



Bgp At Doors of Autonomous Systems is Simple (BADASS)

Summary: This document is a subject on network administration.

Contents

I	Preamble	2
II	Introduction	3
III	General guidelines	4
IV	Mandatory part	5
IV.1	Part 1: GNS3 configuration with Docker	6
IV.2	Part 2: Discovering a VXLAN	8
IV.3	Part 3: Discovering BGP with EVPN	11
V	Submission and peer-evaluation	16

Chapter I

Preamble



Chapter II

Introduction

The purpose of this project is to expand on the knowledge you have gained through Net-Practice. You will have to simulate a network and configure it using **GNS3** with images **docker**.

BGP EVPN is based on BGP (RFC 4271) and its extensions MP-BGP (RFC4760). BGP is the routing protocol that drives the Internet. Through MP-BGP extensions, it can be used to carry reachability information (NLRI) for various protocols (IPv4, IPv6, L3 VPN and in this case, EVPN). EVPN is a special family for publishing information about MAC addresses and the end devices that access them..

Chapter III

General guidelines

- The whole project has to be done in a **virtual machine**.
- This project involves using and installing **docker** as well as **GNS3**.
- You have to put all the configuration files of your project in folders located at the root of your repository (go to Submission and peer-evaluation for more information). The folders of the mandatory part will be named: P1, P2 and P3.



WARNING: This project uses many new concepts Don't be afraid to take some time to read up on how BGP and VXLANs work. Many new terms are voluntarily used.

Chapter IV

Mandatory part

This project will consist of setting up several environments under specific rules.

It is divided into three parts you have to do in the following order:

- Part 1: GNS3 configuration with Docker.
- Part 2: Discovering a VXLAN.
- Part 3: Discovering BGP with EVPN.



You must read the **whole subject** to understand what you need to do.

IV.1 Part 1: GNS3 configuration with Docker

For this first part you will have to configure **GNS3**. It is thus necessary to install and configure **GNS3** as well as **docker** in your virtual machine.

Now that everything works you need to use two images **docker** that you have to make.

A first image with a system of your choice containing at least **busybox** or an equivalent solution.



Alpine seems to be a good solution.

A second image using a system of your choice with the following constraints:

- A software that manages packet routing (zebra or quagga)..
- The service **BGPD** active and configured.
- The service **OSPFD** active and configured.
- An **IS-IS** routing engine service.
- **Busybox** or an equivalent.



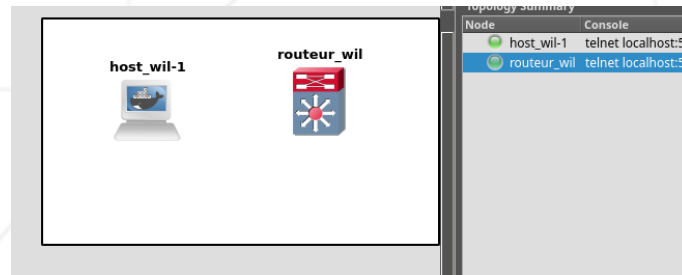
There are pre-build images that need to be configured with this kind of service. Your containers must work in **GNS3** with the requested services. You can add what you wish to realize this project



Warning: Your images will be used throughout this project. No IP address should be configured by default.

Bgp At Doors of Autonomous Systems is Simple (BADASS)

You must use these two images **docker** in **GNS3** and realize this small diagram. You need to have both machines working. We must be able to connect to them by **GNS3**.



The name of the machines is not put at random it will be necessary to have your login in the name of each equipment (here wil).

Below is an example of each image configured in GNS3:

```
/ # ps
PID  USER      TIME  COMMAND
    1  root         0:00  /bin/sh
    38  root         0:00  /gns3/bin/busybox script -qfc while true; do TERM=vt100 /g
    44  root         0:00  /bin/ash -c while true; do TERM=vt100 /gns3/bin/busybox sh
    45  root         0:00  /gns3/bin/busybox sh
    59  root         0:00  ps
/ # hostname
host_wil-1
/ #

/ # ps
PID  USER      TIME  COMMAND
    1  root         0:00  /sbin/tini -- /usr/lib/frr/docker-start
    38  root         0:00  /gns3/bin/busybox script -qfc while true; do TERM=vt100 /g
    46  root         0:00  /gns3/bin/busybox sh
    94  root         0:00  tail -f /dev/null
   136  frr         0:00  /usr/lib/frr/zebra -d -F traditional -A 127.0.0.1 -s 90000
   141  frr         0:00  /usr/lib/frr/bgpd -d -F traditional -A 127.0.0.1
   148  frr         0:00  /usr/lib/frr/ospfd -d -F traditional -A 127.0.0.1
   151  frr         0:00  /usr/lib/frr/isisd -d -F traditional -A 127.0.0.1
   254  root         0:00  ps
/ # hostname
routeur_wil
/ #
```

