

# Google gNOI Operation Demo – Using gNOI capabilities to simplify software upgrade use case

Saju Salahudeen

Principal Consulting Engineer, NOKIA

Member – NANOG Education Committee

# Drivers for a Network Software Upgrade



End  
Users



Network Planning,  
Architecture, Sales



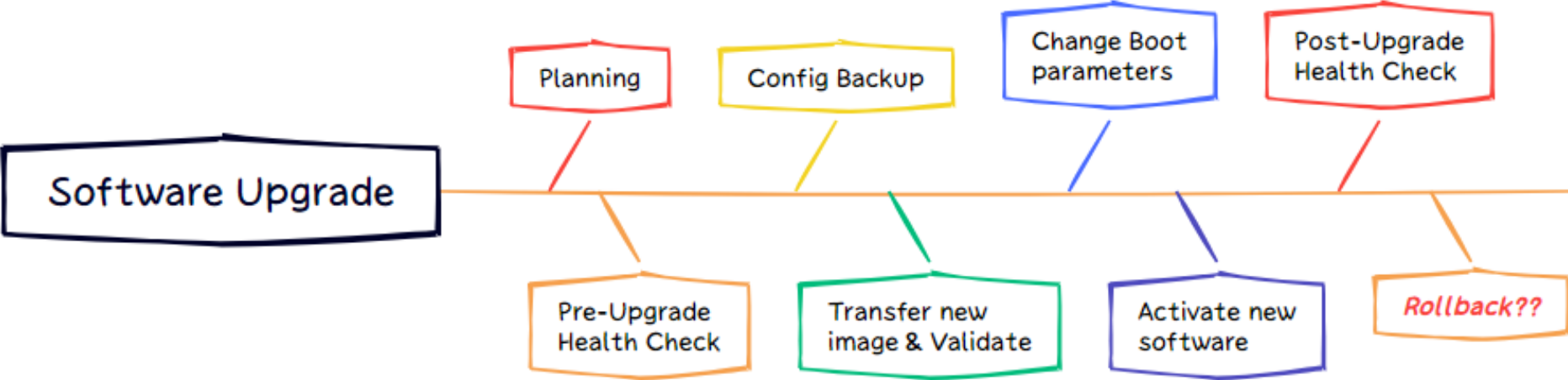
Network  
Operations

# Operations – Checklist for Upgrade

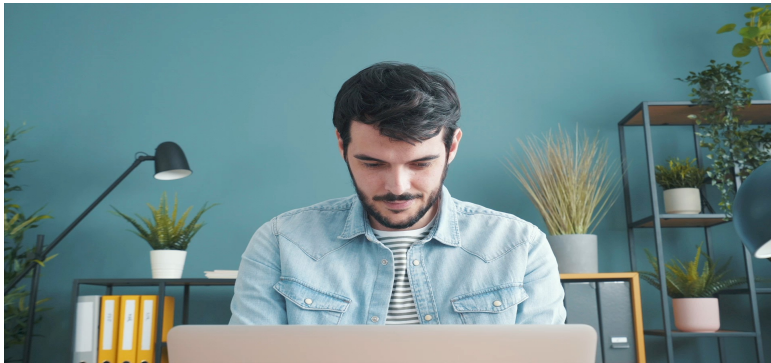
- Testing new software
- Testing external interfaces
- Testing Scripts
- Application/NMS/OSS compatibility
- Developing MOP
- Spares
- MW planning
- Notifying End Users
- Vendor Support



