TÉCNICO LISBOA

Segurança e Arquiteturas Avançadas de Redes

Module 2 Report

METI 2022/2023

Report written by:

Group 7

Afonso Dias - ist193571

João Pereira - ist193587

Francisco Correia - ist176470

TÉCNICO LISBOA

Table of contents

IPSec using ESP in tunnel mode

Network structure

The goal of this laboratory was to connect the sites of the organization with an IPSec tunnel between R1 and R2 with the following ISAKMP policy:

- Hashing based on SHA
- Authentication based on Pre-Shared Keys (PSK)
- Group using DH group 5
- Lifetime equal to 1 day
- Encryption based on AES with 256 bits

The IPSec policy used ESP protocol in tunnel mode with AES as the encryption algorithm and SHAC-HMAC as the authentication method.
For the public networks (200.1.1.0/24 and 200.2.2.0/24) we use OSPF as the routing protocol. The next figure illustrates the network topology used for this exercise.

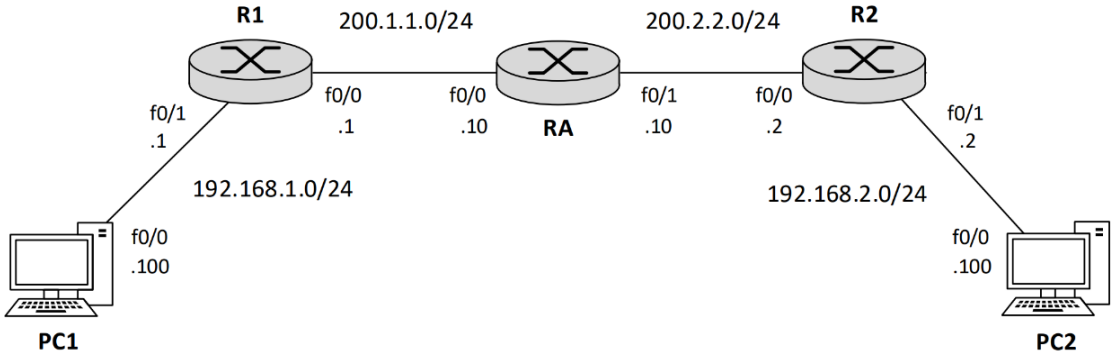


Figure 1: Network topology

IPsec stands for Internet Protocol Security and it is a Internet Protocol extension with a suite of protocols for securing IP communications at the network layer. IPsec provides confidentiality, integrity, authentication and anti-replay. ESP stands for Encapsulating Security Payload and is a protocol used for providing encryption and authentication services.

Internet Key Exchange

In order to implement this framework we need two IPSec peers with a built IPSec tunnel. Firstly an IPsec's negotiation must occur to make the tunnel operational:

1) An Internet Security Association Key Management Protocol (ISAKMP) tunnel is initiated when the R1 and R2 exchange "interesting" traffic. In order to do that we configured a firewall containing the interesting traffic as an access-list on both routers, as we can see in the next figure. So, for example, a ping to PC2 from PC1 can be considered as an interesting traffic.

```
!Define interesting traffic
access-list 110 permit ip 192.168.1.0 0.0.0.255 192.168.2.0 0.0.0.255
```

Figure 2: R1 Access-list with the interesting traffic

The next figure represents all the ISAKMP messages exchanged between R1 and R2 after a ping performed to PC2 from PC1.The firsts 6 messages comprehend the Internet Key Exchange phase 1 and the last 3 the Internet Key Exchange phase 2, explained next;

2038	4419.958043	200.1.1.1	200.2.2.2	ISAKMP	210 Identity Protection (Main Mode)
2039	4420.002384	200.2.2.2	200.1.1.1	ISAKMP	150 Identity Protection (Main Mode)
2040	4420.027450	200.1.1.1	200.2.2.2	ISAKMP	414 Identity Protection (Main Mode)
2041	4420.073992	200.2.2.2	200.1.1.1	ISAKMP	434 Identity Protection (Main Mode)
2042	4420.189013	200.1.1.1	200.2.2.2	ISAKMP	790 Identity Protection (Main Mode)
2046	4422.002607	200.2.2.2	200.1.1.1	ISAKMP	758 Identity Protection (Main Mode)
2059	4423.724142	200.1.1.1	200.2.2.2	ISAKMP	230 Quick Mode
2060	4423.802991	200.2.2.2	200.1.1.1	ISAKMP	230 Quick Mode
2061	4423.857467	200.1.1.1	200.2.2.2	ISAKMP	102 Quick Mode

Figure 3: Network capture with the ISAKMP messages exchanged after a ping

2) **IKE (Internet Key Exchange) Phase 1**: R1 and R2 negotiate the ISAKMP SA policy. After they both agree on the policy and are authenticated, the secure tunnel is created. This phase has three communication steps:

2.1) Step 1 ISAKMP **Security Association(SA)**: R1 and R2 exchange their ISAKMP policy configuration. This ISAKMP policy exchange messages allow the IPsec peers to negotiate and agree upon the security parameters and policies for their communication. This includes aspects such as encryption algorithms, integrity algorithms, Diffie-hellman group, lifetime values, and authentication methods. These messages enforce policies, compatibility, interoperability and are essential for the establishment of the Security Association (SA) between IPsec peers. The next figure contains the packet details regarding the R1 ISAKMP policy sent from R1 to R2 after the ping. We could conclude after analyzing the capture that the details correspond to our ISAKMP configuration done on R1 (figure 5).

```
▾ Payload: Transform (3) # 1
    Next payload: NONE / No Next Payload  (0)
    Reserved: 00
    Payload length: 40
    Transform number: 1
    Transform ID: KEY_IKE (1)
    Reserved: 0000
  › IKE Attribute (t=1,l=2): Encryption-Algorithm: AES-CBC
  › IKE Attribute (t=14,l=2): Key-Length: 256
  › IKE Attribute (t=2,l=2): Hash-Algorithm: SHA
  › IKE Attribute (t=4,l=2): Group-Description: 1536 bit MODP group
  › IKE Attribute (t=3,l=2): Authentication-Method: Pre-shared key
  › IKE Attribute (t=11,l=2): Life-Type: Seconds
  › IKE Attribute (t=12,l=4): Life-Duration: 86400
```

Figure 4: Security Association payload

```
!IKE Phase 1 - Configure ISAKMP policy
crypto isakmp policy 10
hash sha
authentication pre-share
group 5
lifetime 86400
encryption aes 256
```

Figure 5: R1 ISAKMP policy configuration

The security association payload observed in Figure 4 contains the Life Duration equals to the lifetime we configured as we can see in figure 5.
After R2's answer, with the message accepting the ISAKMP policy we jump to the next step, Key exchange.

2.2) Step 2 **Key exchange**:
Once the IPSec peers have agreed upon the security parameters, they initiate the key exchange process to establish the shared secret key material that will be used for encryption and authentication within the IPSec SA. R1 and R2 in our exercise exchange Diffie-Hellman public keys using the agreed DH group. We can see in the next capture the key exchange payload containing the public key sent to R2 from R1.

2038	4419.958043	200.1.1.1	200.2.2.2	ISAKMP	210 Identity Protection (Main Mode)
2039	4420.002384	200.2.2.2	200.1.1.1	ISAKMP	150 Identity Protection (Main Mode)
2040	4420.027450	200.1.1.1	200.2.2.2	ISAKMP	414 Identity Protection (Main Mode)
2041	4420.073992	200.2.2.2	200.1.1.1	ISAKMP	434 Identity Protection (Main Mode)
2042	4420.189013	200.1.1.1	200.2.2.2	ISAKMP	790 Identity Protection (Main Mode)
2046	4422.002607	200.2.2.2	200.1.1.1	ISAKMP	758 Identity Protection (Main Mode)
2059	4423.724142	200.1.1.1	200.2.2.2	ISAKMP	230 Quick Mode
2060	4423.802991	200.2.2.2	200.1.1.1	ISAKMP	230 Quick Mode
2061	4423.857467	200.1.1.1	200.2.2.2	ISAKMP	102 Quick Mode

```
> Frame 2040: 414 bytes on wire (3312 bits), 414 bytes captured (3312 bits) on interface -, id 0
> Ethernet II, Src: 0c:66:f8:b1:00:00 (0c:66:f8:b1:00:00), Dst: ca:01:6b:72:00:08 (ca:01:6b:72:00:08)
> Internet Protocol Version 4, Src: 200.1.1.1, Dst: 200.2.2.2
> User Datagram Protocol, Src Port: 500, Dst Port: 500
v Internet Security Association and Key Management Protocol
    Initiator SPI: e0661c0a08219aae
    Responder SPI: a6d05fe19f0ab241
    Next payload: Key Exchange (4)
  > Version: 1.0
    Exchange type: Identity Protection (Main Mode) (2)
  > Flags: 0x00
    Message ID: 0x00000000
    Length: 372
  v Payload: Key Exchange (4)
      Next payload: Nonce (10)
      Reserved: 00
      Payload length: 196
      Key Exchange Data: 7abf99040cc2858238b23b6c03388b4e677d5f842088d1a247b37f48ec60c3c3841974fb…
  > Payload: Nonce (10)
  > Payload: Certificate Request (7)
  > Payload: Vendor ID (13) : RFC 3706 DPD (Dead Peer Detection)
  > Payload: Vendor ID (13) : Unknown Vendor ID
  > Payload: Vendor ID (13) : XAUTH
```

Figure 6: Key exchange Payload

2.3) Step 3 ISAKMP **Identity Protection**: The purpose of these messages is to exchange identity information between the peers and verify each other's identities using the pre-shared key configured on both routers. Figure 7 illustrates the command used to configure the pre-shared secret key and Figure 8 contains the packet regarding this last step of the IKE phase 1. So an ISAKMP tunnel was successfully made after this step and we can proceed to the next IKE phase.

```
crypto isakmp key saar address 200.2.2.2
```