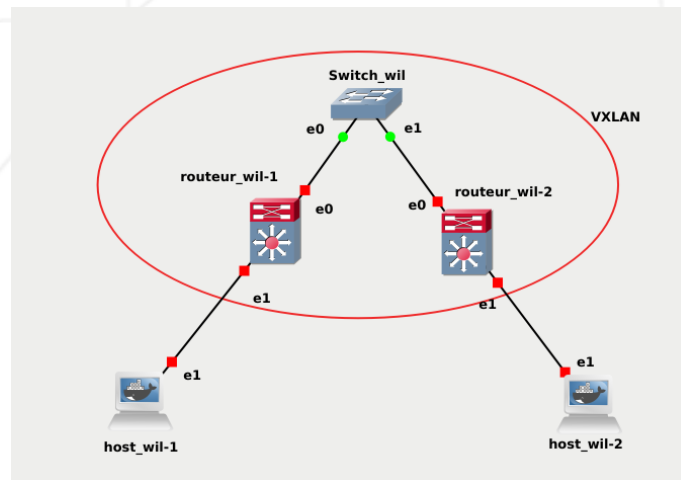
You must render this project in a P1 folder at the root of your git repository. You should also add the configuration files with comments to explain the set up of each equipment.



You must export this project with a ZIP compression including the image bases. This file must be visible in your git repository

IV.2 Part 2: Discovering a VXLAN

You now have a functional basis to start setting up your first VXLAN (RFC 7348) network. First in static then in dynamic multicast. Here is the topology of your first VXLAN:



You must configure this network using a VXLAN with an ID of 10 as shown in the examples below. You can use any VXLAN name you like here: `vxlan10`. You must set up a bridge here: `br0`. You must configure your ETHERNET interfaces as you wish. Below is an example of the expected result when we inspect the traffic between our two machines in our VXLAN

```
# hostname
host_wil-1
# ifconfig eth1
eth1  Link encap:Ethernet  HWaddr DA:DA:AB:D7:54:C4
       inet addr:30.1.1.1  Bcast:0.0.0.0  Mask:255.255.255.0
       inet6 addr: fe80::d8da:abff:fed7:54c4/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:50 errors:0 dropped:0 overruns:0 frame:0
       TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:3972 (3.8 KiB)  TX bytes:1216 (1.1 KiB)

# ping 30.1.1.2
PING 30.1.1.2 (30.1.1.2): 56 data bytes
64 bytes from 30.1.1.2: seq=0 ttl=64 time=0.443 ms
^C
--- 30.1.1.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.443/0.443/0.443 ms
#
```

```
# hostname
host_wil-2
# ifconfig eth1
eth1  Link encap:Ethernet  HWaddr 92:E6:7C:16:F3:18
       inet addr:30.1.1.2  Bcast:0.0.0.0  Mask:255.255.255.0
       inet6 addr: fe80::90e6:7cff:fe16:f318/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:28 errors:0 dropped:0 overruns:0 frame:0
       TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:2180 (2.1 KiB)  TX bytes:1216 (1.1 KiB)
#
```

No.	Time	Source	Destination	Protocol	Length	Info
5	30.726761	30.1.1.1	30.1.1.2	ICMP	148	Echo (ping) request id=0x0039 seq=0/0
6	36.796971	30.1.1.2	30.1.1.1	ICMP	148	Echo (ping) reply id=0x0039 seq=0/0

```
> Frame 5: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
> Ethernet II, Src: ca:f0:be:8d:3b:fa (ca:f0:be:8d:3b:fa), Dst: 46:41:59:cc:db:8f (46:41:59:cc:db:8f)
> Internet Protocol Version 4, Src: 10.1.1.1, Dst: 10.1.1.2
> User Datagram Protocol, Src Port: 59899, Dst Port: 4789
> Virtual extensible Local Area Network
  > Flags: 0x0000, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 10
    Reserved: 0
  > Ethernet II, Src: da:da:ab:d7:54:c4 (da:da:ab:d7:54:c4), Dst: 92:e6:7c:16:f3:18 (92:e6:7c:16:f3:18)
> Internet Protocol Version 4, Src: 30.1.1.1, Dst: 30.1.1.2
> Internet Control Message Protocol
  0000  46 41 59 cc db 8f ca f6
  0010  00 06 50 a3 00 00 40 15
```

Bgp At Doors of Autonomous Systems is Simple (BADASS)

We are now going to see the same thing using the groups whose goal will be to be able to make a dynamic multicast.