# Software Upgrade MOP



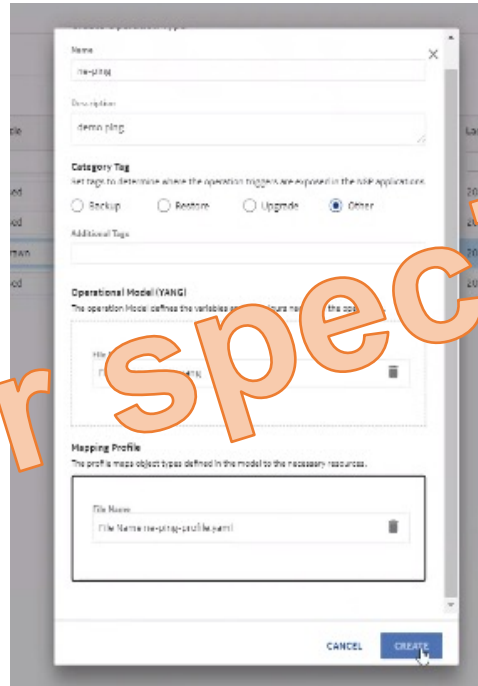
# Current Software Upgrade Tools



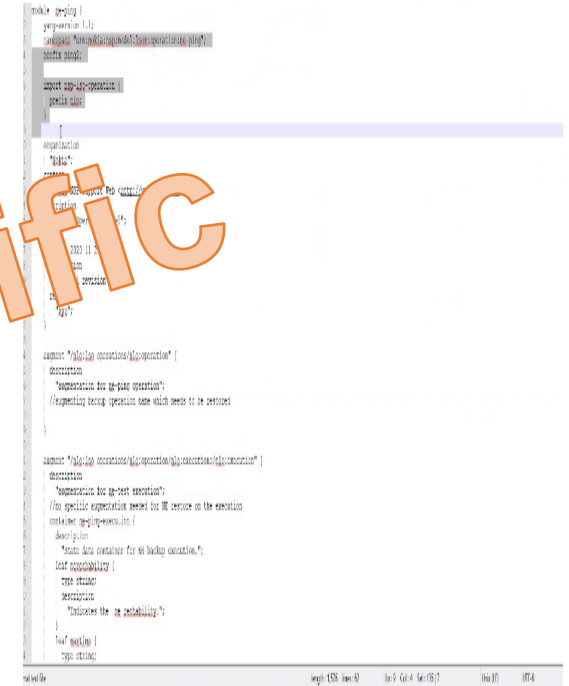
Manual using CLI

```
PORT A/1 - [WAITING-FOR-NETWORK] -> [PROVISIONING]
PORT A/1 - Auto node provisioning started
PORT A/1 - Started provisioning process for router management interface management
PORT A/1 - Launching DHCP/DHCPv6 process for management interface management, timeout = 20 s
PORT A/1 - DHCP client: nexthop 172.16.1.1 is not reachable for routes 0.0.0.0/0
PORT A/1 - DHCP client: nexthop 172.16.1.1 is not reachable for routes 0.0.0.0/0
PORT A/1 - DHCP finished successfully
PORT A/1 - DHCP parameter: router management interface management
PORT A/1 - DHCP parameter: address = 172.16.18.2
PORT A/1 - DHCP parameter: mask = 255.255.255.0
PORT A/1 - DHCP parameter: lease time = 7200s
PORT A/1 - DHCP parameter: server address = 192.168.132.74
PORT A/1 - DHCP parameter: gateway = 172.16.18.1
```