Figure 7: R1 pre-shared key configuration

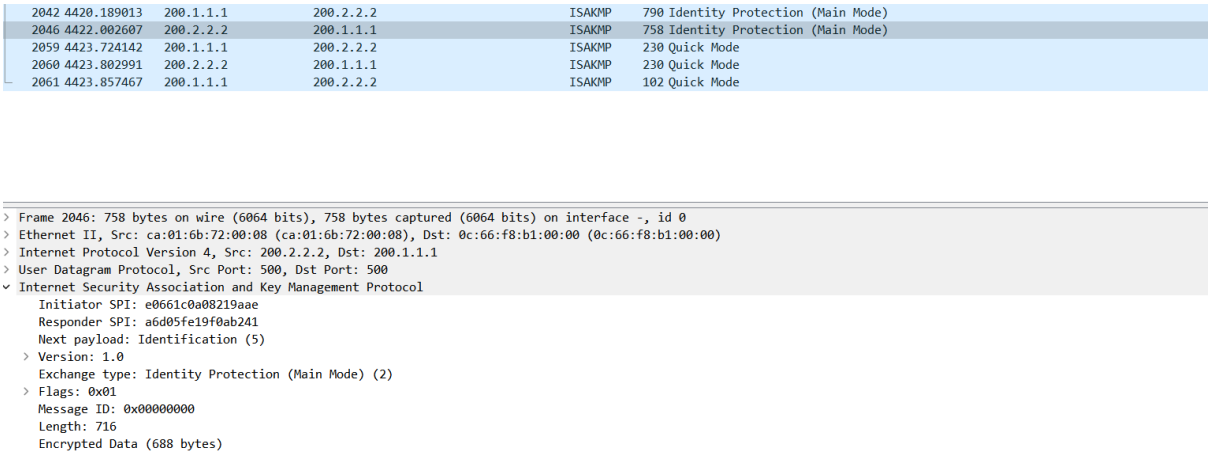


Figure 8: Identity protection

3) IKE (Internet Key Exchange) Phase 2: R1 and R2 negotiate IPsec SA policy over the secure tunnel created on IKE Phase 1 in order to build an IPSec tunnel. The next figure illustrates the first packet sent from R1 to R2.

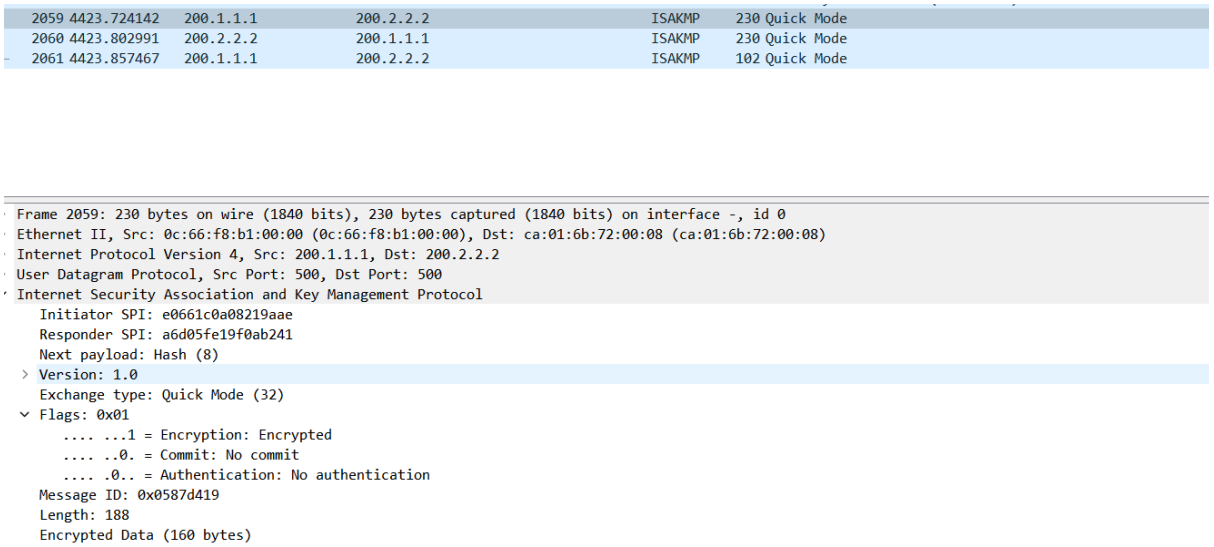


Figure 9: IPsec SA negotiation

TÉCNICO LISBOA

Encapsulating Security Payload

ESP is one of the two main protocols used in IPSec to provide secure communication over IP networks. ESP operates at the network layer and is responsible for providing authentication, integrity, anti-replay and encryption for the encapsulated IP packets.
After the IPSec tunnel was established, since we made a ping from PC1 to PC2 we could conclude that the number of ICMP messages received on both PCs correspond to the number of ESP packets we have in the following capture (Figure 10).

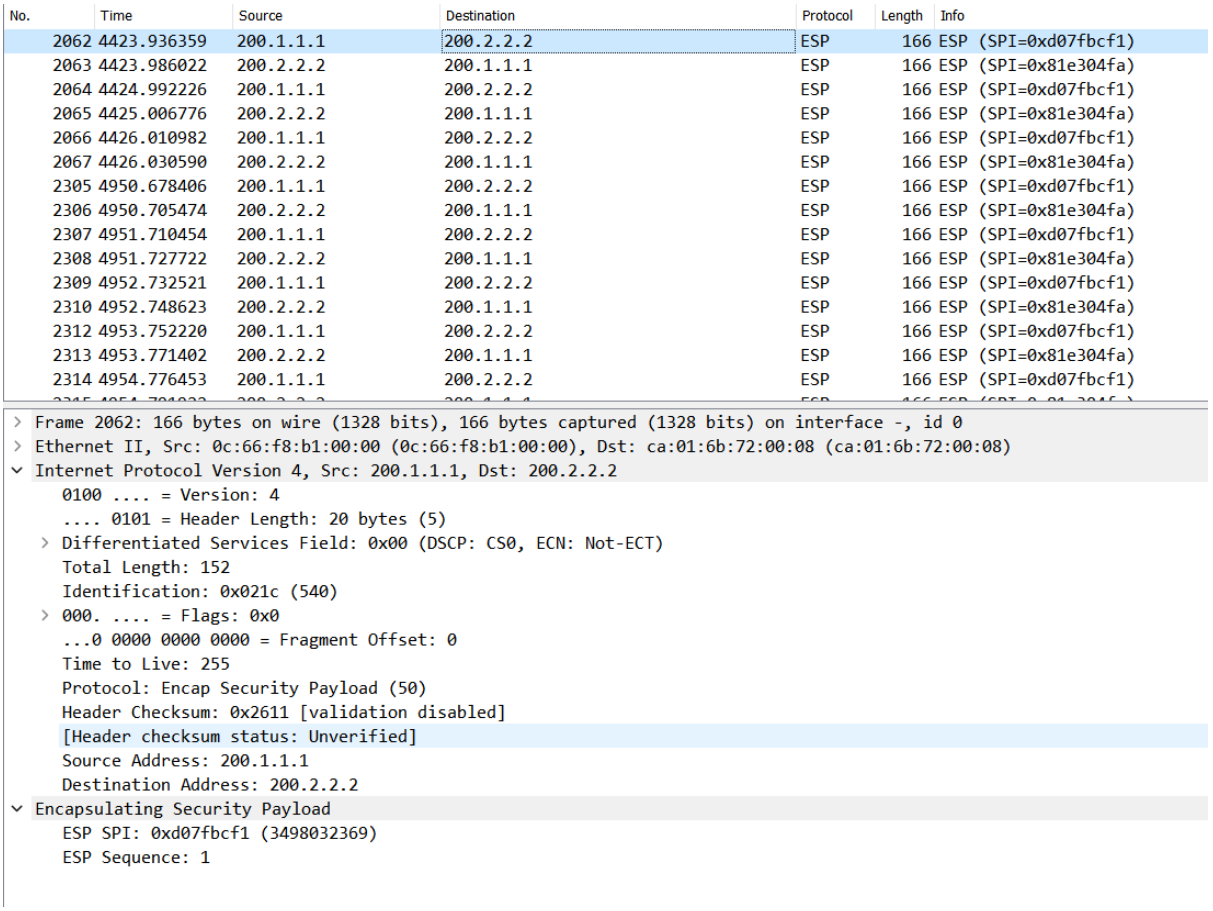


Figure 10: ESP payload

After observing the packet content illustrated in the previous figure we can conclude that the protocol type of ESP is identified by the value 50 in the IP protocol header field (IPv4). By observation of the IP source and destination we can conclude that the tunnel was successfully performed since the two IP addresses (source and destination) are corresponding to R1 and R2, where the tunnel was configured, both placed on different networks. The ESP header is placed immediately after the IP header and contains the necessary information for ESP processing. It includes fields such as Security Parameters Index (SPI) and Sequence Number. We can also observe that after making the ping between the two computers and after the IPSec tunnel was made, the last packet number of IKE phase 2 is 2061 (Figure 3) followed by the actual ESP packets transporting the ICMP messages with the number 2062 (Figure 10).

IPSec with digital certificates and certificate authority

Network structure

The network topology used in this exercise is the same as the previous section, but now R1 and R2 are IOSv reuters and RA is a 7200 router serving as the Certification Authority.