The screenshot displays a network configuration and packet capture. The top part shows a Wireshark capture on interface eth0 of router_wil-1. The capture shows several ARP requests and replies, and ICMP echo requests and replies. A red circle highlights a packet from the router to host_wil-1. Below the Wireshark window, two terminal windows show the configuration and ping results for host_wil-1 and host_wil-2.

```
# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 12:5A:DE:F7:EA:53
          inet addr:30.1.1.1  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::105a:deff:fe7f:ea53/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3760 (3.6 KiB)  TX bytes:936 (936.0 B)

# # hostname
host_wil-1
# # ping 30.1.1.2
PING 30.1.1.2 (30.1.1.2): 56 data bytes
64 bytes from 30.1.1.2: seq=0 ttl=64 time=0.907 ms
64 bytes from 30.1.1.2: seq=1 ttl=64 time=0.573 ms
^C
--- 30.1.1.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.573/0.740/0.907 ms
# #
```

```
# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr EA:A6:14:12:8E:A4
          inet addr:30.1.1.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::e8a6:14ff:fe12:8ea4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25 errors:0 dropped:1 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1954 (1.9 KiB)  TX bytes:1006 (1006.0 B)

# # hostname
host_wil-2
# #
```

We can notice that our machines now have a group (here 239.1.1.1 you can modify this part):

```
# ip -d link show vxlan10
3: vxlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master br0 state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether da:83:84:19:04:d5 brd ff:ff:ff:ff:ff:ff promiscuity 1 minmtu 68 maxmtu 65535
    vxlan id 10 group 239.1.1.1 dev eth0 srcport 0 dstport 4789 ttl auto ageing 300 udpchecksum noudp6zerocsumtx noudp6zerocsumrx
    bridge_slave state forwarding priority 32 cost 100 hairpin off guard off root_block off fastleave off learning on flood on flood on port id 0x8002 port no 0x2 designated port 32770 designated cost 0 designated bridge 8000.DA:83:84:19:04:D5 designated root 8000.DA:83:84:19:04:D5 hold timer 0.00 message age timer 0.00 forward delay timer 0.00 topology change ack 0 config pending 0 proxy arp off proxy arp wifi off mcast router 1 mcast fast leave off mcast flood on mcast to unicast off neigh suppress off group fwd_mask 0 group_fwd_mask_str 0x0 vlan_tunnel off isolated off addrgenmode eui64 numtxqueues 1 numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
# # hostname
router_wil-1
# #
```

```
# ip -d link show vxlan10
3: vxlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master br0 state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether ae:df:08:4c:0c:0d brd ff:ff:ff:ff:ff:ff promiscuity 1 minmtu 68 maxmtu 65535
    vxlan id 10 group 239.1.1.1 dev eth0 srcport 0 dstport 4789 ttl auto ageing 300 udpchecksum noudp6zerocsumtx noudp6zerocsumrx
    bridge_slave state forwarding priority 32 cost 100 hairpin off guard off root_block off fastleave off learning on flood on flood on port id 0x8002 port no 0x2 designated port 32770 designated cost 0 designated bridge 8000.02:AF:63:0B:5E:00 designated root 8000.02:AF:63:0B:5E:00 hold timer 0.00 message age timer 0.00 forward delay timer 0.00 topology change ack 0 config pending 0 proxy arp off proxy arp wifi off mcast router 1 mcast fast leave off mcast flood on mcast to unicast off neigh suppress off group_fwd_mask 0 group_fwd_mask_str 0x0 vlan_tunnel off isolated off addrgenmode eui64 numtxqueues 1 numrxqueues 1 gso_max_size 65536 gso_max_segs 65535
# # hostname
router_wil-2
# #
```

Bgp At Doors of Autonomous Systems is Simple (BADASS)