ZTP



Network Management Software



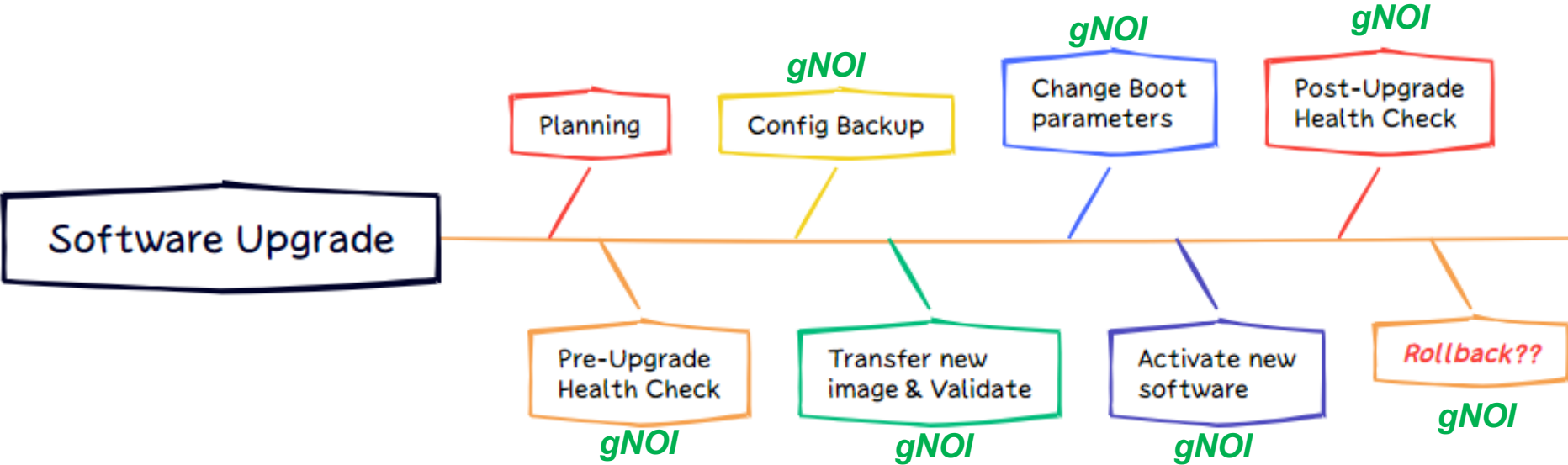
Scripting

Vendor specific

# gNOI

- gRPC Network Operations Interface
- gRPC based service for executing operational commands
- Standards defined by OpenConfig - <https://github.com/openconfig/gnoi>

# Software Upgrade MOP



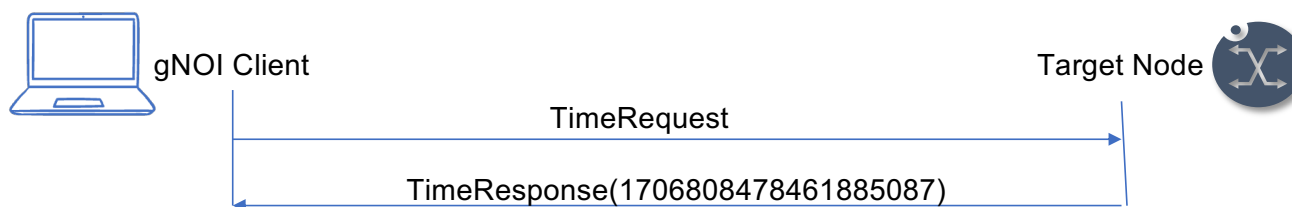
# gRPC Introduction

- RPC framework using HTTP 2.0 as underlying transport
- Does not expose HTTP 2.0 to the user (unlike REST)
- Uses a binary payload
- HTTP 2.0 helps with efficient management of connections
- Requires gRPC software on both client and server



# How gNOI works?

- Example of using SYSTEM service to get time on target node.



## Request

```
> Frame 28: 234 bytes on wire (1872 bits), 234 bytes captured (1872 bits)
> Ethernet II, Src: RealtekU_c8:a1:d3 (52:54:00:c8:a1:d3), Dst: fa:ac:c0:01:04:00 (fa:ac:c0:01:04:00)
> Internet Protocol Version 4, Src: 10.10.10.100, Dst: 10.10.10.104
> Transmission Control Protocol, Src Port: 48834, Dst Port: 57400, Seq: 34, Ack: 64, Len: 168
> HyperText Transfer Protocol 2
> GRPC Message: /gnoi.system.System/Time, Request
```

## Response

```
> Frame 30: 301 bytes on wire (2408 bits), 301 bytes captured (2408 bits)
> Ethernet II, Src: fa:ac:c0:01:04:00 (fa:ac:c0:01:04:00), Dst: RealtekU_c8:a1:d3 (52:54:00:c8:a1:d3)
> Internet Protocol Version 4, Src: 10.10.10.104, Dst: 10.10.10.100
> Transmission Control Protocol, Src Port: 57400, Dst Port: 48834, Seq: 64, Ack: 202, Len: 235
> HyperText Transfer Protocol 2
> GRPC Message: /gnoi.system.System/Time, Response
> Protocol Buffers: /gnoi.system.System/Time,response
```

# List of current gNOI Services and RPCs

## **SYSTEM**

Ping  
Traceroute  
Time  
SetPackage  
SwitchControlProcessor  
Reboot  
RebootStatus  
CancelReboot  
KillProcess

## **FILE**

Get  
TransferToRemote  
Put  
Stat  
Remove

## **FACTORYRESET**

Start

## **HEALTHZ**

Get  
List  
Acknowledge  
Artifact  
Check






## **LINKQUALIFICATION**

Create            Delete  
Get                List  
Capabilities



## **OS**

Install  
Activate  
Verify



# gNOI Services - System

- gNOI System service – allows client to perform operational tasks on target node.
- gNOI System RPCs:
  - PING 
  - TRACEROUTE
  - REBOOT 
  - REBOOTSTATUS 
  - CANCELREBOOT 
  - SWITCHCONTROLPROCESSOR 