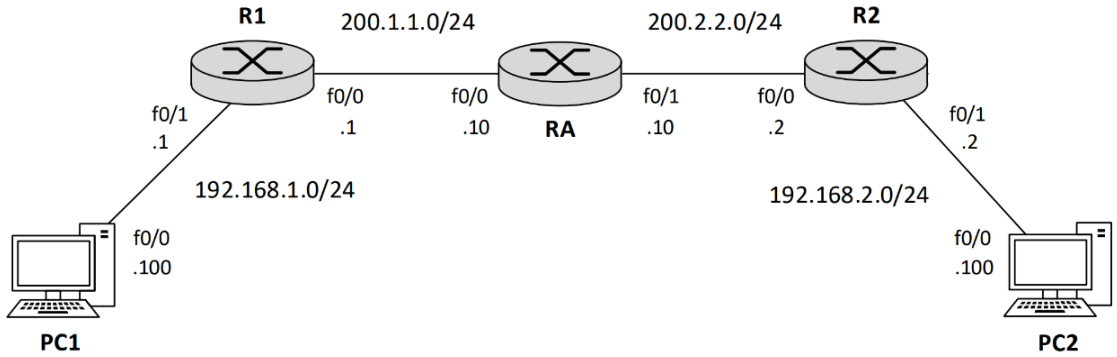


Figure 1: Network topology

The purpose of this exercise was to configure a Certification Authority (CA) and IPSec with authentication using digital signatures. First, we configured the CA and PKI server at router RA. We can see the information on the PKI server below after the implementation (Figure 1).

```
RA#show crypto pki server
Certificate Server myCA:
    Status: enabled
    State: enabled
    Server's configuration is locked  (enter "shut" to unlock it)
    Issuer name: cn=saarCA
    CA cert fingerprint: 9A966FA6 A66DB728 C078DBF0 79C4CB13
    Granting mode is: auto
    Last certificate issued serial number (hex): 1
    CA certificate expiration timer: 18:17:56 UTC Jun 6 2026
    CRL NextUpdate timer: 00:17:56 UTC Jun 8 2023
    Current primary storage dir: nvram:
    Database Level: Minimum - no cert data written to storage
RA#
```

Figure 2: RA PKI server

We then proceed to configure the PKI Clients, R1 and R2 to obtain a certificate from the CA. The R1 and R2 public keys are illustrated in the following figures, respectively.

```
Router#show crypto key mypub rsa
% Key pair was generated at: 18:22:38 UTC Jun 7 2023
Key name: Router
Key type: RSA KEYS
 Storage Device: not specified
 Usage: General Purpose Key
 Key is not exportable.
 Key Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00C0EF4F
  162C2680 F5E5F8B1 C07C0018 07A9DAE7 BFD1D0C5 46283461 257BFA47 63333135
  A008672B 43673488 4A1A6B51 DFA7F37E 2B7F5EBD EF4F121C E39B5F04 DC3306EA
  5507EE0A 364FC6CE B5D300D3 2B97383F CB870779 09236F0D ED13E92A DEE699D7
  F8829D50 C40306C2 544EC175 6E11E894 CE65BFB7 5805F982 3CFCB9E0 FD020301
  0001
% Key pair was generated at: 18:22:40 UTC Jun 7 2023
Key name: Router.server
Key type: RSA KEYS
Temporary key
 Usage: Encryption Key
 Key is not exportable.
 Key Data:
  307C300D 06092A86 4886F70D 01010105 00036B00 30680261 009AAD78 1130D909
  033AA5EA 1F704D72 8FF01629 9AB44D25 8CAE0EB7 D7A518A7 14234618 DD1C2DCB
  C88BACD5 3A40ADBD C3C6A69C 38E9B656 173175C8 25A18476 8F85DE07 515E211F
  A453270D 99983D0F 3F18E704 7A864E05 877909C4 81BA3D5C 79020301 0001
Router#
```

Figure 3: R1 public key

```
Router#show crypto key mypub rsa
% Key pair was generated at: 18:28:57 UTC Jun 7 2023
Key name: Router
Key type: RSA KEYS
 Storage Device: not specified
 Usage: General Purpose Key
 Key is not exportable.
 Key Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00BA9658
  A4515330 7032B3A5 85D6D9B0 CC1B070B 6018B149 E4A68234 3C905322 753A66BC
  A87366BD B91FC5FA 68C82AB2 29E79D2A 276E5840 777F4111 28F25EAC 1EF897AD
  19D01986 2ED40475 A71ADD9C 94E70A4C 9571E7FE CB6F695F 0D7A3537 F8DD30F2
  A3782970 47739399 1C2BAEED 410785FC 2AF06D65 B6997042 A606B3A8 2F020301
  0001
% Key pair was generated at: 18:28:59 UTC Jun 7 2023
Key name: Router.server
Key type: RSA KEYS
Temporary key
 Usage: Encryption Key
 Key is not exportable.
 Key Data:
  307C300D 06092A86 4886F70D 01010105 00036B00 30680261 008CF908 42A82494
  643B2FAD CFC17289 F0292641 EB07AA15 851045BF E5DD95B5 5F69B662 FA98CF08
  482336C2 76F10E17 2E18DD54 4F154F4E C7628F1D 0782E09C DA60B31C 4DAA7103
  C06C9E44 0A4B3BD4 DE07F5AB 6D5F6330 5CC4472B 71547D7F B1020301 0001
Router#
```

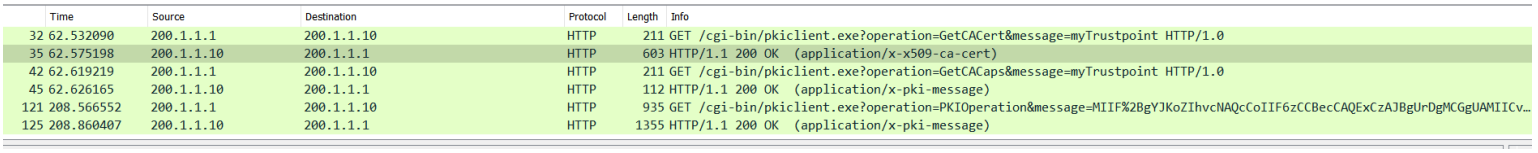
Figure 4: R2 public key

TÉCNICO LISBOA

SCEP

SCEP (Simple Certificate Enrollment Protocol) is a protocol used for enrolling and managing digital certificates in a PKI (Public Key Infrastructure) system, being the enrollment, the process of obtaining a certificate. In order to view the exchange of HTTP messages related to SCEP we installed a Wireshark probe between R1 and RA before starting the configuration. The next figure is the capture obtained for the SCEP-base enrollment. We will explain the role of every packet one by one. The exchange messages flow is the following:

1. The first message is the certificate request done by R1 to RA (GEtCACert operation), initiating the enrollment process by sending a Certificate Request message to the Certificate Authority (CA). This request includes the public key of R1 which will be used to generate the corresponding digital certificate. This Certificate Request message and the following messages are sent over HTTP.
2. The second message is the CA response containing the generated router's certificate signed with CA's private key. By using Wireshark we can extract the certificate, as figure 6 demonstrates.
3. The third message is the GETCACaps Operation sent from R1 to RA, this message requests the CA's (RA) list of capabilities. These capabilities include the supported cryptographic algorithms (e.g. AES, SHA-256) and other features provided by CA. This information helps the router determine the appropriate setting for the enrollment process.
4. The CA (RA) responds with the list of capabilities.
5. R1 sends an PKIOperation message to request a certificate while sending its public key. The router includes the necessary information for the CA to generate the certificate, such as the desired subject name, key type, and key size. The router's public key is also included to associate it with the generated certificate.
6. RA being the Certification Authority responds with the requested certificate along with other information such as the subject name, expiration date, and key usage.



	Time	Source	Destination	Protocol	Length	Info
32	62.532090	200.1.1.1	200.1.1.10	HTTP	211	GET /cgi-bin/pkiclient.exe?operation=GetCACert&message=myTrustpoint HTTP/1.0
35	62.575198	200.1.1.10	200.1.1.1	HTTP	603	HTTP/1.1 200 OK (application/x-x509-ca-cert)
42	62.619219	200.1.1.1	200.1.1.10	HTTP	211	GET /cgi-bin/pkiclient.exe?operation=GetCACaps&message=myTrustpoint HTTP/1.0
45	62.626165	200.1.1.10	200.1.1.1	HTTP	112	HTTP/1.1 200 OK (application/x-pki-message)
121	208.566552	200.1.1.1	200.1.1.10	HTTP	935	GET /cgi-bin/pkiclient.exe?operation=PKIOperation&message=MIIF%2BgYJKoZIhvcNAQcCoIIF6zCCBecCAQExCzAJBgUrDgMCGgUAMIICv…
125	208.860407	200.1.1.10	200.1.1.1	HTTP	1355	HTTP/1.1 200 OK (application/x-pki-message)

