcServiceHandler.svrConnHandler : 0x637460
   INFO [03/20/19 11:36.03] Register proxyID[0x00000005] as clientID[0x00000002]
   INFO [03/20/19 11:36.03] Handshake: Connection to proxy[0x00000005] accepted. Proxy registered as internal client[0x02]
return size: 12          data: []byte{0x1, 0x0, 0x2, 0x0, 0x0, 0x0, 0x0, 0xc, 0x0, 0x0, 0x0, 0x53}
```

[SRx Server]'s CLI

```
Welcome to BGP-SRx!
=====================================
[SRx]> show-proxies
Display proxy mappings:
======================
* Client[0x01](  1): Proxy ID 000.000.000.002 [0x00000002](0000000002) (-------/pre-conf) - #updates=0
* Client[0x02](  2): Proxy ID 000.000.000.005 [0x00000005](0000000005) ( active/dynamic ) - #updates=0
* Client[0x0A]( 10): Proxy ID 010.000.000.001 [0x0A000001](0167772161) (-------/pre-conf) - #updates=0
* Client[0x19]( 25): Proxy ID 010.001.001.002 [0x0A010102](0167837954) (-------/pre-conf) - #updates=0
```

gRPC message Exchange Example (3) – Result Analysis (VerifyRequest & Response)

[gRPC Client] verify update sent & Received verifyResponse with Default result

```
[grpc_client] verify update sent
input data for stream response: []byte{0x3, 0x83, 0x1, 0x1, 0x0, 0x0, 0x0, 0xa9, 0x3, 0x3, 0x0, 0x18, 0x0, 0x0, 0x0, 0x1,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xfd, 0xed, 0x0, 0x0, 0xfd, 0xf
 0x19, 0x75, 0xff, 0x19, 0x31, 0x81, 0x45, 0x8f, 0xb9, 0x2, 0xb5, 0x1, 0xea, 0x97, 0x89, 0xdc, 0x0, 0x48, 0x30, 0x46, 0x2
3, 0x8d, 0x64, 0x5f, 0xa0, 0xb7, 0x20, 0x7e, 0xf3, 0x2c, 0xcc, 0x4b, 0x3f, 0xd6, 0x1b, 0x5f, 0x46, 0x2, 0x21, 0x0, 0xb6,
 0x47, 0x60, 0x25, 0xe0, 0x8c, 0xda, 0x49, 0xf9, 0x1e, 0x22, 0xd8, 0xc0, 0x8e3
data : []byte{0x6, 0x83, 0x3, 0x3, 0x0, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0, 0x1, 0x20, 0x8d, 0xfc, 0x623
```

Received verifyResponse from SRxServer with UpdateID (0x20, 0x8d, 0xfc, 0x62)

[SRx Server] send immediately with Default Result (Undefined: 0x03, 0x03)

```
[processValidationRequest_grpc] function called
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] GRPC_ServiceHandler): Enter processValidationRequest
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] UpdateCache): Store update [ID:0x208DFC62] in update cache.
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Mutex): --> [0x00636FF0] REQ LOCK
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Mutex): <-- [0x00636FF0] LOCKED
  DEBUG [03/20/19 11:36.03] ski_register 546176098
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Mutex): ==> [0x00636FF0] UNLOCK
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Mutex): <== [0x00636FF0] UNLOCKED
+ from updata cache srxRes.roaResult : 03
+ from updata cache srxRes.bgpsecResult : 03
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Command Queue): queueComamnd type (0)
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Mutex): --> [0x006373A0] REQ LOCK
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Mutex): <-- [0x006373A0] LOCKED
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Command Queue): Signale new data to consume...queueCommand
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Condition signal): --> to [0x006373C8]
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Command Queue): UNLOCK readWriteLock...queueCommand
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Mutex): ==> [0x006373A0] UNLOCK
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] Mutex): <== [0x006373A0] UNLOCKED
  DEBUG [03/20/19 11:36.03] ([0xFE1F6700] GRPC_ServiceHandler): Exit processValidationRequest
return size: 16          data: []byte{0x6, 0x83, 0x3, 0x3, 0x0, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0, 0x1, 0x20, 0x8d, 0xfc, 0x623
```

[SRx Server] Received verify Request

```
[processHandshake_grpc] function called
++ grpcServiceHandler : 0x63a160
++ grpcServiceHandler.CommandQueue   : 0x637380
++ grpcServiceHandler.CommandHandler : 0x637420
++ grpcServiceHandler.UpdateCache    : 0x636fe0
++ grpcServiceHandler.svrConnHandler : 0x637460
   INFO [03/20/19 11:36.03] Register proxyID[0x00000005] as clientID[0x00000002]
   INFO [03/20/19 11:36.03] Handshake: Connection to proxy[0x00000005] accepted. Proxy registered as internal client[0x02]
return size: 12          data: []byte{0x1, 0x0, 0x2, 0x0, 0x0, 0x0, 0x0, 0xc, 0x0, 0x0, 0x0, 0x5}
 setLogMode testing : input param: 3
stream server: ◆◆d◆qm◆◆◆a3◆u◆1◆E◆◆◆8H0F!◆◆◆i5n{l◆◆<◆Jc◆d_◆◆ ^?_F!◆
!◆P[▯c=◆s-^G^XI◆"◆ &srx_grpc.PduRequest{Data:[]uint8{0x3, 0x83, 0x1, 0x1, 0x0, 0x0, 0x0, 0xa9, 0x3, 0x3, 0x0, 0x18, 0x0, 0x
x1, 0x1, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xfd, 0xed, 0x0, 0x
4, 0x33, 0xfa, 0x19, 0x75, 0xff, 0x19, 0x31, 0x81, 0x45, 0x8f, 0xb9, 0x2, 0xb5, 0x1, 0xea, 0x97, 0x89, 0xdc, 0x0, 0x48, 0x3
x1c, 0x4a, 0x63, 0x8d, 0x64, 0x5f, 0xa0, 0xb7, 0x20, 0x7e, 0xf3, 0x2c, 0xcc, 0x4b, 0x3f, 0xd6, 0x1b, 0x5f, 0x46, 0x2, 0x21,
3, 0x2d, 0x27, 0x47, 0x60, 0x25, 0xe0, 0x8c, 0xda, 0x49, 0xf9, 0x1e, 0x22, 0xd8, 0xc0, 0x8e3, Length:0xa9, XXX_NoUnkeyedLit
 setLogMode testing : input param: 7
calling SRxServer responseGRPC()
+ [SRX][responseGRPC] calling - size: 169
ret bool: 1
```

```
validation function called
bgpsec_validation data parameter parsing...
myAS:0xedfd0000
bgpsec_path_attr:

90 21 00 69 00 08 01 00 00 00 fd f3 00 61 01 c3
04 33 fa 19 75 ff 19 31 81 45 8f b9 02 b5 01 ea
97 89 dc 00 48 30 46 02 21 00 bd 92 9e 69 35 6e
7b 6c fe 1c bc 3c bd 1c 4a 63 8d 64 5f a0 b7 20
7e f3 2c cc 4b 3f d6 1b 5f 46 02 21 00 b6 0a 7c
82 7f 50 e6 5a 5b d7 8c d1 81 3d bc ca a8 2d 27
47 60 25 e0 8c da 49 f9 1e 22 d8 c0 8e 00 00 00
nlri: 0x7fa7e0000a44
nlri.afi: 100
nlri.safi: 1
nlri.length: 24
nlri.addr.ipV4: 164
hash_message[0]: (nil)
hash_message[1]: (nil)
+++++ ks_getKey called
+++++ storage: 0x1214a20, ski:c34 asn:f3fd0000

Hash(validate):
00 00 fd ed 01 00 00 00 fd f3 01 00 01 01 18 64
01 00

Digest(validate):
f1 49 6e 6c 56 2f 25 48 5f 9f 1b 46 64 ae e4 d0
13 ee fa 6d c9 00 50 0f b9 37 b8 a9 bb 50 cf 3a
[SRxCryptoAPI - DEBUG] stack[1] VERIFY SUCCESS
  DEBUG [03/20/19 11:36.03] [PrefixCache [0xF5FFB700]]: Store new ROA result[0x00] for update [0x208DFC62]
```

[SRx Server] ROA, BGPSec Path Validation
for UpdateID (0x20, 0x8d, 0xfc, 0x62) which was received and stored

As a result,
ROA validation : VALID
BGPSec Path: VALID

[SRx Server] Sending Verify Notify with the validated Result to the gRPC client

```
proxy callback function : arg[16, (unsafe.Pointer)(0x7fa7e4000970)]
My callback function - received arg: 16, []byte{0x6, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0, 0x0, 0x20, 0x8d, 0xfc, 0x623
2019/03/20 23:36:03 sending stream data
```

[gRPC Client] Received Verify Notify Result with the validated data from SRx Server

```
data : []byte{0x6, 0x2, 0x0, 0x0, 0x0, 0x0, 0x0, 0x10, 0x0, 0x0, 0x0, 0x0, 0x20, 0x8d, 0xfc, 0x623
size : 0x10
status: 0x2
2019/03/20 23:36:08 Finished with Resopnse valie: 2
2019/03/20 23:36:08 context canceled
 Validation Result: 02
Press return ...