# gNOI Services - File

- gNOI File service – allows client to transfer files to and from the target node.
- gNOI File RPCs:
  - GET 
  - PUT
  - REMOVE
  - STAT 




# gNOI Services - Healthz

- gNOI Healthz service – allows client to validate the health of target node components.
- gNOI Healthz RPCs:
  - CHECK 
  - GET 
  - LIST

## Status

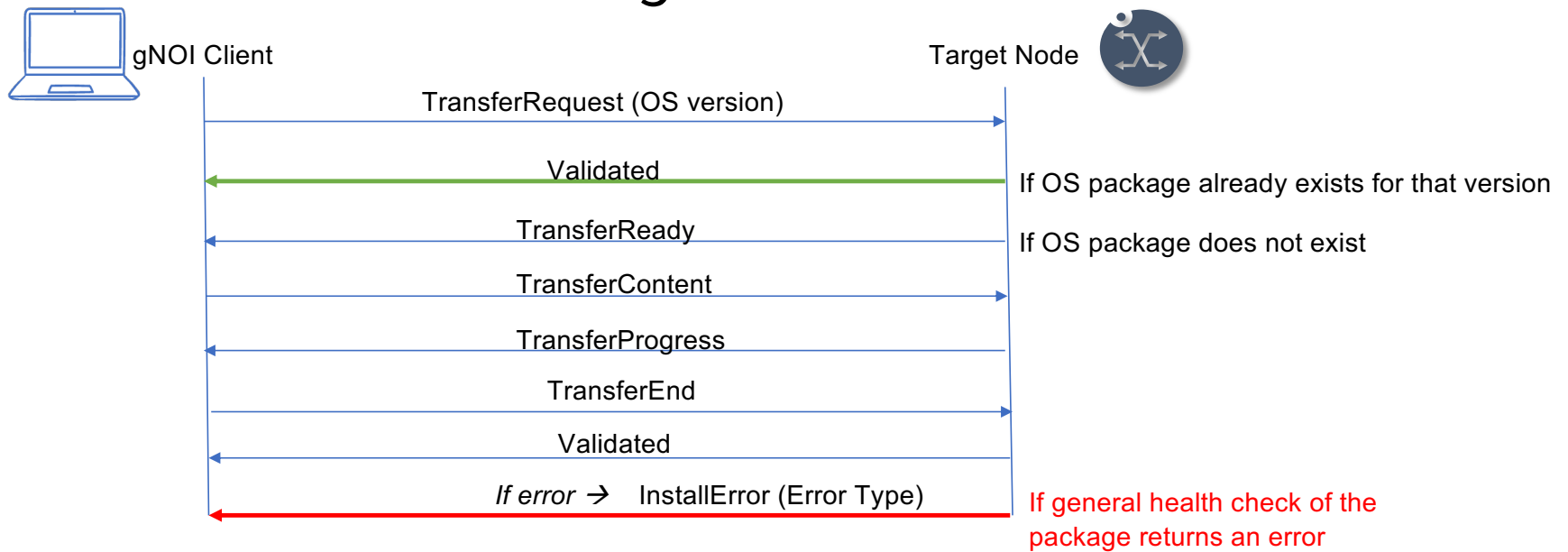
NAME	NUMBER	DESCRIPTION
STATUS_UNSPECIFIED	0	The path doesn't support healthz
STATUS_HEALTHY	1	The path is healthy
STATUS_UNHEALTHY	2	The path is unhealthy

# gNOI Services - OS

- gNOI OS service – allows client to install an OS package on a target node.
- gNOI Healthz RPCs:
  - INSTALL 
  - ACTIVATE 
  - VERIFY 