Figure 5: SCEP messages

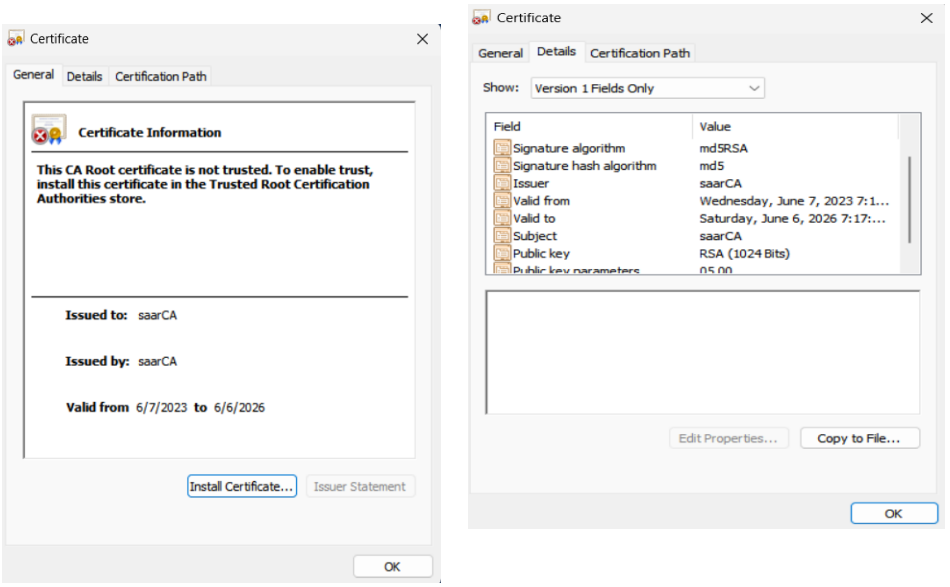


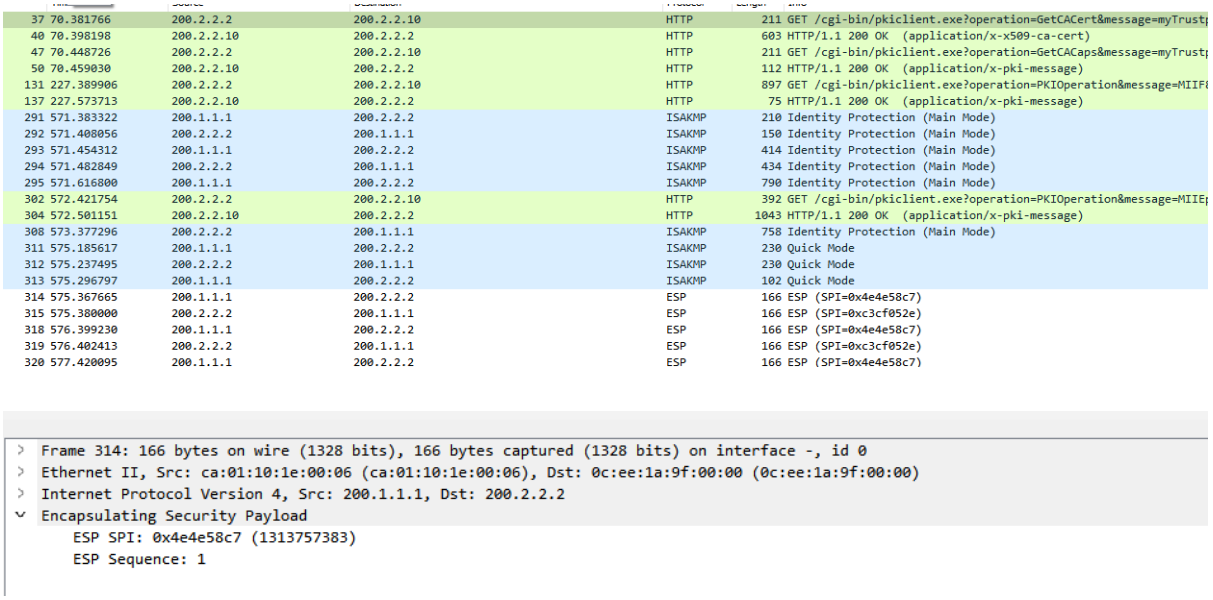
Figure 6: Certificate

After the SCEP-based enrollment process when we pinged from PC1 to PC2, since it is considered as an interesting traffic, R1 and R2 start to negotiate the ISAKMP policy and after verify each other's certificate to see if it is revoked by sending to RA (Certificate Authority) each other's public key, as we can see in green in the capture below. Certificates are digital documents that bind a public key to a specific entity and are issued by a trusted third-party called Certificate Authority, in our case, RA. Certificates take advantage of asymmetric cryptography as an authentication method, using the public key to decrypt the communication and the private key to encrypt, as we can observe in figure 9, the authentication method is done through RSA signatures. In comparison, Pre-shared keys instead use symmetric key cryptography, shared in advance, for both encryption and decryption.

3550	7740.510376	200.1.1.1	200.2.2.2	ISAKMP	210 Identity Protection (Main Mode)
3551	7740.545389	200.2.2.2	200.1.1.1	ISAKMP	150 Identity Protection (Main Mode)
3552	7740.574923	200.1.1.1	200.2.2.2	ISAKMP	414 Identity Protection (Main Mode)
3553	7740.608644	200.2.2.2	200.1.1.1	ISAKMP	434 Identity Protection (Main Mode)
3554	7740.734284	200.1.1.1	200.2.2.2	ISAKMP	790 Identity Protection (Main Mode)
3558	7742.579739	200.2.2.2	200.1.1.1	ISAKMP	758 Identity Protection (Main Mode)
3565	7743.424784	200.1.1.1	200.1.1.10	HTTP	402 GET /cgi-bin/pkiclient.exe?operation=PKIOperation&message=MIIEpQYJKoZIhvcNAQcCoIIEljCCBJI
3567	7743.514031	200.1.1.10	200.1.1.1	HTTP	1043 HTTP/1.1 200 OK (application/x-pki-message)
3571	7744.286898	200.1.1.1	200.2.2.2	ISAKMP	230 Quick Mode
3572	7744.376814	200.2.2.2	200.1.1.1	ISAKMP	230 Quick Mode
3573	7744.434940	200.1.1.1	200.2.2.2	ISAKMP	102 Quick Mode

Figure 7: SCEP HTTP and ISAKMP messages

After the ping, performed from PC1 to PC2, we can conclude that the communication is indeed ciphered, using ESP after the IPSec negotiation, demonstrated in the next figure.



37	70.381766	200.2.2.2	200.2.2.10	HTTP	211 GET /cgi-bin/pkiclient.exe?operation=GetCACert&message=myTrust
40	70.398198	200.2.2.10	200.2.2.2	HTTP	603 HTTP/1.1 200 OK (application/x-x509-ca-cert)
47	70.448726	200.2.2.2	200.2.2.10	HTTP	211 GET /cgi-bin/pkiclient.exe?operation=GetCACaps&message=myTrust
50	70.459030	200.2.2.10	200.2.2.2	HTTP	112 HTTP/1.1 200 OK (application/x-pki-message)
131	227.389906	200.2.2.2	200.2.2.10	HTTP	897 GET /cgi-bin/pkiclient.exe?operation=PKIOperation&message=MIIF
137	227.573713	200.2.2.10	200.2.2.2	HTTP	75 HTTP/1.1 200 OK (application/x-pki-message)
291	571.383322	200.1.1.1	200.2.2.2	ISAKMP	210 Identity Protection (Main Mode)
292	571.408056	200.2.2.2	200.1.1.1	ISAKMP	150 Identity Protection (Main Mode)
293	571.454312	200.1.1.1	200.2.2.2	ISAKMP	414 Identity Protection (Main Mode)
294	571.482849	200.2.2.2	200.1.1.1	ISAKMP	434 Identity Protection (Main Mode)
295	571.616800	200.1.1.1	200.2.2.2	ISAKMP	790 Identity Protection (Main Mode)
302	572.421754	200.2.2.2	200.2.2.10	HTTP	392 GET /cgi-bin/pkiclient.exe?operation=PKIOperation&message=MIIE
304	572.501151	200.2.2.10	200.2.2.2	HTTP	1043 HTTP/1.1 200 OK (application/x-pki-message)
308	573.377296	200.2.2.2	200.1.1.1	ISAKMP	758 Identity Protection (Main Mode)
311	575.185617	200.1.1.1	200.2.2.2	ISAKMP	230 Quick Mode
312	575.237495	200.2.2.2	200.1.1.1	ISAKMP	230 Quick Mode
313	575.296797	200.1.1.1	200.2.2.2	ISAKMP	102 Quick Mode
314	575.367665	200.1.1.1	200.2.2.2	ESP	166 ESP (SPI=0x4e4e58c7)
315	575.380000	200.2.2.2	200.1.1.1	ESP	166 ESP (SPI=0xc3cf052e)
318	576.399230	200.1.1.1	200.2.2.2	ESP	166 ESP (SPI=0x4e4e58c7)
319	576.402413	200.2.2.2	200.1.1.1	ESP	166 ESP (SPI=0xc3cf052e)
320	577.420095	200.1.1.1	200.2.2.2	ESP	166 ESP (SPI=0x4e4e58c7)

```
> Frame 314: 166 bytes on wire (1328 bits), 166 bytes captured (1328 bits) on interface -, id 0
> Ethernet II, Src: ca:01:10:1e:00:06 (ca:01:10:1e:00:06), Dst: 0c:ee:1a:9f:00:00 (0c:ee:1a:9f:00:00)
> Internet Protocol Version 4, Src: 200.1.1.1, Dst: 200.2.2.2
v Encapsulating Security Payload
      ESP SPI: 0x4e4e58c7 (1313757383)
      ESP Sequence: 1
```

Figure 8: Ping from PC1 to PC2

```
v Payload: Security Association (1)
    Next payload: Vendor ID (13)
    Reserved: 00
    Payload length: 60
    Domain of interpretation: IPSEC (1)
  > Situation: 00000001
  v Payload: Proposal (2) # 1
      Next payload: NONE / No Next Payload  (0)
      Reserved: 00
      Payload length: 48
      Proposal number: 1
      Protocol ID: ISAKMP (1)
      SPI Size: 0
      Proposal transforms: 1
    v Payload: Transform (3) # 1
        Next payload: NONE / No Next Payload  (0)
        Reserved: 00
        Payload length: 40
        Transform number: 1
        Transform ID: KEY_IKE (1)
        Reserved: 0000
      > IKE Attribute (t=1,l=2): Encryption-Algorithm: AES-CBC
      > IKE Attribute (t=14,l=2): Key-Length: 256
      > IKE Attribute (t=2,l=2): Hash-Algorithm: SHA
      > IKE Attribute (t=4,l=2): Group-Description: 1536 bit MODP group
      > IKE Attribute (t=3,l=2): Authentication-Method: RSA signatures
      > IKE Attribute (t=11,l=2): Life-Type: Seconds
      > IKE Attribute (t=12,l=4): Life-Duration: 86400
```