```

0x0, 0x0 (ROA, BGPSecPath), 0: means VALID

[SRx Server's CLI] check Update Cache and Prefix Cache inside SRx Server

```
<?xml version="1.0" encoding="iso-8859-1"?>
<prefix-cache>
  <prefix ip="100.1.0.0" length="24" roa-coverage="1" no-valid-updates="1" no-other-updates="0" state-of-other="
INVALID">
    <as as-number="65011" update-count="1">
      <roa valCacheID="1" as="65011" max-length="24" roa-count="1" deferred-count="0" update-count="1" />
    </as>
    <valid no-updates="1">
      <update update-id="0x208DFC62" as="65011" roa-match="1" />
    </valid>
    <other no-updates="0" state="INVALID" />
  </prefix>
  <updates>
    <update update-id="0x208DFC62" origin-as="65011" prefix="100.1.0.0/24" roa-count="1" val-state="VALID" />
  </updates>
</prefix-cache>
```

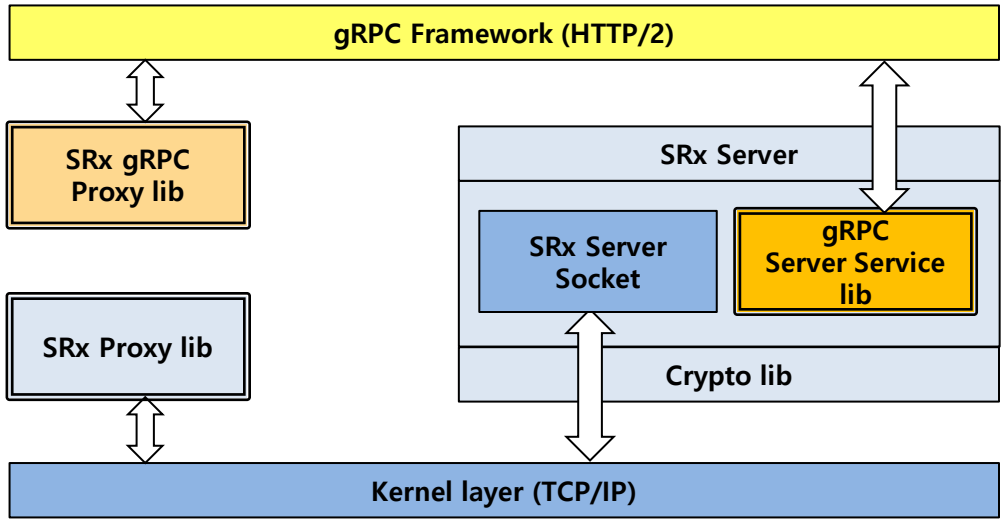
```
  DEBUG [03/20/19 11:42.28] [console(10)] Process command dump-ucache
<?xml version="1.0" encoding="iso-8859-1"?>
<update-cache current-gc-time="2084">
  <updates>
    <update update-id="0x208DFC62" no-clients="1" client-list="100" gc="0" origin-as="65011" prefix="100.1.0.0/2
4" roa-count="0" origin-val="VALID!" path-val="VALID!" def-origin-val="UNDEFINED!" def-path-val="UNDEFINED!" hop
s="1" bgpsec-len="109" />
  </updates>
```

Work TODO

- **gRPC Client side**