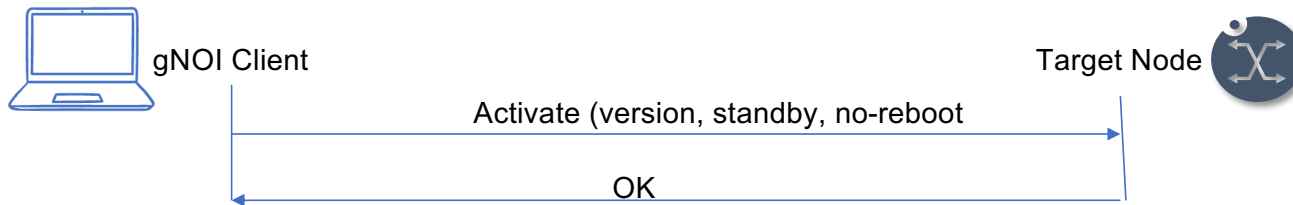
# gNOI OS Service – INSTALL RPC

- Transfers the OS package to the target node.
- Bi-directional streaming RPC.



# gNOI OS Service – ACTIVATE RPC

- Sets the requested OS version for the target node to use at the next reboot.
- Optional flag to avoid reboot during activation.



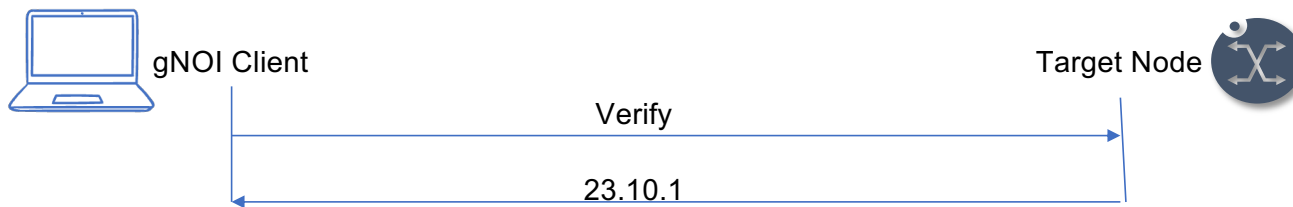


# REBOOT RPC (SYSTEM SERVICE)

- If no-reboot flag is used with ACTIVATE RPC, the target node can be rebooted later using the SYSTEM service REBOOT RPC.

# gNOI OS Service – VERIFY RPC

- Checks the OS version running on the target node.



# gNOI Client

- gNOIc – gNOI CLI client that provides support for select gNOI services
- <https://gnoic.kmrd.dev/>
- <https://gnxi.srlinux.dev/>



File v0.1.0

gNOI Protobuf Documentation

file

SEARCH

Search...

file

S SERVICES

- File

M MESSAGES

- GetRequest
- GetResponse
- PutRequest
- PutRequest.Details
- PutResponse
- RemoveRequest
- RemoveResponse
- StatInfo
- StatRequest
- StatResponse
- TransferToRemoteRequest
- TransferToRemoteResponse