Figure 9: Security Association Payload

When the two routers try to communicate with each other, since we are in IPSec context, they exchange public keys in order to verify each other's authenticity and certificate validity, and establish the tunnel for the communication. The next two figures demonstrate that they have each other's public key after the ISAKMP SA agreement. We also included a

screenshot of R2's serial number in order to explain the hostname attached to the public key of R2 at R1's terminal in figure 11.

```
Router#show crypto key pubkey-chain rsa name Router
 Key name: Router
 Subject name(X.500 DN name): hostname=Router+serialNumber=9RY0UMKKG3VESOTMNC43Q
 Key id: 11
 Serial number: 02
 Usage: Signature Key
 Source: Certificate
 Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00C0EF4F
  162C2680 F5E5F8B1 C07C0018 07A9DAE7 BFD1D0C5 46283461 257BFA47 63333135
  A008672B 43673488 4A1A6B51 DFA7F37E 2B7F5EBD EF4F121C E39B5F04 DC3306EA
  5507EE0A 364FC6CE B5D300D3 2B97383F CB870779 09236F0D ED13E92A DEE699D7
  F8829D50 C40306C2 544EC175 6E11E894 CE65BFB7 5805F982 3CFCB9E0 FD020301
  0001
```

Figure 10: show crypto key pubkey-chain rsa name R1 at R2 router terminal

```
Router#show crypto key pubkey-chain rsa name Router
 Key name: Router
 Subject name(X.500 DN name): hostname=Router+serialNumber=9P9JGDIUSOTYMJQN72BFA
 Key id: 11
 Serial number: 03
 Usage: Signature Key
 Source: Certificate
 Data:
  30819F30 0D06092A 864886F7 0D010101 05000381 8D003081 89028181 00BA9658
  A4515330 7032B3A5 85D6D9B0 CC1B070B 6018B149 E4A68234 3C905322 753A66BC
  A87366BD B91FC5FA 68C82AB2 29E79D2A 276E5840 777F4111 28F25EAC 1EF897AD
  19D01986 2ED40475 A71ADD9C 94E70A4C 9571E7FE CB6F695F 0D7A3537 F8DD30F2
  A3782970 47739399 1C2BAEED 410785FC 2AF06D65 B6997042 A606B3A8 2F020301
  0001
```

Figure 11: show crypto key pubkey-chain rsa name R2 at R1 router terminal

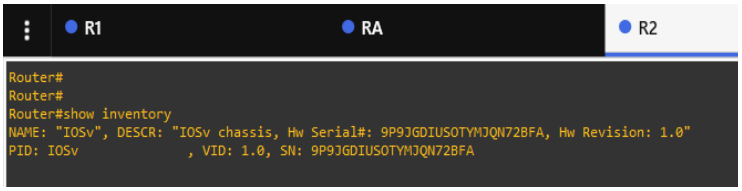


Figure 12: Router 2 serial key

Revocation

When a certificate authority(CA) determines that a certificate is no longer valid or trustworthy, it revokes the certificate. This can occur due to various reasons, such as the compromise of the private key, expiration or the user's request. The CA maintains a Certificate Revocation List (CRL) which is a digitally-signed document that contains a list of revoked certificates. The CRL is periodically updated and made available to relying parties. Before establishing communication with a certificate-based entity (such as a server or client), the relying party

performs certificate validation. This involves verifying the digital signature on the certificate, checking the certificate's expiration date, and ensuring that it has not been revoked. If the certificate is found in the CRL, indicating its revocation, the relying party refuses to establish communication or trust the entity holding the revoked certificate. The communication is prevented to protect against potential security risks associated with compromised or invalidated certificates. The figure 13 prove we cannot communicate with each other after the certificate revocation.

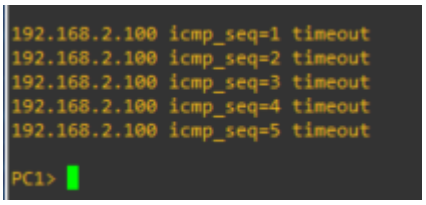


Figure 13: ping after revoke

GRE over IPSec

Network Structure

In this exercise we configured the network topology illustrated in the figure below to support GRE over IPSec (AH protocol in tunnel mode).

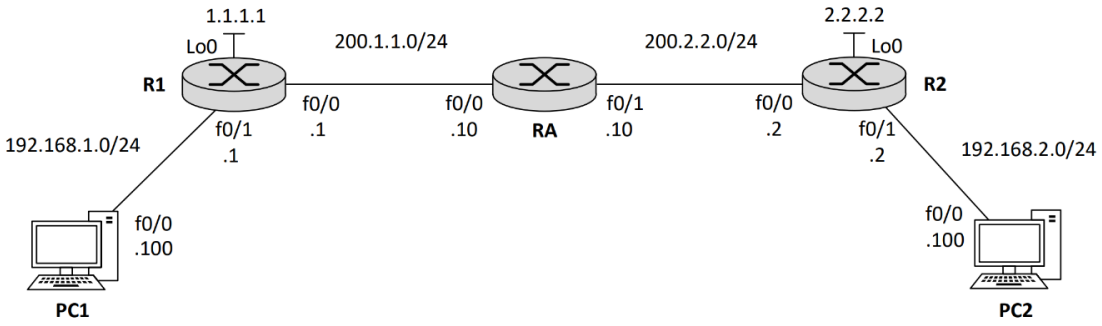


Figure 1: Network topology

GRE

GRE stands for Generic Routing Encapsulation. It is a protocol used to encapsulate a wide range of network layer protocols within IP packets. GRE allows the transmission of data packets that may not be compatible with the underlying network infrastructure by encapsulating them within IP packets.

GRE adds an additional header to the original packet, allowing it to be transmitted over an IP network. This additional header includes information such as the source and destination IP addresses, protocol type, and other fields. GRE facilitates the creation of virtual private networks (VPNs) by encapsulating packets within IP packets, creating a tunnel through which the encapsulated packets can be transmitted. and is often used in combination with routing protocols to establish connections between networks or to connect remote networks over an IP infrastructure. It allows packets to traverse through intermediate networks by encapsulating them within IP packets. GRE is transparent to the underlying network infrastructure, meaning that routers along the transmission path treat the encapsulated packets as regular IP packets and forward them accordingly. GRE is also a stateless protocol, meaning it does not maintain any specific connection state information. Each encapsulated packet is treated independently, without requiring any previous context or connection setup.
The encapsulation process of GRE over IPSec involves adding additional headers to the original packet to enable transmission over an IP network while ensuring security.

Mode

The main difference between tunnel and transport modes in IPSec lies in the scope of protection provided by IPSec.
Tunnel Mode:
In tunnel mode, the entire original IP packet, including the original IP header, is encapsulated within a new IP packet. This new packet is then protected by IPSec using AH. The outer IP header is added with new source and destination IP addresses representing the IPSec gateways, as we could observe in both packet contents below.

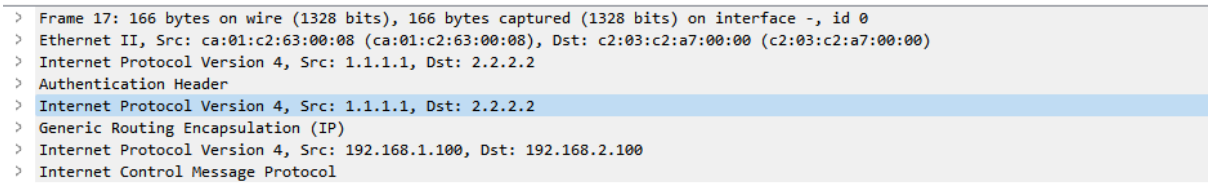


Figure 2: ICMP message tunnel mode

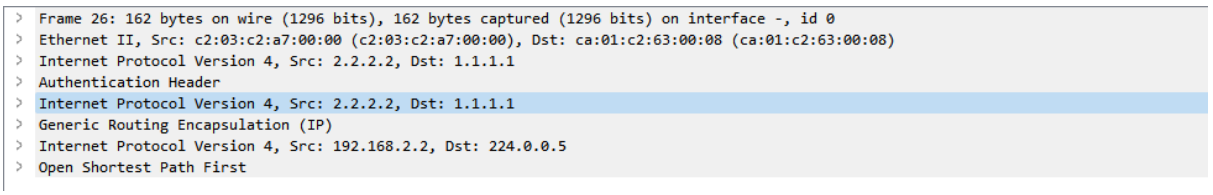


Figure 3: OSPF message tunnel mode

Transport Mode:
In transport mode, IPSec only protects the payload of the IP packet. The original IP header remains intact, and only the payload, including the GRE packet, is encapsulated and

protected by IPSec with AH. The next figure illustrates the packet content of the ICMP message from a ping performed between the two computers in transport mode.

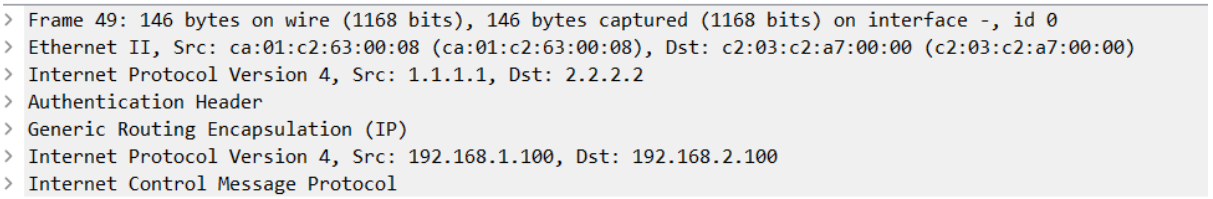


Figure 4: ICMP message Transport mode

In summary, tunnel mode is generally the more appropriate choice for GRE over IPsec due to its encapsulation and protection of the entire GRE packet since it provides stronger security for communications between two private networks over a public network. Transport mode, on the other hand, may have limited use cases where only the payload encryption and authentication are required within the same trusted network segment or device.