  - SRx Proxy
    - Code organizing and optimizing
  - gRPC Client driver
    - Bug Correction
- **gRPC Server side**
  - SRx Server
    - Enhance concurrency task for streaming gRPC api
    - Add remaining functions of proxy call
      - DeleteUpdate, PeerChange and so on
  - gRPC Server driver
- **gRPC service method for each functions**
  - Added method into IDL and protoc compile
  - Effectiveness of gRPC's serialization / deserialization
- **Robustness Testing**
- **Packaging**

SRx gRPC Support Extensions (1)

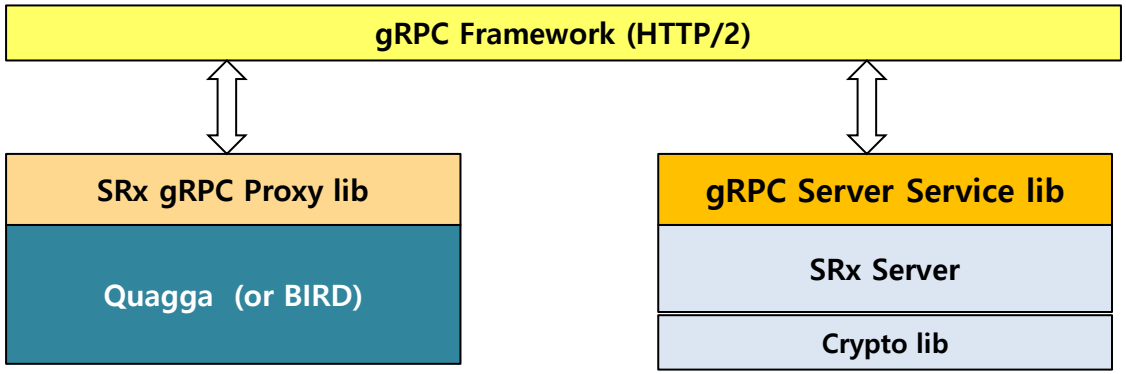
- **SRx Proxy consists of 2 parts**
  - SRx Proxy library (TCP/IP)
  - SRx gRPC Proxy library (gRPC)



SRx gRPC Support Extensions (2)

◆ Case 1 :

- BGP routing implementation which doesn't support gRPC mechanism

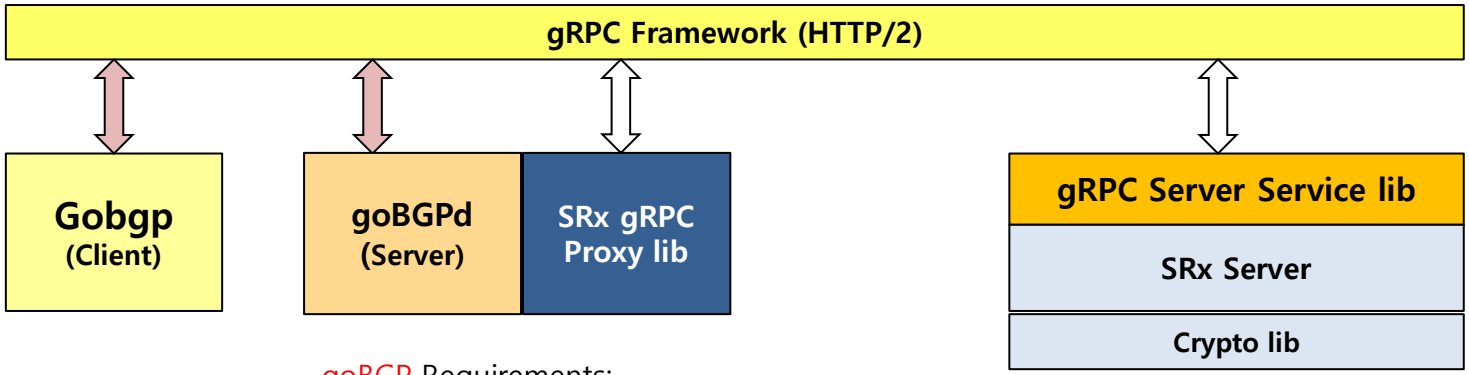


Quagga (BIRD) Requirements:
- **go language**
- **go-gRPC (protocol buffer)**
- **Dynamic call gRPC proxy lib**

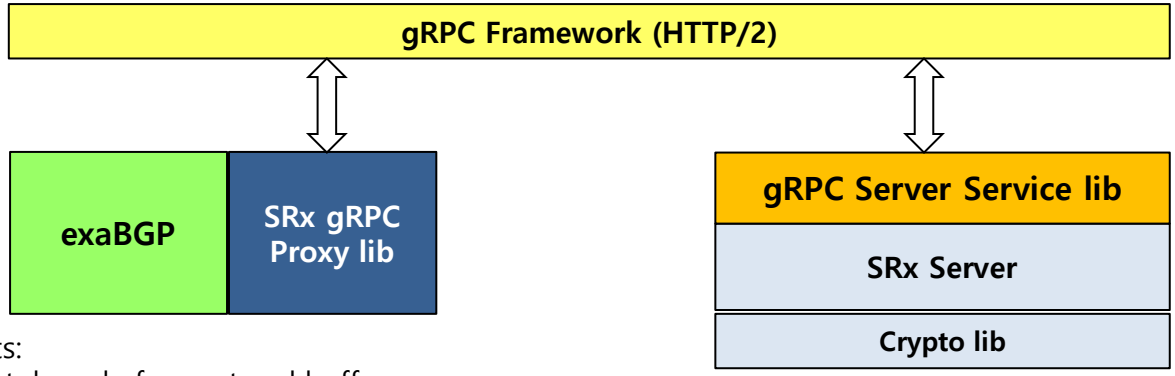
SRx gRPC Support Extensions (3)

Case 2

- goBGP and exaBGP which is able to support gRPC by its language



goBGP Requirements:
- import SRx gRPC Proxy library
- Dynamic call from goBGPd server

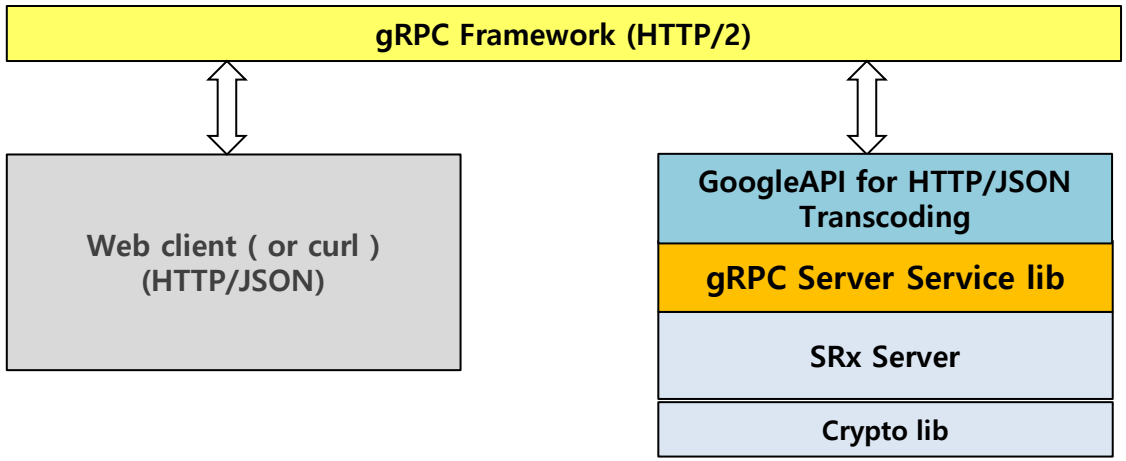


exaBGP Requirements:
- Generate python stub code for protocol buffer