SERVICES

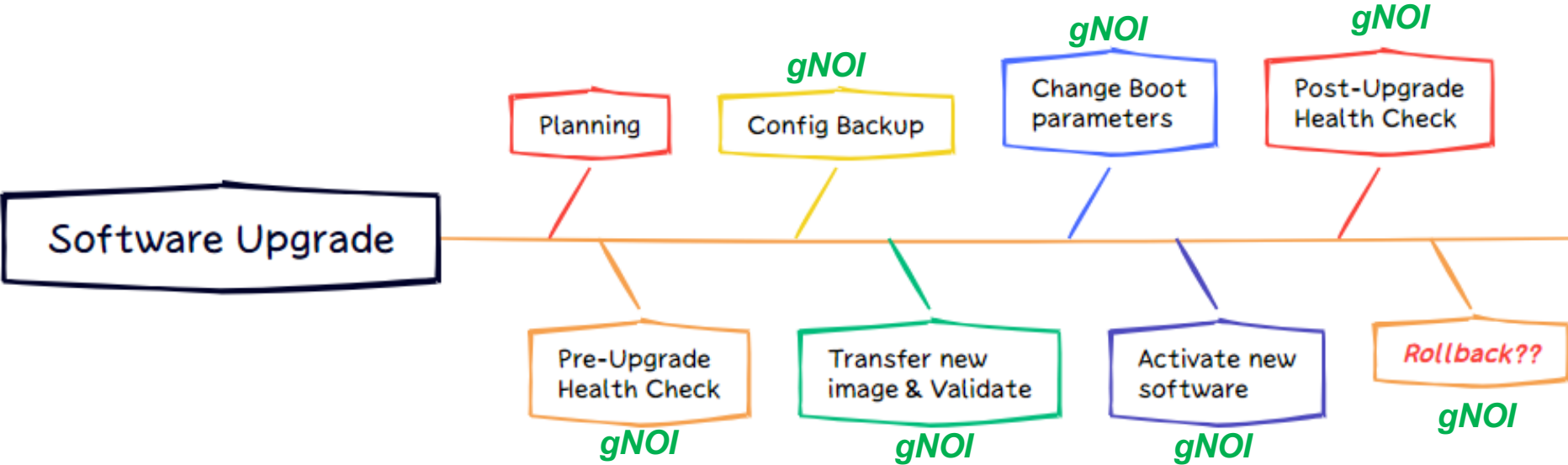
File

METHOD NAME	REQUEST / RESPONSE TYPE	DESCRIPTION
<b>Get</b>	<a href="#">GetRequest</a> <a href="#">GetResponse</a>	Get reads and streams the contents of a file from the target. The file is streamed by sequential messages, each containing up to 64KB of data. A final message is sent prior to closing the stream that contains the hash of the data sent. An error is returned if the file does not exist or there was an error reading the file.
<b>TransferToRemote</b>	<a href="#">TransferToRemoteRequest</a> <a href="#">TransferToRemoteResponse</a>	TransferToRemote transfers the contents of a file from the target to a specified remote location. The response contains the hash of the data transferred. An error is returned if the file does not exist, the file transfer fails, or if there was an error reading the file. This is a blocking call until the file transfer is complete.
<b>Put</b>	<a href="#">PutRequest</a> stream <a href="#">PutResponse</a> stream	Put streams data into a file on the target. The file is sent in sequential messages, each message containing up to 64KB of data. A final message must be sent that includes the hash of the data sent. An error is returned if the location does not exist or there is an error writing the data. If no checksum is received, the target must assume the operation is incomplete and remove the partially transmitted file. The target should initially write the file to a temporary location so a failure does not destroy the original file.
<b>Stat</b>	<a href="#">StatRequest</a> <a href="#">StatResponse</a>	Stat returns metadata about a file on the target. An error is returned if the file does not exist or there is an error in accessing the metadata.
<b>Remove</b>	<a href="#">RemoveRequest</a> <a href="#">RemoveResponse</a>	Remove removes the specified file from the target. An error is returned if the file does not exist, is a directory, or the remove operation encounters an error (e.g., permission denied).



# DEMO

# Software Upgrade MOP



# PRE-UPGRADE HEALTH CHECK

```
[root@localhost ~]#  
[root@localhost ~]# gnoic --config gnoic.yaml --format json healthz get --path /  
platform/fan-tray[id=1]
```

I

# CONFIG BACKUP

```
[root@localhost ~]#
```

```
[root@localhost ~]# gnoic --config gnoic.yaml os verify
```

```
I
```

# SOFTWARE TRANSFER

```
[root@localhost ~]#  
[root@localhost ~]# gnoic --config gnoic.yaml os install --version srlinux_23.10.1-218  
--pkg /opt/23.10/srlinux-23.10.1-218.bin
```

I



# SOFTWARE ACTIVATE

```
[root@localhost ~]#  
[root@localhost ~]#  
[root@localhost ~]# gnoic --config gnoic.yaml os verify --format json
```

I

# SOFTWARE VERIFY

```
[root@localhost ~]#  
[root@localhost ~]#  
[root@localhost ~]# gnoic --config gnoic.yaml os verify --format json  
{  
  "target": "138.120.180.160:57400",  
  "response": {  
    "version": "23.7.2-84",  
    "verify_standby": {  
      "State": {  
        "StandbyState": {  
          "state": 1  
        }  
      }  
    }  
  }  
}  
[root@localhost ~]# gnoic --config gnoic.yaml os activate --version 23.10.1-218  
INFO[0005] target "138.120.180.160:57400" activate response "activate_ok:{}"  
[root@localhost ~]# █
```

# Summary

- Streamline the upgrade process in a multi-vendor network using a standards-based service like gNOI
- gNOI can simplify the software upgrade procedure by automating key steps of the software upgrade MOP.
- The client gNOIc supports gNOI services



**Thank you**