ESP vs AH

AH (Authentication Header) and ESP (Encapsulating Security Payload) are two protocols used in the IPsec (Internet Protocol Security) suite to provide security services for IP packets. While both AH and ESP serve the goal of ensuring secure communication, they differ in terms of the specific security services they offer. The main difference resides in:

- **Integrity and Authentication:** AH provides both data integrity and authentication for the entire IP packet, including the IP header and payload. ESP, on the other hand, offers these services optionally and specifically for the payload (with the addition of confidentiality services). Figure 6 represents the packet content of an ICMP message with AH.
- **Confidentiality:** ESP provides encryption services for the payload, ensuring its confidentiality. AH does not offer encryption and focuses solely on integrity and authentication. The next figure represents an ESP packet protecting a ICMP message from a ping performed between the two computers.

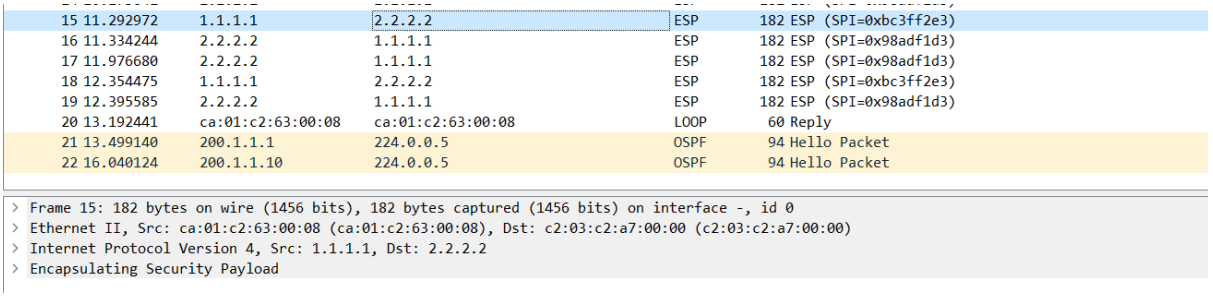


Figure 5: ESP message

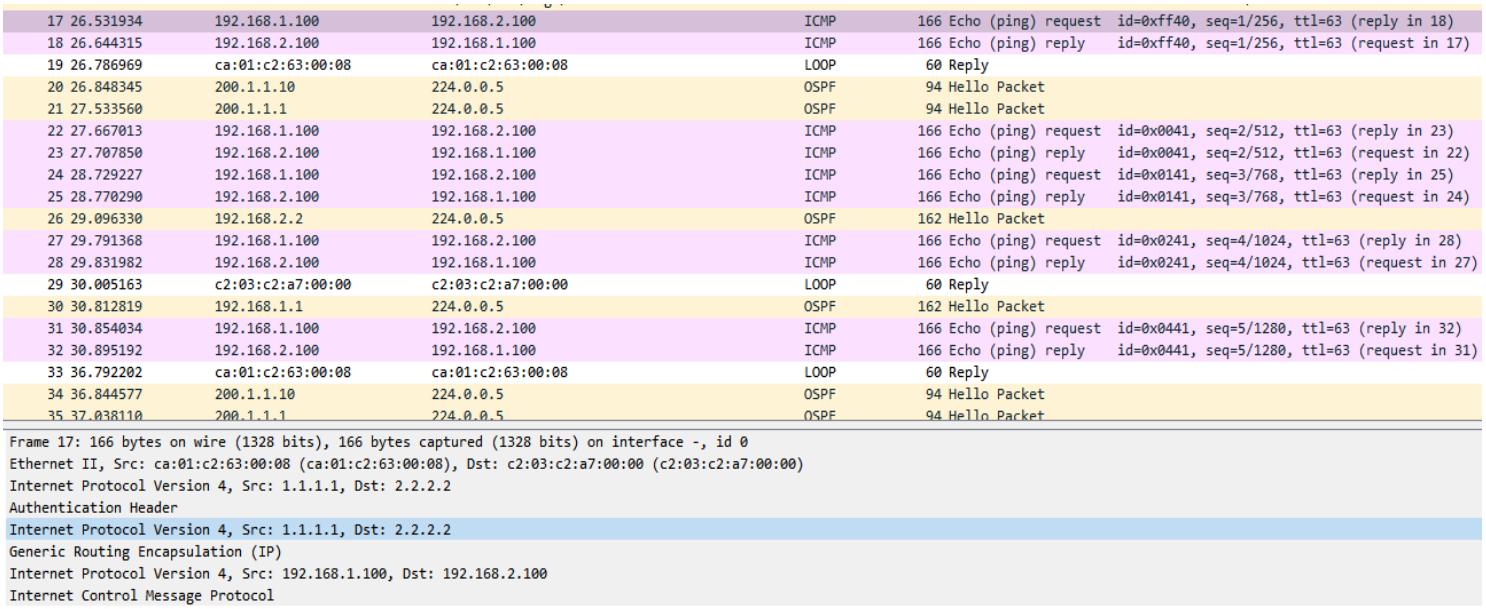


Figure 6: ICMP message with AH

OSPF

As we explained before, GRE is often combined with routing protocols in order to establish communications between private networks. We configured for the public networks the OSPF process 1 and for the private networks at R2 and R1 the OSPF process 2. We configured two OSPF processes to provide network segmentation and protection since the OSPF Link State Database is not shared between processes and each process maintains its database. The main purpose of this configuration is to isolate and protect the internal networks and to not advertise private over public networks. The OSPF process 2, represents the overlay, contains the private networks and is only advertised between R1 and R2 through the tunnel. The OSPF process 1 represents the underlay and contains the entire public network. The next capture illustrates both OSPF processes running at R1 perspective and figure 8 represents OSPF 1 and OSPF 2 Link State Database at R1 terminal. As we can see, the OSPF process 2 only has the R1 and R2 router-id differing from OSPF process 1 that has the entire public network router-ids.

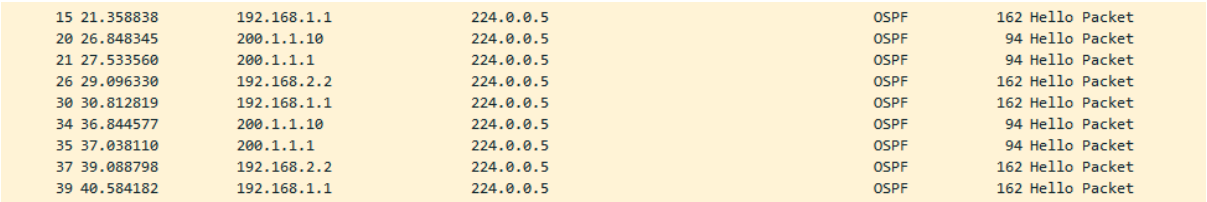


Figure 7: OSPF messages

```
      2.0.0.0/32 is subnetted, 1 subnets
O        2.2.2.2 [110/12] via 200.1.1.10, 00:22:20, FastEthernet0/0
O     192.168.2.0/24 [110/1001] via 192.168.2.2, 00:22:03, Tunnel0
O     200.2.2.0/24 [110/11] via 200.1.1.10, 00:22:20, FastEthernet0/0
R1#show ip ospf database

            OSPF Router with ID (10.10.10.10) (Process ID 2)

                Router Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum Link count
10.10.10.10     10.10.10.10     1372        0x80000002 0x009292 2
20.20.20.20     20.20.20.20     1373        0x80000002 0x006992 2

            OSPF Router with ID (10.10.10.200) (Process ID 1)

                Router Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum Link count
10.10.10.200    10.10.10.200    1390        0x80000002 0x00E8C5 2
20.20.20.200    20.20.20.200    1391        0x80000002 0x002544 2
100.100.100.200 100.100.100.200 1391        0x80000003 0x008843 2

                Net Link States (Area 0)

Link ID         ADV Router      Age         Seq#       Checksum
200.1.1.10      100.100.100.200 1391        0x80000001 0x007919
200.2.2.10      100.100.100.200 1391        0x80000001 0x00CBA6
R1#
```

Figure 8: OSPF 1 and OSPF 2 LSDB at R1

IPSec with IKEv2

Network structure

For this exercise we used the exact same network as the one in the first exercise of this lab (see Figure 1 from the IPSec using ESP in tunnel mode exercise) with one key difference: we've configured IKEv2 instead of IKEv1 on both routers R1 and R2.

In order to get connectivity between PC1 and PC2, we created a tunnel between R1 and R2 which encrypts all traffic that goes between private networks. Said connectivity was later tested using the **ping** command (see Figure 1).