Below is an example of how to display our mac address table in our two routers:

```

# /usr/bin/zsh 59x24
/ # hostname
routeur_wil-1
/ # brctl showmacs br0
port no mac addr      is local?   ageing time
r
2  02:af:63:db:5e:00    no          136.98
1  12:5a:de:f7:ea:53    no          202.52
2  ae:df:08:4c:0c:0d    no          196.37
2  da:83:84:19:04:d5    yes         0.00
2  da:83:84:19:04:d5    yes         0.00
2  ea:a6:14:12:8e:a4    no          215.32
1  ee:3f:ab:b3:36:38    yes         0.00
1  ee:3f:ab:b3:36:38    yes         0.00
/ #

# /usr/bin/zsh 63x24
/ # hostname
routeur_wil-2
/ # brctl showmacs br0
port no mac addr      is local?   ageing timer
r
1  02:af:63:db:5e:00    yes         0.00
1  02:af:63:db:5e:00    yes         0.00
2  12:5a:de:f7:ea:53    no          169.23
2  ae:df:08:4c:0c:0d    yes         0.00
2  ae:df:08:4c:0c:0d    yes         0.00
2  da:83:84:19:04:d5    no          21.77
1  ea:a6:14:12:8e:a4    no          182.03
/ #
```

You must render this project in a P2 folder at the root of your git repository. You should also add the configuration files with comments to explain the set up of each equipment.



You must export this project with a ZIP compression including the image bases. This file must be visible in your git repository.



You must use correct and consistent names for your equipment here with the login of one of the group members.

IV.3 Part 3: Discovering BGP with EVPN

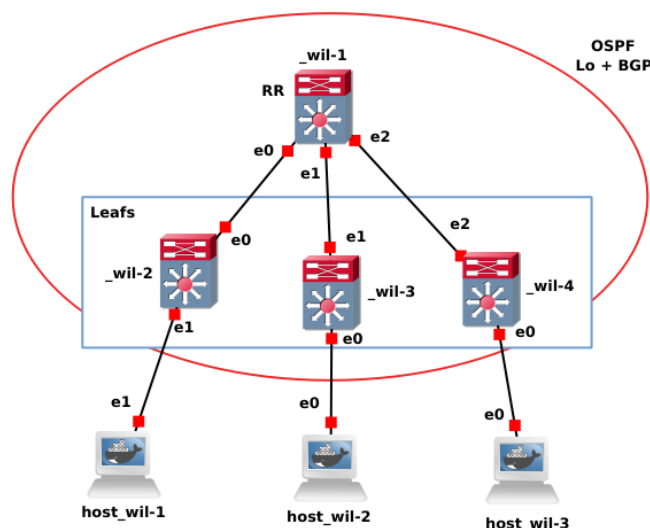
Now that you have mastered the basic principle of the VXLAN we will go a little further and explore the principle of the BGP EVPN (rfc 7432) without using MPLS to simplify things. The controller will learn the MAC addresses. We will use our VXLAN with ID 10 seen in the previous part.

As in the second part we start with the topology of the expected network. We are going to use the principle of the route reflection (=RR). Our leafs (VTEP) will be configured to have dynamic relations.

This diagram represents a small datacenter.



For the sake of readability the names are shorter here. You will have to use OSPF to simplify the evaluation.



We can see our visibility from our VTEP _wil-4 the 3 VTEPs 1.1.[1.4]

```
wil-4(config-router)# do sh ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, I - IS-IS, B - BGP, E - EIGRP, N - NHRP,
       T - Table, v - VNC, V - VNC-Direct, A - Babel, D - SHARP,
       F - PBR, f - OpenFabric,
       > - selected route, * - FIB route, q - queued, r - rejected, b - backup

O>* 1.1.1.1/32 [110/10000] via 10.1.1.9, eth2, weight 1, 00:00:44
O>* 1.1.1.3/32 [110/20000] via 10.1.1.9, eth2, weight 1, 00:00:44
O   1.1.1.4/32 [110/0] is directly connected, lo, weight 1, 00:00:58
C>* 1.1.1.4/32 is directly connected, lo, 00:00:58
O>* 10.1.1.0/30 [110/20000] via 10.1.1.9, eth2, weight 1, 00:00:44
O>* 10.1.1.4/30 [110/20000] via 10.1.1.9, eth2, weight 1, 00:00:44
O   10.1.1.8/30 [110/10000] is directly connected, eth2, weight 1, 00:00:49
C>* 10.1.1.8/30 is directly connected, eth2, 00:00:58
wil-4(config-router)#
```

Bgp At Doors of Autonomous Systems is Simple (BADASS)