- Import SRx gRPC Proxy lib
- Dynamic call from the python stub code

SRx gRPC Support Extensions (4)

Case 3

- Web client

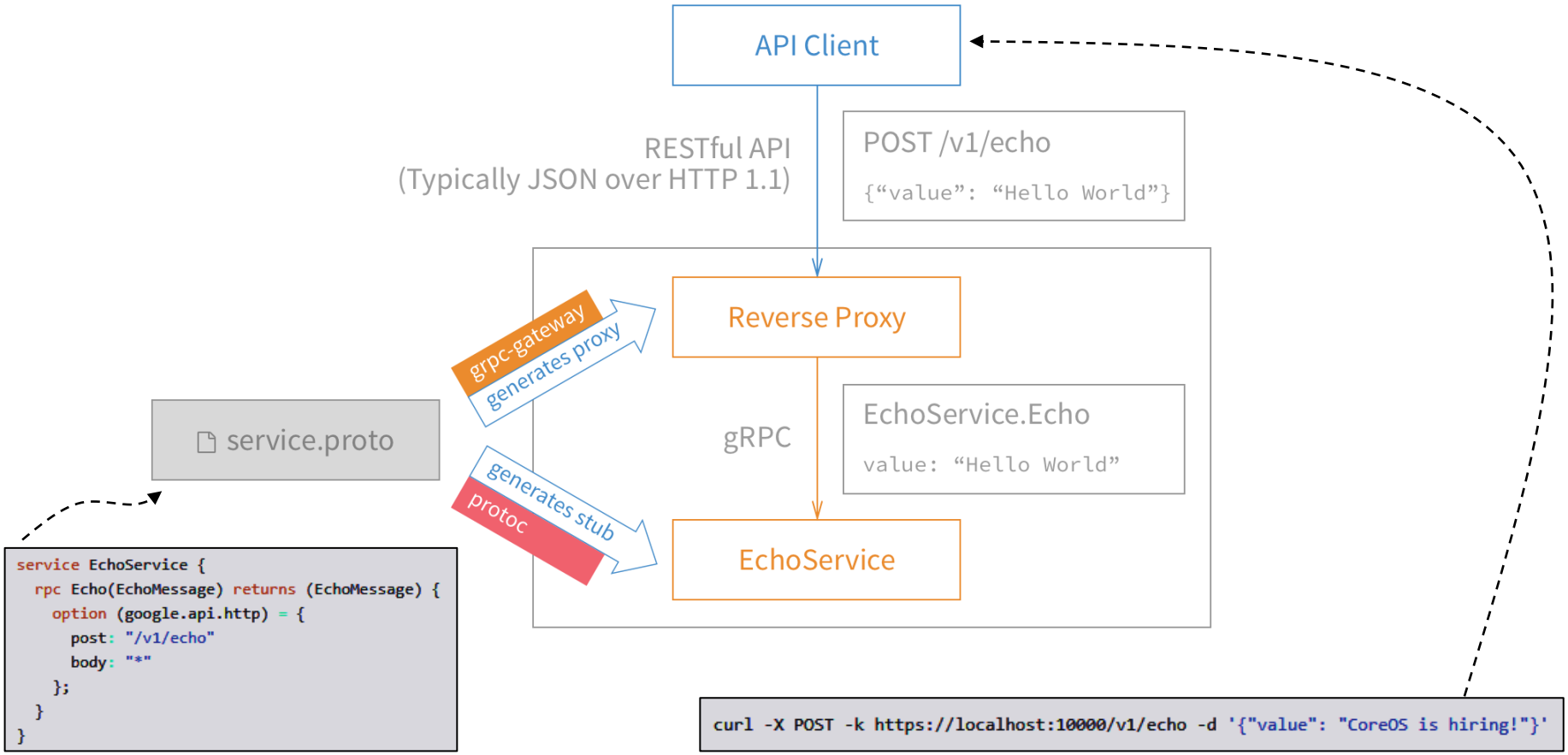


gRPC transcoding for SRxServer

- Using HTTP Client, will be able to support RESTful service with CRUD methods(Create, Read, Update, and Delete)
- For Example,
  - Connect SRxServer → POST with a new URI
  - Get ROA, BGPSec Validation→ GET, UPDATE the object
  - Error handling, Goodbye → DELETE

Transcoding HTTP/JSON to gRPC

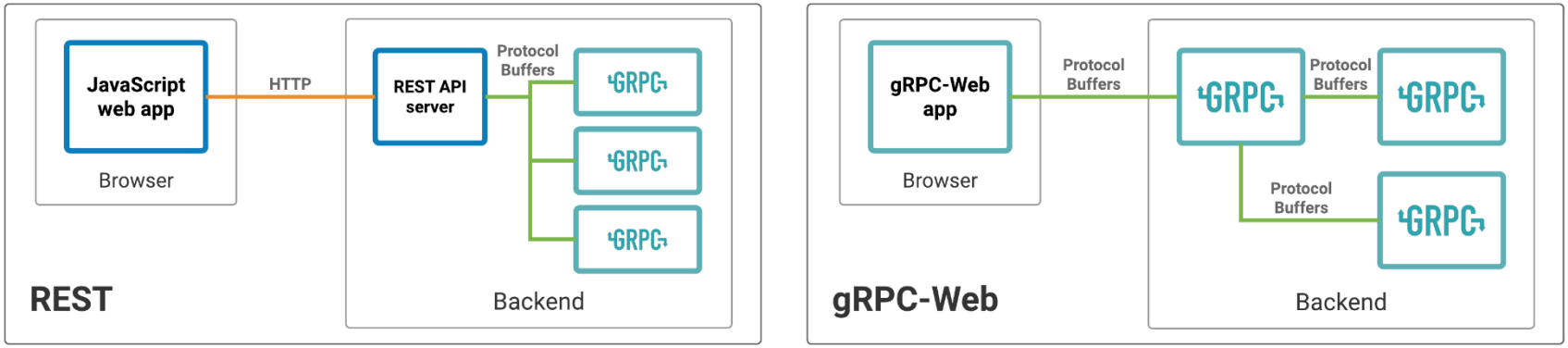
- The gRPC API configuration standard
  - how data should be translated from HTTP/JSON to gRPC – using GoogleAPIs for grpc
    - https://cloud.google.com/endpoints/docs/grpc/transcoding
    - https://www.grpc.io/blog/coreos/



SRx gRPC Support Extensions (5)

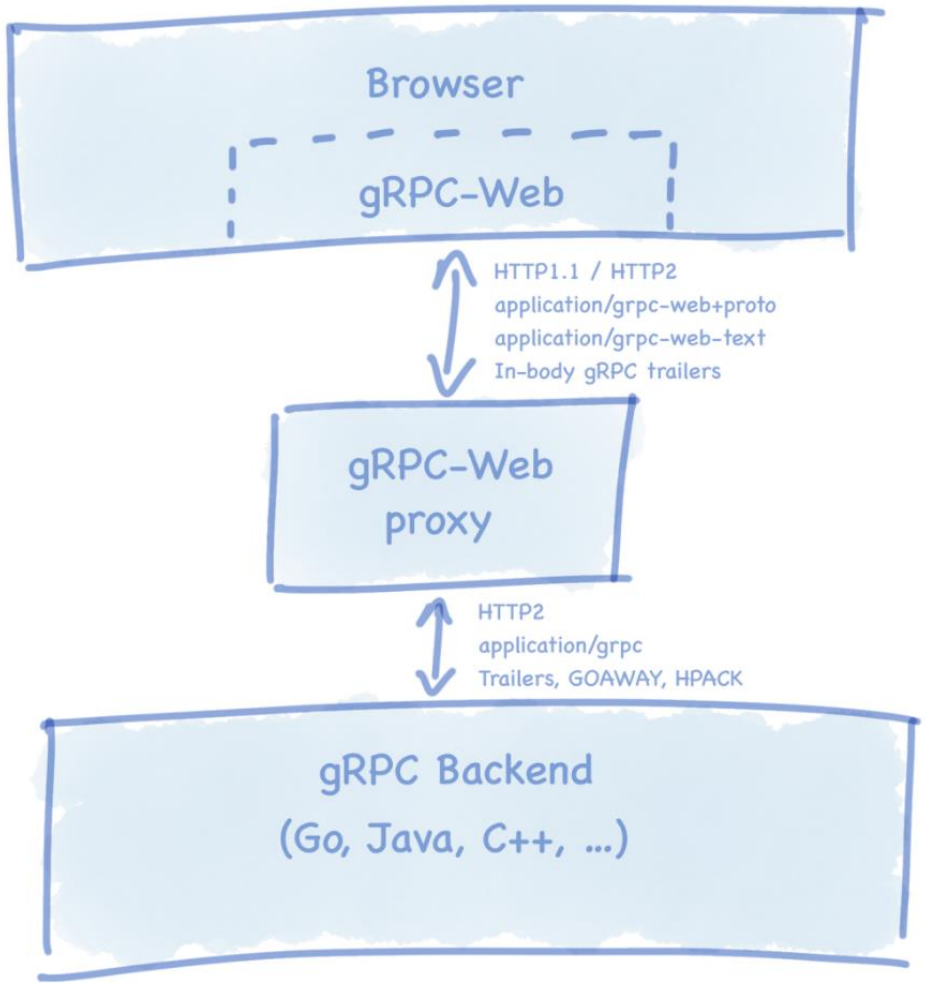
◆ gRCP-Web

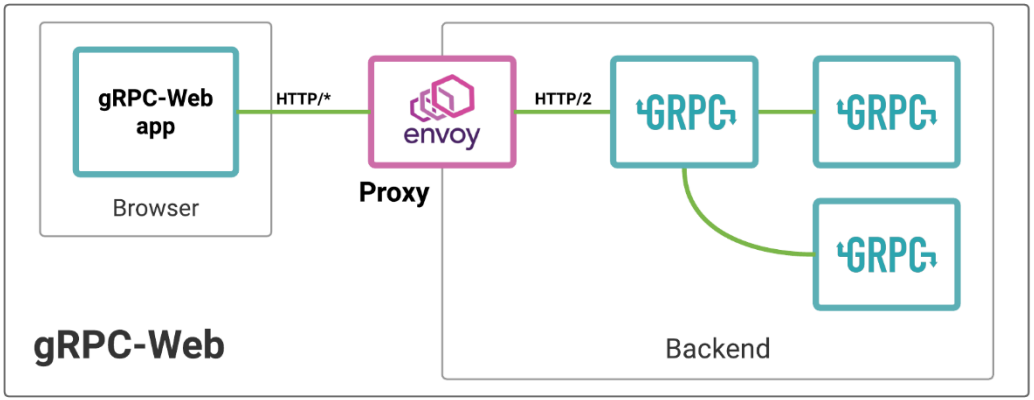
- Client-backend interaction in a REST API vs. gRPC-Web



- Links
  - https://grpc.io/docs/tutorials/basic/web/
  - gRPC-Web is going GA - Cloud Native Computing Foundation
  - Microsoft brings gRPC-Web support to .NET

Into the Tech Details





The role of Envoy in a gRPC-Web application

Envoy translates the HTTP/1.1 calls produced by the client into HTTP/2 calls that can be handled by those services (gRPC uses HTTP/2 for transport)