1	0.000000	0c:e3:8a:00:00:00	0c:e3:8a:00:00:00	LOOP	60 Reply
2	2.287784	200.1.1.10	224.0.0.5	OSPF	94 Hello Packet
3	3.254186	c2:01:05:be:00:00	c2:01:05:be:00:00	LOOP	60 Reply
4	8.981954	200.1.1.1	200.2.2.2	ESP	166 ESP (SPI=0x635b7795)
5	9.007800	200.2.2.2	200.1.1.1	ESP	166 ESP (SPI=0xd54fc7d7)
6	9.923350	200.1.1.1	224.0.0.5	OSPF	94 Hello Packet
7	10.019315	200.1.1.1	200.2.2.2	ESP	166 ESP (SPI=0x635b7795)
8	10.046412	200.2.2.2	200.1.1.1	ESP	166 ESP (SPI=0xd54fc7d7)
9	11.051848	200.1.1.1	200.2.2.2	ESP	166 ESP (SPI=0x635b7795)
10	11.064242	200.2.2.2	200.1.1.1	ESP	166 ESP (SPI=0xd54fc7d7)
11	12.073538	200.1.1.1	200.2.2.2	ESP	166 ESP (SPI=0x635b7795)
12	12.091714	200.2.2.2	200.1.1.1	ESP	166 ESP (SPI=0xd54fc7d7)
13	12.293708	200.1.1.10	224.0.0.5	OSPF	94 Hello Packet
14	13.100793	200.1.1.1	200.2.2.2	ESP	166 ESP (SPI=0x635b7795)
15	13.141449	200.2.2.2	200.1.1.1	ESP	166 ESP (SPI=0xd54fc7d7)
16	13.245685	c2:01:05:be:00:00	c2:01:05:be:00:00	LOOP	60 Reply
17	13.531590	0c:e3:8a:00:00:00	0c:e3:8a:00:00:00	LOOP	60 Reply

Figure 1: Initial ping between PC1 and PC2

IKE Messages

When a new SA is established, there are certain messages that are exchanged between the tunnel's endpoints, as seen in Figure 2. In order to establish the SA the routers exchange "INFORMATIONAL" packets, followed by "IKE_SA_INIT" (see Figures 3 and 4) and "IKE_AUTH" packets (see Figures 5 and 6).

11	22.807255	200.2.2.2	200.1.1.1	ISAKMP	154 INFORMATIONAL MID=00 Responder Request
12	22.829237	200.1.1.1	200.2.2.2	ISAKMP	138 INFORMATIONAL MID=00 Initiator Response
21	44.306882	200.2.2.2	200.1.1.1	ISAKMP	816 IKE_SA_INIT MID=00 Initiator Request
22	44.464604	200.1.1.1	200.2.2.2	ISAKMP	816 IKE_SA_INIT MID=00 Responder Response
23	44.670846	200.2.2.2	200.1.1.1	ISAKMP	666 IKE_AUTH MID=01 Initiator Request
24	44.786351	200.1.1.1	200.2.2.2	ISAKMP	362 IKE_AUTH MID=01 Responder Response

Figure 2: IKEv2 messages on tunnel creation

```
▾ Internet Security Association and Key Management Protocol
    Initiator SPI: f1ca822a82ae8369
    Responder SPI: 0000000000000000
    Next payload: Security Association (33)
  › Version: 2.0
    Exchange type: IKE_SA_INIT (34)
  › Flags: 0x08 (Initiator, No higher version, Request)
    Message ID: 0x00000000
    Length: 774
  › Payload: Security Association (33)
  › Payload: Key Exchange (34)
  › Payload: Nonce (40)
  › Payload: Vendor ID (43) : Cisco Delete Reason Supported
  › Payload: Vendor ID (43) : Cisco VPN Revision 2
  › Payload: Vendor ID (43) : Cisco Dynamic Route Supported
  › Payload: Vendor ID (43) : Cisco FlexVPN Supported
  › Payload: Notify (41) - NAT_DETECTION_SOURCE_IP
  › Payload: Notify (41) - NAT_DETECTION_DESTINATION_IP
```

Figure 3: IKE_SA_INIT request packet

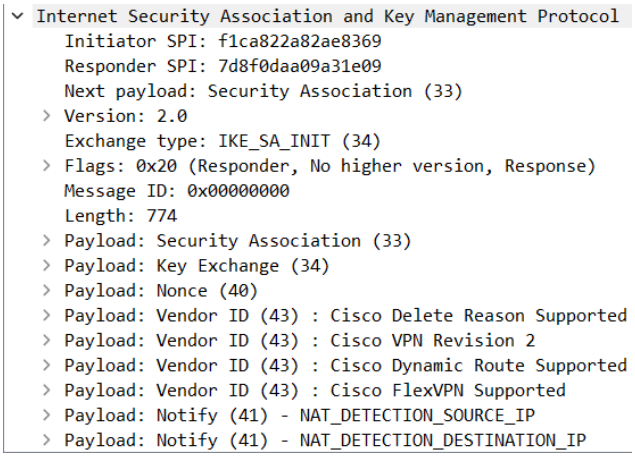


Figure 4: IKE_SA_INIT response packet

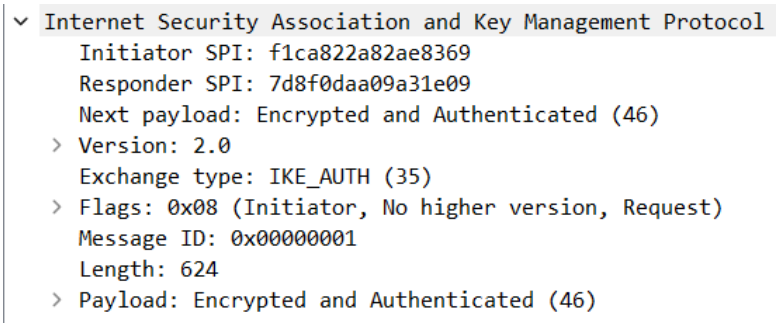


Figure 5: IKE_AUTH request packet

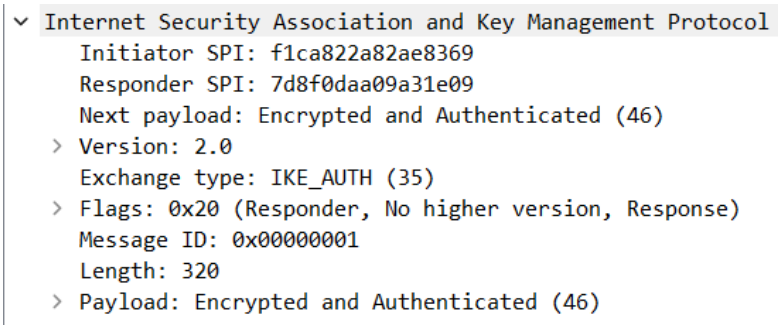


Figure 6: IKE_AUTH response packet

There are some fields that are particularly noteworthy in the IKE_SA_INIT messages, such as the “Initiator SPI” and “Responder SPI” parameters that are used as identifiers, the “Payload: Security Association” parameter that contains the cryptographic algorithms that are being proposed (“Payload: Proposal” parameter in Figure 7), the “Payload: Key Exchange” parameter which contains the public Diffie-Hellman value (Figure 8), and the “Payload: Nonce” parameter which contains - as the name entails - the value for the nonce (Figure 9).

This last parameter contains a random or pseudo-random value that is used to make the message unique, with the ultimate goal of preventing replay attacks by malicious parties.

As for the IKE_AUTH messages most of the content is encrypted. The most important part that we can observe without decrypting the packets are the SPIs for the initiator and responder, which are the same as in the IKE_SA_INIT messages.

```
▾ Payload: Proposal (2) # 1
    Next payload: NONE / No Next Payload  (0)
    Reserved: 00
    Payload length: 44
    Proposal number: 1
    Protocol ID: IKE (1)
    SPI Size: 0
    Proposal transforms: 4
  ▾ Payload: Transform (3)
      Next payload: Transform (3)
      Reserved: 00
      Payload length: 12
      Transform Type: Encryption Algorithm (ENCR) (1)
      Reserved: 00
      Transform ID (ENCR): ENCR_AES_CBC (12)
    › Transform Attribute (t=14,l=2): Key Length: 256
  ▾ Payload: Transform (3)
      Next payload: Transform (3)
      Reserved: 00
      Payload length: 8
      Transform Type: Pseudo-random Function (PRF) (2)
      Reserved: 00
      Transform ID (PRF): PRF_HMAC_SHA2_512 (7)
  ▾ Payload: Transform (3)
      Next payload: Transform (3)
      Reserved: 00
      Payload length: 8
      Transform Type: Integrity Algorithm (INTEG) (3)
      Reserved: 00
      Transform ID (INTEG): AUTH_HMAC_SHA2_512_256 (14)
  ▾ Payload: Transform (3)
      Next payload: NONE / No Next Payload  (0)
      Reserved: 00
      Payload length: 8
      Transform Type: Diffie-Hellman Group (D-H) (4)
      Reserved: 00
      Transform ID (D-H): 4096 bit MODP group (16)
```

Figure 7: Cryptographic algorithms proposal

```
▾ Payload: Key Exchange (34)
    Next payload: Nonce (40)
    0... .... = Critical Bit: Not critical
    .000 0000 = Reserved: 0x00
    Payload length: 520
    DH Group #: 4096 bit MODP group (16)
    Reserved: 0000
    Key Exchange Data: d02e68d841420b006f2c2ae937f39a1ff503c758164e58bc8236503102860ca2f1966917…
```

Figure 8: Public Diffie-Hellman value

```
Payload: Nonce (40)
    Next payload: Vendor ID (43)
    0... .... = Critical Bit: Not critical
    .000 0000 = Reserved: 0x00
    Payload length: 36
    Nonce DATA: 6faf208ea27e8964967f7569b7fade38258906bae9c770d8880dfcff8ba771c5
```