We have only one route for the moment with our controller (RR) :

```
wil-4(config-router)# do sh bgp summary

IPv4 Unicast Summary:
BGP router identifier 1.1.1.4, local AS number 1 vrf-id 0
BGP table version 0
RIB entries 0, using 0 bytes of memory
Peers 1, using 14 KiB of memory

Neighbor      V      AS  MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd  PfxSnt
1.1.1.1        4        1         7         7         0     0     0 00:02:12      0           0

Total number of neighbors 1

L2VPN EVPN Summary:
BGP router identifier 1.1.1.4, local AS number 1 vrf-id 0
BGP table version 0
RIB entries 1, using 192 bytes of memory
Peers 1, using 14 KiB of memory

Neighbor      V      AS  MsgRcvd  MsgSent  TblVer  InQ  OutQ  Up/Down  State/PfxRcd  PfxSnt
1.1.1.1        4        1         7         7         0     0     0 00:02:12      0           1

Total number of neighbors 1
wil-4(config-router)#
```

When there is no host running we can see our VNI (10 here) as well as our pre-configured routes (type 3). No route type 2 seems to exist and it is quite normal.

```
wil-4(config-router)# do sh bgp l2vpn evpn
BGP table version is 1, local router ID is 1.1.1.4
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
EVPN type-1 prefix: [1]:[ESI]:[EthTag]:[IPlen]:[VTEP-IP]
EVPN type-2 prefix: [2]:[EthTag]:[MAClen]:[MAC]:[IPlen]:[IP]
EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
EVPN type-4 prefix: [4]:[ESI]:[IPlen]:[OrigIP]
EVPN type-5 prefix: [5]:[EthTag]:[IPlen]:[IP]

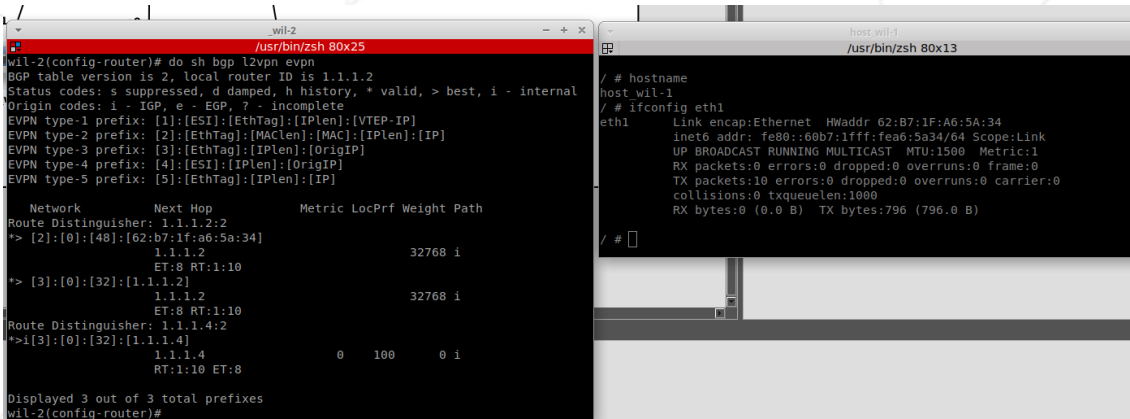
  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 1.1.1.2:2
*>i[3]:[0]:[32]:[1.1.1.2]
  1.1.1.2                      0    100      0 i
  RT:1:10 ET:8
Route Distinguisher: 1.1.1.4:2
*> [3]:[0]:[32]:[1.1.1.4]
  1.1.1.4                      32768 i
  ET:8 RT:1:10

Displayed 2 out of 2 total prefixes
wil-4(config-router)#
```

Bgp At Doors of Autonomous Systems is Simple (BADASS)

A machine `host_wil-1` is now functional. We can notice that without assigning an IP address our VTEP (`wil_2`) automatically discovers the MAC address of the functional machines.