Installation SRx Suite with gRPC support

- **SRx Server and Proxy for gRPC enabled**
  - Download somewhere from the repository
    - NIST official github ? Or Independent repo (yum repo or else)
  - Configure with "--enable-grpc" switch

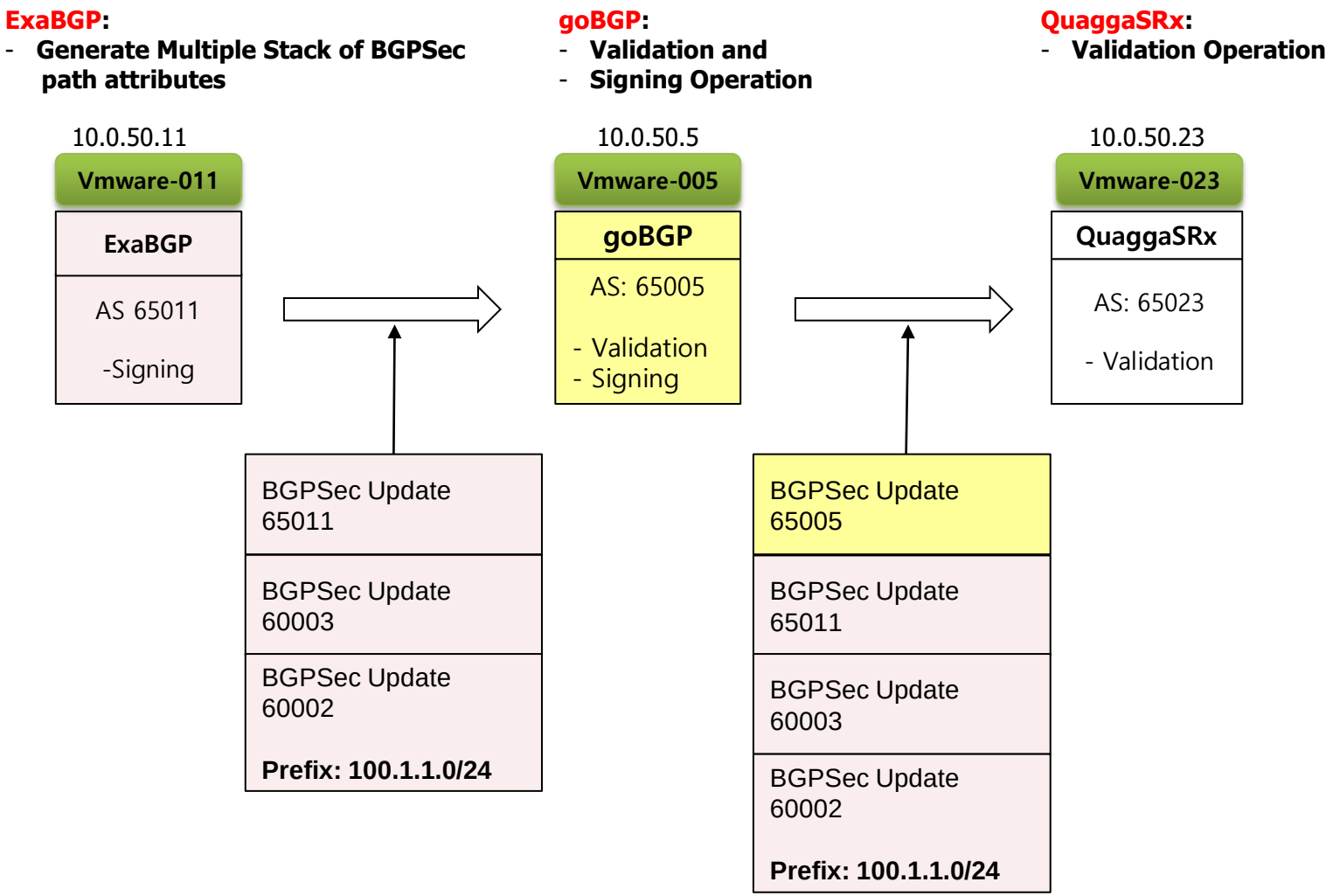
- **Go language**
  - Download the archive(https://golang.org/dl/)
  - and extract it into /usr/local, creating a Go tree in /usr/local/go

- **gRPC and Protocol buffer**
  - download from https://github.com/protocolbuffers/protobuf/releases
  - simply unzip and place the binary(protoc) somewhere in PATH

- **Go protocol buffer plugin**
  - Hit the command below
    - go get -u google.golang.org/grpc
    - go get -u github.com/golang/protobuf/protoc-gen-go

BGPSec InterOperations

- ExaBGP, goBGP and QuaggaSRx



```
vmware-023> sh ip bgp
BGP table version is 0, local router ID is 10.0.50.23
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, R Removed
Validation:    v - valid, n - notfound, i - invalid, ? - undefined
SRx Status:    I - route ignored, D - SRx evaluation deactivated
SRxVal Format: validation result (origin validation, path validation)
Origin codes: i - IGP, e - EGP, ? - incomplete

   Ident    SRxVal SRxLP Status Network          Next Hop            Metric  LocPrf Weight Path