We can also see the automatic creation of a route type 2:



The image shows two terminal windows. The left window is titled `wil-2` and shows the output of the command `do sh bgp l2vpn evpn` on a router. It displays BGP table version 2, local router ID 1.1.1.2, and various EVPN types and their prefixes. It also shows a table of routes with columns for Network, Next Hop, Metric, LocPrf, Weight, and Path. The right window is titled `host_wil-1` and shows the output of the command `show interface eth1`, displaying details about the Ethernet interface, including its MAC address and statistics.

```
wil-2(config-router)# do sh bgp l2vpn evpn
BGP table version is 2, local router ID is 1.1.1.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
EVPN type-1 prefix: [1]:[ESI]:[EthTag]:[IPlen]:[VTEP-IP]
EVPN type-2 prefix: [2]:[EthTag]:[MAClen]:[MAC]:[IPlen]:[IP]
EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
EVPN type-4 prefix: [4]:[ESI]:[IPlen]:[OrigIP]
EVPN type-5 prefix: [5]:[EthTag]:[IPlen]:[IP]

  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 1.1.1.2:2
*> [2]:[0]:[48]:[62:b7:1f:a6:5a:34]
      1.1.1.2
      ET:8 RT:1:10
      32768 i
*> [3]:[0]:[32]:[1.1.1.2]
      1.1.1.2
      ET:8 RT:1:10
      32768 i
Route Distinguisher: 1.1.1.4:2
*> i[3]:[0]:[32]:[1.1.1.4]
      1.1.1.4
      RT:1:10 ET:8
      0 100 0 i
Displayed 3 out of 3 total prefixes
wil-2(config-router)#
```

```
host_wil-1
/usr/bin/zsh 80x13

/ # hostname
host_wil-1
/ # ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 62:B7:1F:A6:5A:34
          inet6 addr: fe80::60b7:1fff:fe6a:5a34/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:796 (796.0 B)

/ #
```

In the same way, when we look at a second VTEP(`_wil-4`), we can notice the creation of a new route type 2 generated by our RR:



The image shows a terminal window titled `_wil-4` displaying the output of the command `do sh bgp l2vpn evpn` on a router. It shows BGP table version 2, local router ID 1.1.1.4, and various EVPN types and their prefixes. It also shows a table of routes with columns for Network, Next Hop, Metric, LocPrf, Weight, and Path.

```
_wil-4(config-router)# do sh bgp l2vpn evpn
BGP table version is 2, local router ID is 1.1.1.4
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
EVPN type-1 prefix: [1]:[ESI]:[EthTag]:[IPlen]:[VTEP-IP]
EVPN type-2 prefix: [2]:[EthTag]:[MAClen]:[MAC]:[IPlen]:[IP]
EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
EVPN type-4 prefix: [4]:[ESI]:[IPlen]:[OrigIP]
EVPN type-5 prefix: [5]:[EthTag]:[IPlen]:[IP]

  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 1.1.1.2:2
*> i[2]:[0]:[48]:[62:b7:1f:a6:5a:34]
      1.1.1.2
      RT:1:10 ET:8
      0 100 0 i
*> i[3]:[0]:[32]:[1.1.1.2]
      1.1.1.2
      RT:1:10 ET:8
      0 100 0 i
Route Distinguisher: 1.1.1.4:2
*> [3]:[0]:[32]:[1.1.1.4]
      1.1.1.4
      ET:8 RT:1:10
      32768 i
Displayed 3 out of 3 total prefixes
_wil-4(config-router)#
```


Bgp At Doors of Autonomous Systems is Simple (BADASS)

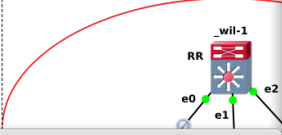
We repeat the operation with a second machine (host_wil-3). We can notice the second route set up by type 2. There is no assignment of IP address:

```
wil-2(config-router)# do sh bgp l2vpn evpn
BGP table version is 2, local router ID is 1.1.1.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete
EVPN type-1 prefix: [1]:[ESI]:[EthTag]:[IPlen]:[VTEP-IP]
EVPN type-2 prefix: [2]:[EthTag]:[MAClen]:[MAC]:[IPlen]:[IP]
EVPN type-3 prefix: [3]:[EthTag]:[IPlen]:[OrigIP]
EVPN type-4 prefix: [4]:[ESI]:[IPlen]:[OrigIP]
EVPN type-5 prefix: [5]:[EthTag]:[IPlen]:[IP]

  Network          Next Hop          Metric LocPrf Weight Path
Route Distinguisher: 1.1.1.2:2
*> [2]:[0]:[48]:[62:b7:1f:a6:5a:34]
      1.1.1.2                      32768 i
      ET:8 RT:1:10
*> [3]:[0]:[32]:[1.1.1.2]
      1.1.1.2                      32768 i
      ET:8 RT:1:10
Route Distinguisher: 1.1.1.4:2
*>i[2]:[0]:[48]:[1e:80:95:23:b8:45]
      1.1.1.4                      0    100    0 i
      RT:1:10 ET:8
*>i[3]:[0]:[32]:[1.1.1.4]
      1.1.1.4                      0    100    0 i
      RT:1:10 ET:8

Displayed 4 out of 4 total prefixes
wil-2(config-router)#
```

For our verification a simple ping allows us to see that we can access all the machines through our RR using the VTEPs. We can see the VXLAN configured to 10 as well as our packets ICMP. We also see packets OSPF configured:



```
host_wil-1 /usr/bin/zsh 80x22
# # hostname
host_wil-1
# # ping 20.1.1.2
PING 20.1.1.2 (20.1.1.2): 56 data bytes
64 bytes from 20.1.1.2: seq=0 ttl=64 time=0.993 ms
64 bytes from 20.1.1.2: seq=1 ttl=64 time=0.660 ms
64 bytes from 20.1.1.2: seq=2 ttl=64 time=0.612 ms
^C
--- 20.1.1.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.612/0.755/0.993 ms
# # ifconfig eth1
eth1  Link encap:Ethernet  HWaddr 62:B7:1F:A6:5A:34
       inet addr:20.1.1.1 Bcast:0.0.0.0 Mask:255.255.255.0
       inet6 addr: fe80::60b7:1fff:fe6d:5a34/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:22 errors:0 dropped:0 overruns:0 frame:0
       TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueueLen:1000
       RX bytes:1664 (1.6 KiB)  TX bytes:1384 (1.3 KiB)

# #
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	ea:1e:d4:81:84:a6	2a:a6:79:c3:92:a0	ARP	42	Who has 10.1.1.2? Tell 10.1.1.1
2	0.000079	2a:a6:79:c3:92:a0	ea:1e:d4:81:84:a6	ARP	42	10.1.1.2 is at 2a:a6:79:c3:92:a0
3	4.608091	10.1.1.2	224.0.0.5	OSPF	82	Hello Packet
4	5.067560	10.1.1.1	224.0.0.5	OSPF	82	Hello Packet
5	14.440283	62:b7:1f:a6:5a:34	Broadcast	ARP	92	Who has 20.1.1.2? Tell 20.1.1.1
6	14.449575	1e:80:95:23:b8:45	62:b7:1f:a6:5a:34	ARP	92	20.1.1.2 is at 1e:80:95:23:b8:45
7	14.449728	20.1.1.1	20.1.1.2	ICMP	148	Echo (ping) request id=0x0029, seq=0
8	14.450029	20.1.1.2	20.1.1.1	ICMP	148	Echo (ping) reply id=0x0029, seq=0
9	14.611476	10.1.1.2	224.0.0.5	OSPF	82	Hello Packet
10	15.067560	10.1.1.1	224.0.0.5	OSPF	82	Hello Packet
11	15.450339	20.1.1.1	20.1.1.2	ICMP	148	Echo (ping) request id=0x0029, seq=1
12	15.450741	20.1.1.2	20.1.1.1	ICMP	148	Echo (ping) reply id=0x0029, seq=1
13	16.455903	20.1.1.1	20.1.1.2	ICMP	148	Echo (ping) request id=0x0029, seq=2
14	16.456291	20.1.1.2	20.1.1.1	ICMP	148	Echo (ping) reply id=0x0029, seq=2
15	19.456560	1e:80:95:23:b8:45	62:b7:1f:a6:5a:34	ARP	92	Who has 20.1.1.1? Tell 20.1.1.2
16	19.456742	62:b7:1f:a6:5a:34	1e:80:95:23:b8:45	ARP	92	20.1.1.1 is at 62:b7:1f:a6:5a:34
17	24.580155	ea:1e:d4:81:84:a6	2a:a6:79:c3:92:a0	ARP	42	Who has 10.1.1.2? Tell 10.1.1.1
18	24.580223	2a:a6:79:c3:92:a0	ea:1e:d4:81:84:a6	ARP	42	10.1.1.2 is at 2a:a6:79:c3:92:a0
19	24.611523	10.1.1.2	224.0.0.5	OSPF	82	Hello Packet
20	25.068015	10.1.1.1	224.0.0.5	OSPF	82	Hello Packet