*> 309D9E76 v(v,v)              100.1.1.0/24     10.0.50.5                                0 65005 65011 60003 60002 i

Total number of prefixes 1
vmware-023>
```

Operational Conditions

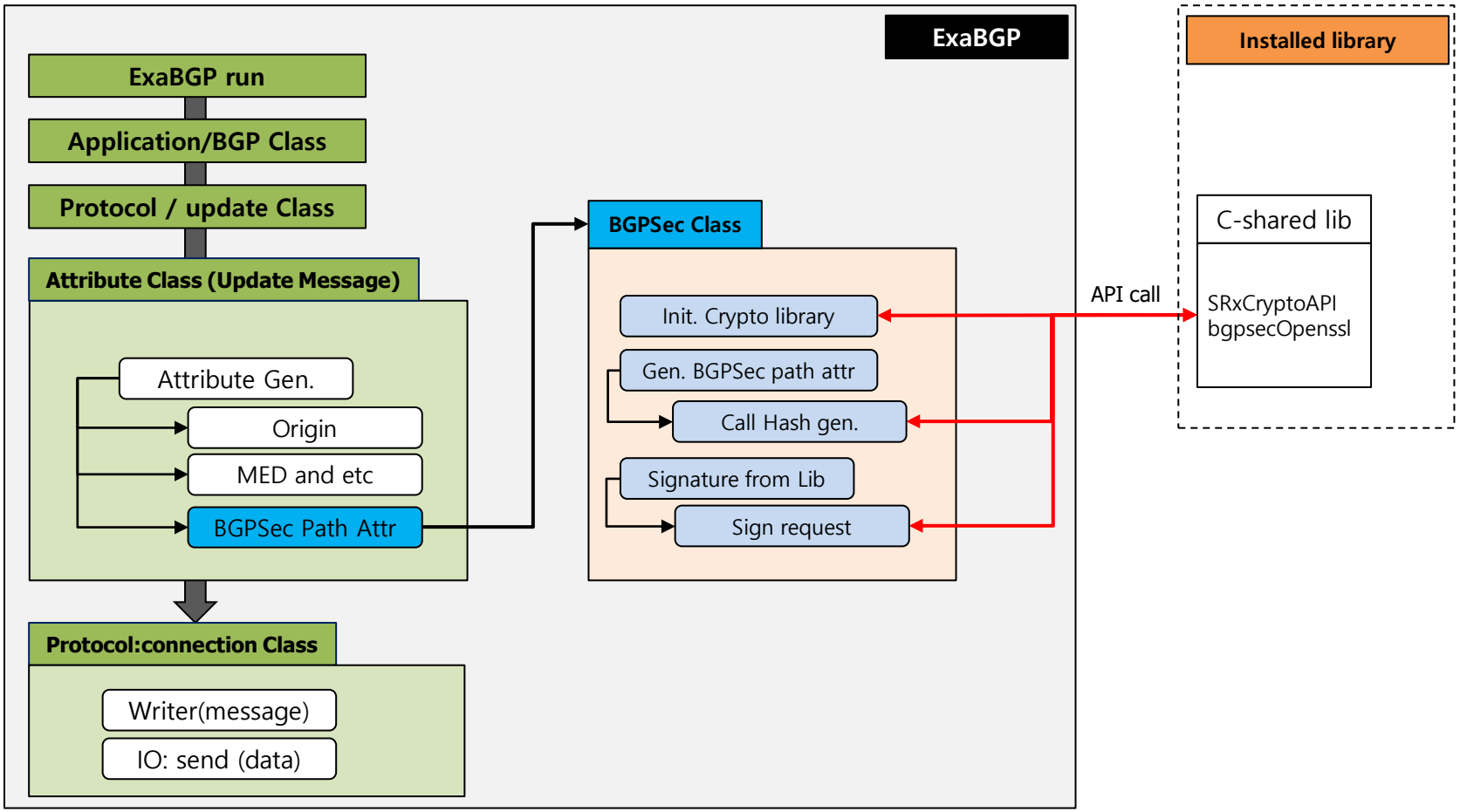
ExaBGP

- Requirements
  - Python version > v2.7
  - SRxCryptoAPI installation
- Configuration
  - route update information
    - Provided by various way, direct input from the config file, or standard input, script program etc
  - BGPSec options
    - import SRxCrypto library,
    - AS path and SKI pairs to generate BGPSec Path attributes
- Running
  - No need to install or Compile, Just copy ExaBGP package to somewhere and run by python

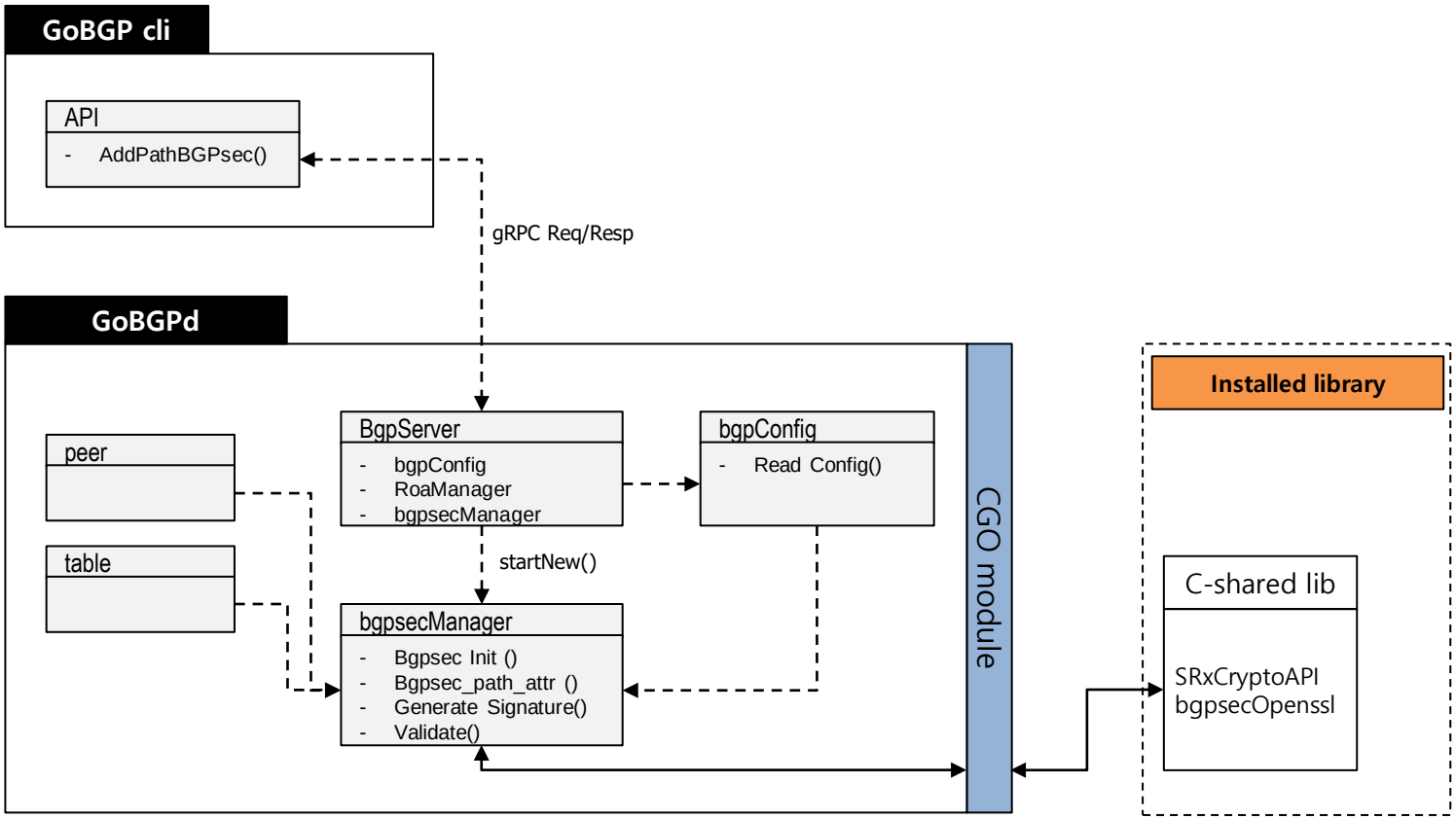
goBGP

- Requirements
  - Go version > v1.10
  - Protocol buffer compiler > v3.6
  - Protocol buffer plugins for golang
  - SRxCryptoAPI installation
- Configuration
  - Route update information
    - Uses gobgp client (communicates to gobgpd by gRPC protocol) to feed the updates
    - Can be automated by programming or using shell script to input, and so on
  - BGPSec Options
    - RPKI policy options
    - RPKI rtr server connection setting
    - SKI and SRxCrypto library LDFlag, Cflag configuration for import library and include header files
- Running
  - Building (Compiling), Installation for binary executables
  - Running with go dependent packages which is able to be imported by github at the run-time dynamically

ExaBGP with BGPSec features

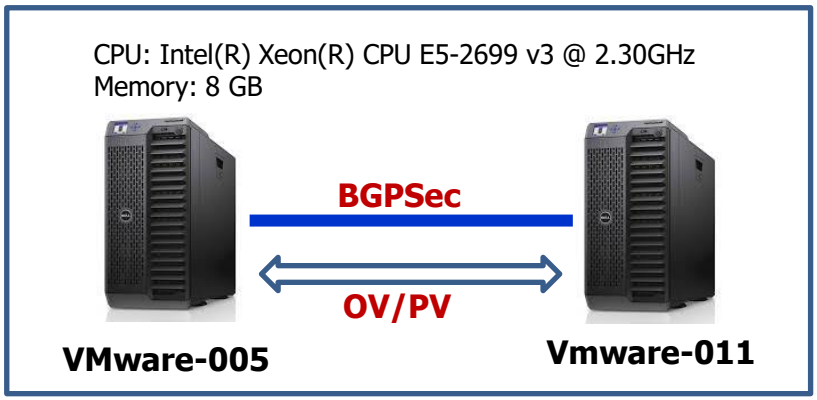


goBGP with BGPSec features



Performance Analysis

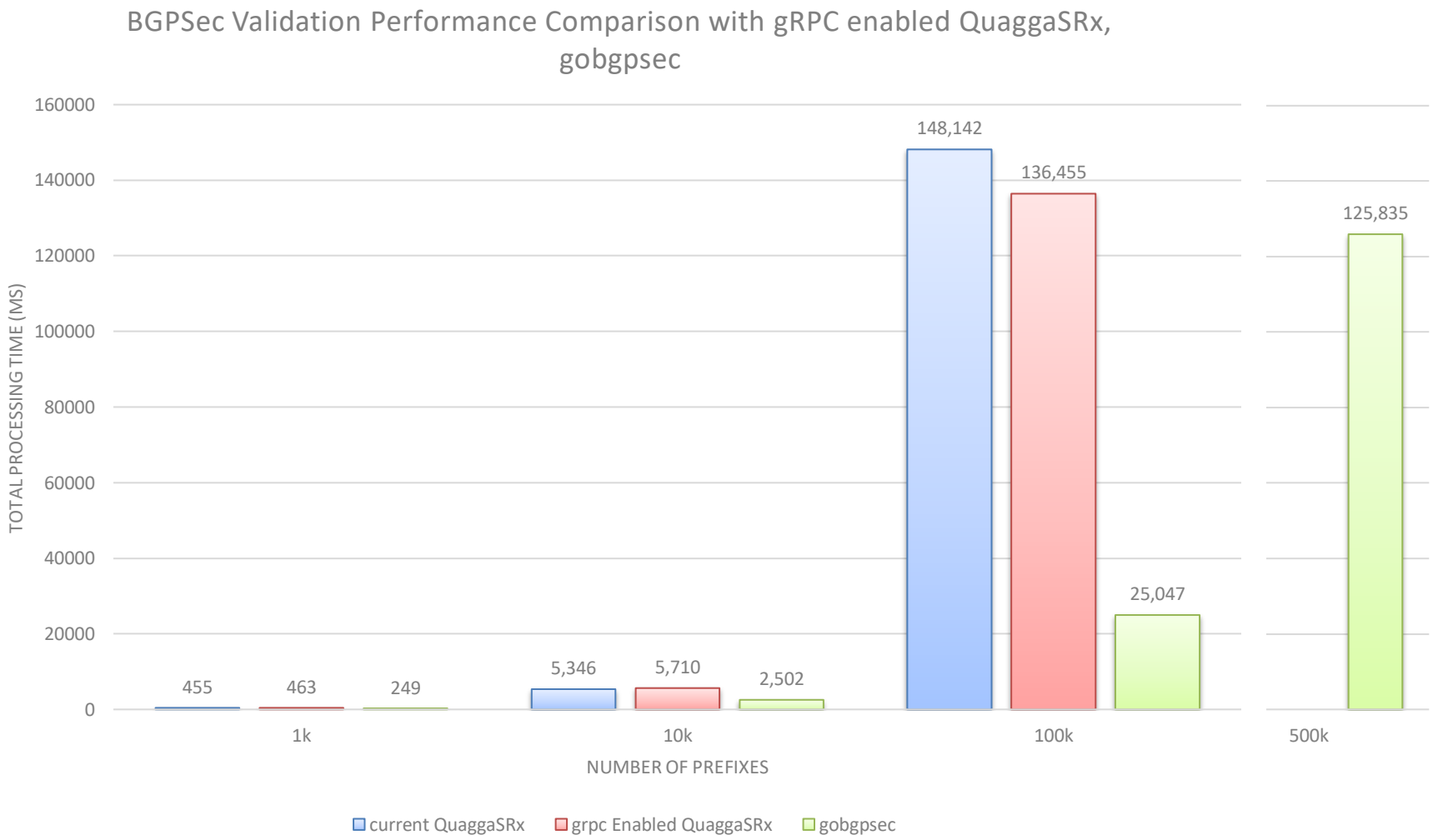
BGPSec Routers Connection Topologies

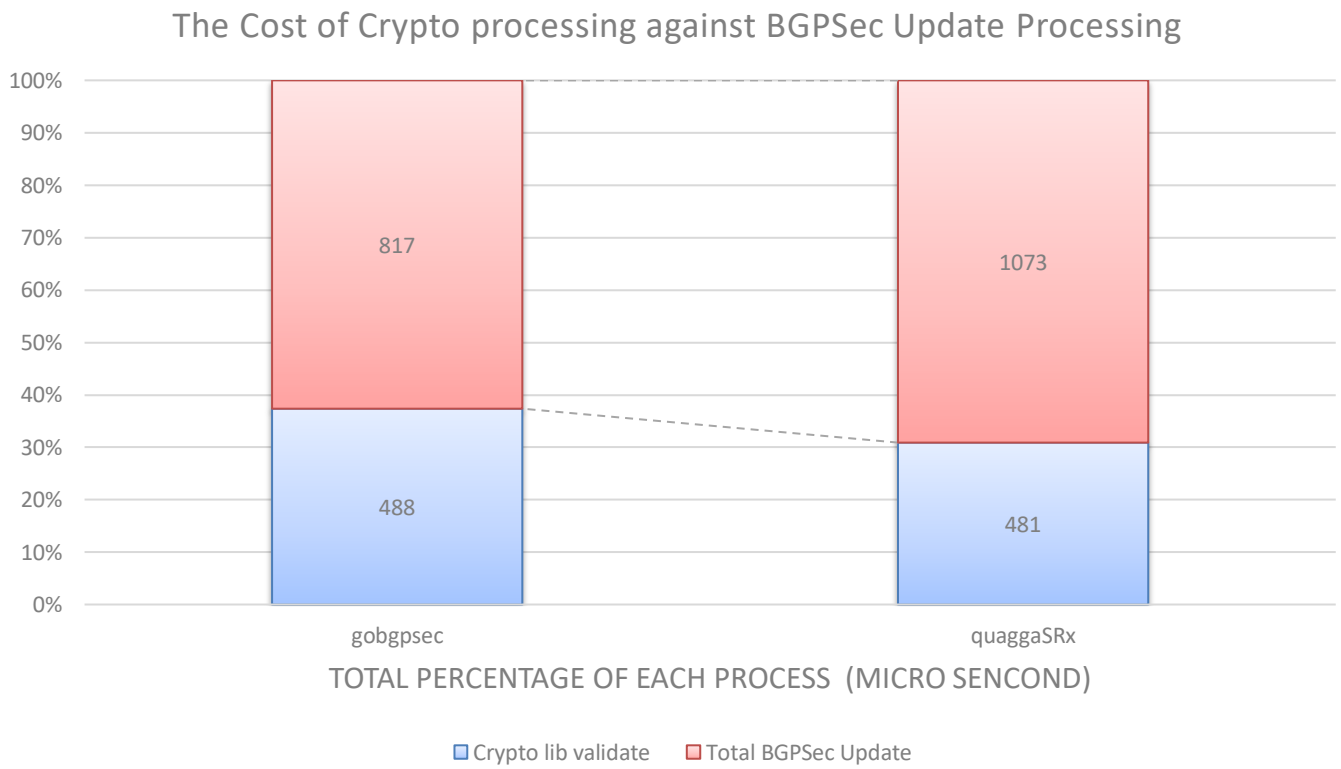


AS number	: 65005
Router-id	: 10.0.50.5
Running Target application	

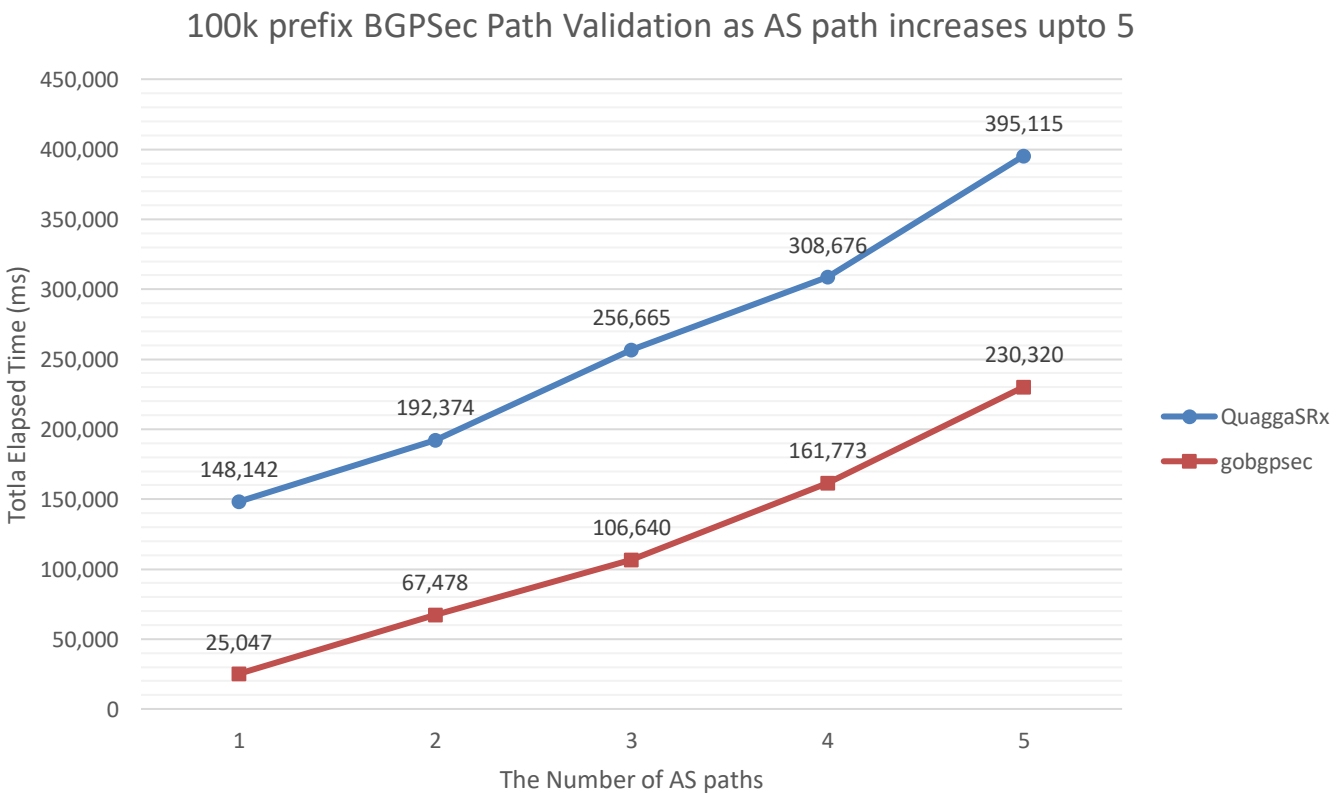
AS number	: 65011
Router-id	: 10.0.50.11
Running Traffic Gen.	

gRPC enabled QuaggaSRx performance



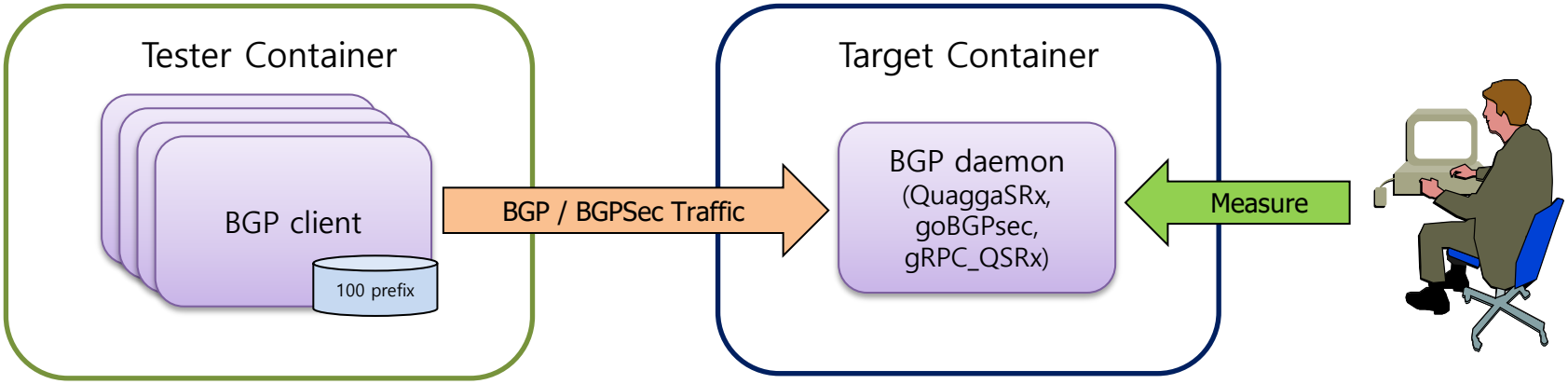


- **Single BGPSec Update Processing**
- **Comparison of Total Processing Time vs. Crypto call in each case**

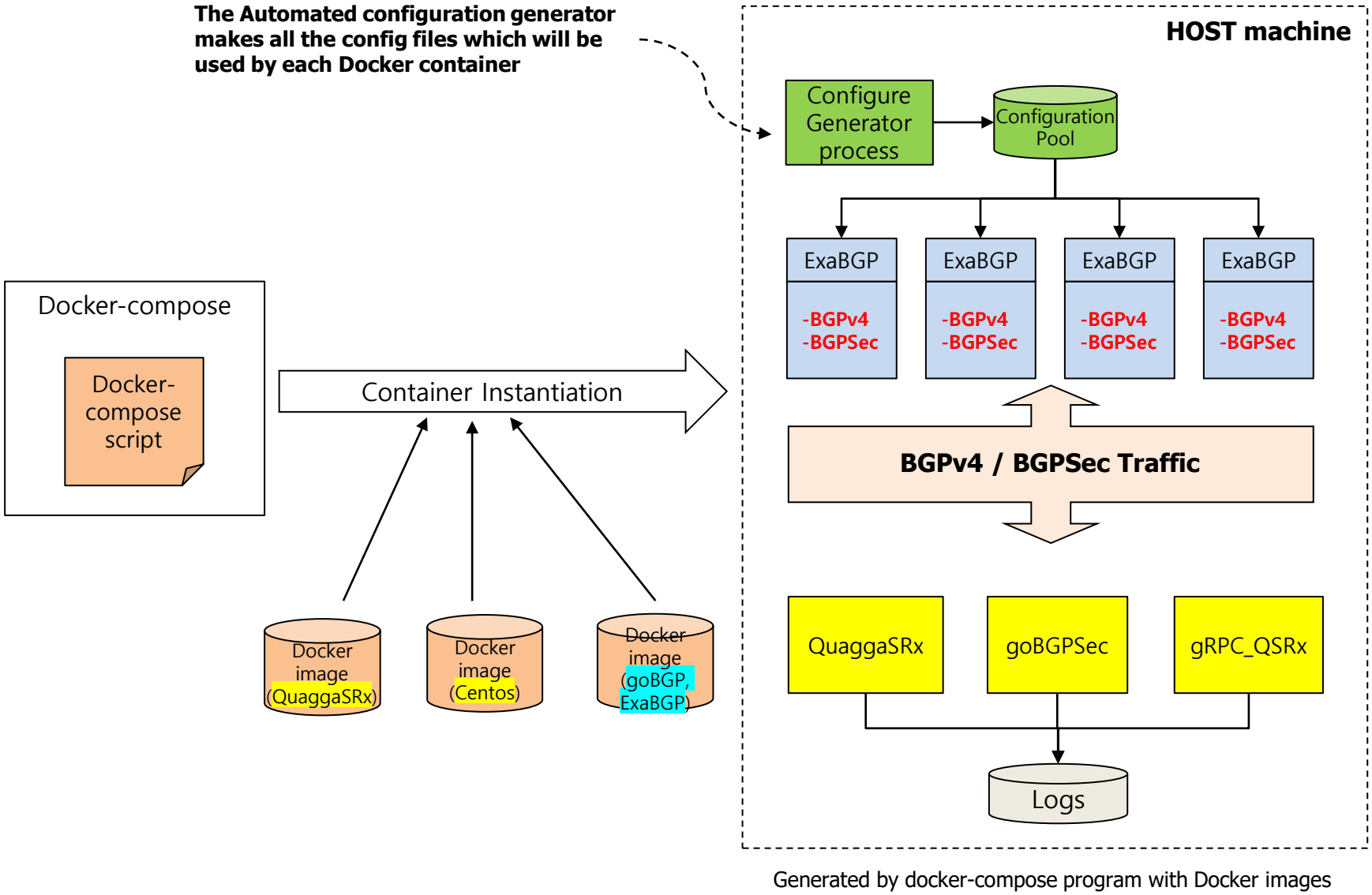


- **Traffic generated by ExaBGP**
- **BGPSec update with upto 5 As path with signatures are fed into QuaggaSRx and gobgp**

Peering Performance Test Conditions



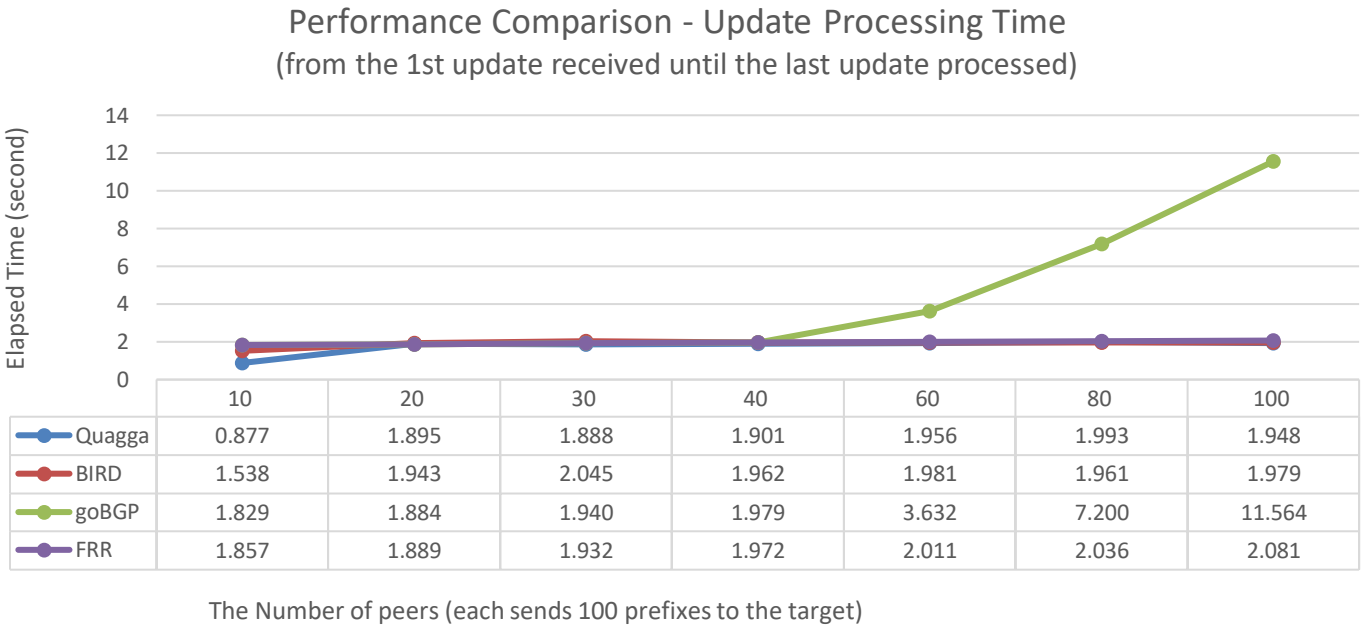
Performance Analysis Test Architecture



Generated by docker-compose program with Docker images

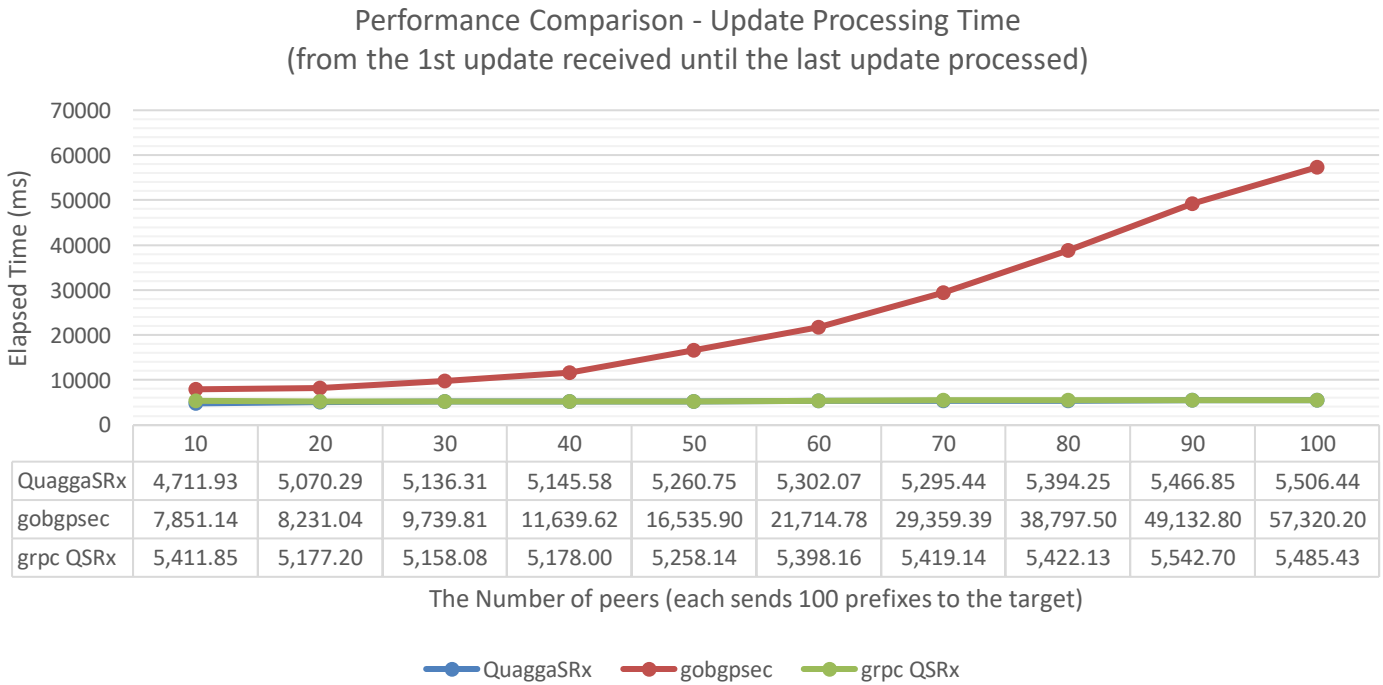
BGPv4 Comparison:

- **Update Processing**
- **Total Elapsed Time**



	10	20	30	40	60	80	100