```
Frame 7: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface 0
  Ethernet II, Src: 2a:a6:79:c3:92:a0 (2a:a6:79:c3:92:a0), Dst: ea:1e:d4:81:84:a6 (ea:1e:d4:81:84:a6)
  Internet Protocol Version 4, Src: 10.1.1.2, Dst: 10.1.1.10
  User Datagram Protocol, Src Port: 53458, Dst Port: 4789
  Virtual extensible Local Area Network
    Flags: 0x0000, VLAN Network ID (VNI)
    Group Policy ID: 0
  VXLAN Network Identifier (VNI): 10
  Reserved: 0
  Ethernet II, Src: 62:b7:1f:a6:5a:34 (62:b7:1f:a6:5a:34), Dst: 1e:80:95:23:b8:45 (1e:80:95:23:b8:45)
  Internet Protocol Version 4, Src: 20.1.1.1, Dst: 20.1.1.2
  Internet Control Message Protocol
```

```
host_wil-3 /usr/bin/zsh 80x12
# # ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 1E:80:95:23:B8:45
       inet addr:20.1.1.2 Bcast:0.0.0.0 Mask:255.255.255.0
       inet6 addr: fe80::1c80:95ff:fe23:b845/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:8 errors:0 dropped:0 overruns:0 frame:0
       TX packets:17 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueueLen:1000
       RX bytes:588 (588.0 B)  TX bytes:1314 (1.2 KiB)
```

Bgp At Doors of Autonomous Systems is Simple (BADASS)

You must render this project in a P3 folder at the root of your git repository. You should also add the configuration files with comments to explain the set up of each equipment.



You must export this project with a ZIP compression including the image bases. This file must be visible in your git repository.



You must use correct and consistent names for your equipment here with the login of one of the group members.

Chapter V

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your folders and files to ensure they are correct.

Reminder:

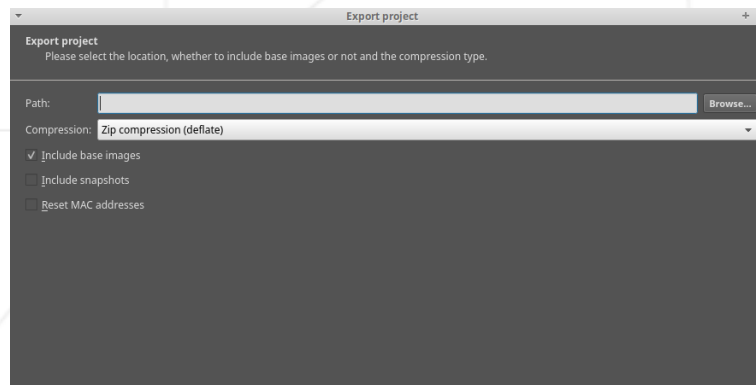
- Turn the mandatory part in three folders located at the root of your repository: P1, P2 et P3.

Below is an example of the expected directory structure:

```
> find -maxdepth 2 -ls
424242      4 drwxr-xr-x  6 wandre  wil42      4096 sept. 17 23:42 .
424242      4 drwxr-xr-x  3 wandre  wil42      4096 sept. 17 23:42 ./P1
424242      4 -rw-r--r--  1 wandre  wil42      XXXX sept. 17 23:42 ./P1/P1.gns3project
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P1/_wil-1_host
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P1/_wil-2
424242      4 drwxr-xr-x  3 wandre  wil42      4096 sept. 17 23:42 ./P2
424242      4 -rw-r--r--  1 wandre  wil42      XXXX sept. 17 23:42 ./P2/P2.gns3project
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P2/_wil-1_g
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P2/_wil-1_host
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P2/_wil-1_s
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P2/_wil-2_g
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P2/_wil-2_host
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P2/_wil-2_s
424242      4 drwxr-xr-x  3 wandre  wil42      4096 sept. 17 23:42 ./P3
424242      4 -rw-r--r--  2 wandre  wil42      4096 sept. 17 23:42 ./P3/P3.gns3project
424242      4 -rw-r--r--  2 wandre  wil42      4096 sept. 17 23:42 ./P3/_wil-1
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P3/_wil-1_host
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P3/_wil-2
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P3/_wil-2_host
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P3/_wil-3
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P3/_wil-3_host
424242      4 -rw-r--r--  2 wandre  wil42      XXXX sept. 17 23:42 ./P3/_wil-4
> file P3/P3.gns3project
P3/P3.gns3project: Zip archive data, at least v2.0 to extract
```

Bgp At Doors of Autonomous Systems is Simple (BADASS)

To export your projects in zip format go to the menu file then export portable project:



During the evaluation you will have to explain the terms used in the subject. We strongly encourage you to take time to understand each of these.



The evaluation process will happen on the computer of the evaluated group.