Quagga	0.877	1.895	1.888	1.901	1.956	1.993	1.948
BIRD	1.538	1.943	2.045	1.962	1.981	1.961	1.979
goBGP	1.829	1.884	1.940	1.979	3.632	7.200	11.564
FRR	1.857	1.889	1.932	1.972	2.011	2.036	2.081

BGPSec Comparison:



	10	20	30	40	50	60	70	80	90	100
QuaggaSRx	4,711.93	5,070.29	5,136.31	5,145.58	5,260.75	5,302.07	5,295.44	5,394.25	5,466.85	5,506.44
gobgpsec	7,851.14	8,231.04	9,739.81	11,639.62	16,535.90	21,714.78	29,359.39	38,797.50	49,132.80	57,320.20
grpc QSRx	5,411.85	5,177.20	5,158.08	5,178.00	5,258.14	5,398.16	5,419.14	5,422.13	5,542.70	5,485.43

TODOs and in-progress milestones

Scalable Test with Containers and Performance Analysis

- Use up to 500,000 prefixes
- ExaBGP and goBGP utilization and consumption of cpu, memory, etc
- Performance Evaluation of processing BGPSec

Packaging for distribution

- Pre-requisite
  - Need to successfully run with a fresh new install (Done)
- Distribution Considerations
  - (1) Source code-base
    - Git-hub (NIST) or public
  - (2) Completion version
    - Docker base packaging (SRx Suite were already done, ready to use)
    - NIST has docker repository ? Otherwise need to use public repository for dockerizing
  - (3) Need to update with the latest version ?

Documentation

- NIST SP and technical note for ExaBGP, goBGP, gRPC enabled SRxServer