Figure 9: Nonce value

Rekeying

In order to better study the rekeying process for both IKE and IPSec, we forced a shutdown on one of the tunnel's endpoint interfaces and cleared the IKE and IPSec SAs on the routers. When restarting the tunnel connection, instead of having a IKE_SA_INIT packet exchange, we instead only have an exchange of INFORMATIONAL packets (same first two packets as in Figure 2). After a while we got packet flows for the IKE and IPSec SAs (Figures 10 and 11). Both of these flows start with an exchange of CREATE_CHILD_SA packets, which are encrypted (Figures 12 and 13). The biggest difference between IKE and IPSec in these packets is that the IKE rekeying is started by the responder, while the IPSec rekeying is started by the initiator.

```
167 332.540581   200.2.2.2   200.1.1.1   ISAKMP   762 CREATE_CHILD_SA MID=00 Responder Request
168 332.673192   200.1.1.1   200.2.2.2   ISAKMP   762 CREATE_CHILD_SA MID=00 Initiator Response
169 332.822345   200.2.2.2   200.1.1.1   ISAKMP   138 INFORMATIONAL MID=01 Responder Request
170 332.843698   200.1.1.1   200.2.2.2   ISAKMP   138 INFORMATIONAL MID=01 Initiator Response
```

Figure 10: IKE rekeying packet exchange

```
Internet Security Association and Key Management Protocol
    Initiator SPI: b981fb8f909aee6c
    Responder SPI: 3eee19abbcb57d50
    Next payload: Encrypted and Authenticated (46)
  > Version: 2.0
    Exchange type: CREATE_CHILD_SA (36)
  > Flags: 0x00 (Responder, No higher version, Request)
    Message ID: 0x00000000
    Length: 720
  > Payload: Encrypted and Authenticated (46)
```

Figure 11: IKE CREATE_CHILD_SA packet detail

```
▾ Internet Security Association and Key Management Protocol
      Initiator SPI: b981fb8f909aee6c
      Responder SPI: 0000000000000000

▾ Internet Security Association and Key Management Protocol
      Initiator SPI: b981fb8f909aee6c
      Responder SPI: 3eee19abbcb57d50
```

Figure 12: IKE_SA_INIT request and response SPIs after rekeying

```
Router#show crypto ikev2 sa detailed
 IPv4 Crypto IKEv2  SA

Tunnel-id Local                 Remote                fvrf/ivrf            Status
1         200.2.2.2/500         200.1.1.1/500         none/none            READY
      Encr: AES-CBC, keysize: 256, PRF: SHA512, Hash: SHA512, DH Grp:16, Auth sign: PSK, Auth verify: PSK
      Life/Active Time: 180/79 sec
      CE id: 1003, Session-id: 2
      Status Description: Negotiation done
      Local spi: 4B71FB6FF007B6D0       Remote spi: 518F70B07CC85360
      Local id: 200.2.2.2
      Remote id: 200.1.1.1
      Local req msg id:  2              Remote req msg id:  0
      Local next msg id: 2              Remote next msg id: 0
      Local req queued:  2              Remote req queued:  0
      Local window:      5              Remote window:      5
      DPD configured for 0 seconds, retry 0
      Fragmentation not  configured.
      Dynamic Route Update: disabled
      Extended Authentication not configured.
      NAT-T is not detected
      Cisco Trust Security SGT is disabled
      Initiator of SA : Yes

 IPv6 Crypto IKEv2  SA

Router#
```

Figure 13: IKEv2 SA detailed information

```
136 258.110501    200.1.1.1           200.2.2.2           ISAKMP    266 CREATE_CHILD_SA MID=02 Initiator Request
137 258.188593    200.2.2.2           200.1.1.1           ISAKMP    266 CREATE_CHILD_SA MID=02 Responder Response
138 258.268072    200.1.1.1           200.2.2.2           ISAKMP    138 INFORMATIONAL MID=03 Initiator Request
139 258.333063    200.2.2.2           200.1.1.1           ISAKMP    138 INFORMATIONAL MID=03 Responder Response
```

Figure 14: IPSec rekeying packet exchange

```
▾ Internet Security Association and Key Management Protocol
    Initiator SPI: b981fb8f909aee6c
    Responder SPI: 3eee19abbcb57d50
    Next payload: Encrypted and Authenticated (46)
  › Version: 2.0
    Exchange type: CREATE_CHILD_SA (36)
  › Flags: 0x08 (Initiator, No higher version, Request)
    Message ID: 0x00000002
    Length: 224
  › Payload: Encrypted and Authenticated (46)
```

Figure 15: IPSec CREATE_CHILD_SA packet detail

```
Router#show crypto ipsec sa

interface: Tunnel0
    Crypto map tag: Tunnel0-head-0, local addr 200.1.1.1

   protected vrf: (none)
   local  ident (addr/mask/prot/port): (0.0.0.0/0.0.0.0/0/0)
   remote ident (addr/mask/prot/port): (0.0.0.0/0.0.0.0/0/0)
   current_peer 200.2.2.2 port 500
     PERMIT, flags={origin_is_acl,}
    #pkts encaps: 3, #pkts encrypt: 3, #pkts digest: 3
    #pkts decaps: 3, #pkts decrypt: 3, #pkts verify: 3
    #pkts compressed: 0, #pkts decompressed: 0
    #pkts not compressed: 0, #pkts compr. failed: 0
    #pkts not decompressed: 0, #pkts decompress failed: 0
    #send errors 0, #recv errors 0

     local crypto endpt.: 200.1.1.1, remote crypto endpt.: 200.2.2.2
     plaintext mtu 1438, path mtu 1500, ip mtu 1500, ip mtu idb GigabitEthernet0/0
     current outbound spi: 0xD498516A(3566752106)
     PFS (Y/N): N, DH group: none

     inbound esp sas:
      spi: 0x12538D30(307465520)
        transform: esp-aes esp-sha-hmac ,
        in use settings ={Tunnel, }
        conn id: 10, flow_id: SW:10, sibling_flags 80000040, crypto map: Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4322501/71)
        IV size: 16 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)

     inbound ah sas:

     inbound pcp sas:

     outbound esp sas:
      spi: 0xD498516A(3566752106)
        transform: esp-aes esp-sha-hmac ,
        in use settings ={Tunnel, }
        conn id: 9, flow_id: SW:9, sibling_flags 80000040, crypto map: Tunnel0-head-0
        sa timing: remaining key lifetime (k/sec): (4322501/71)
        IV size: 16 bytes
        replay detection support: Y
        Status: ACTIVE(ACTIVE)

     outbound ah sas:

     outbound pcp sas:
Router#
```

Figure 16: IPSec SA detailed information

DMVPN Phase 3

Network Structure

For this exercise we have created a network with 4 routers and 3 PCs (Figure 1).

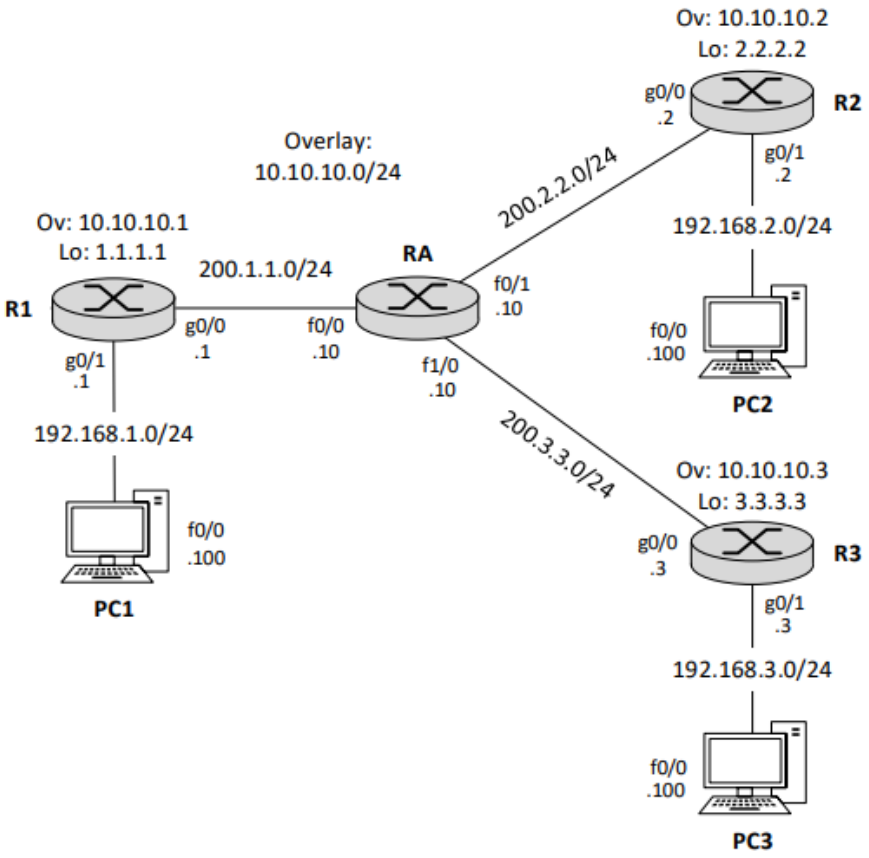


Figure 1: Network structure and IP distribution

In this exercise the goal is to configure a DMVPN Phase 3, this one is built on Phase 2's features optimizing it, improving the spoke-to-spoke communications (Figure 2), it also uses NHRP to resolve the physical interface of the next hop, enhancing the use of NHRP redirect and also shortcut capabilities, with the hub providing information of the spoke's destination to the spoke's source only once (figure 3).

```
> Frame 233: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface -, id 0
> Ethernet II, Src: 0c:b0:e8:81:00:00 (0c:b0:e8:81:00:00), Dst: c2:01:06:bb:00:01 (c2:01:06:bb:00:01)
> Internet Protocol Version 4, Src: 2.2.2.2, Dst: 3.3.3.3
> Generic Routing Encapsulation (IP)
> Internet Protocol Version 4, Src: 192.168.2.100, Dst: 192.168.3.100
> Internet Control Message Protocol
```

Figure 2: ICMP packet request from R2 directly to R3

```
> Frame 269: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface -, id 0
> Ethernet II, Src: 0c:ab:aa:7a:00:00 (0c:ab:aa:7a:00:00), Dst: c2:01:06:bb:00:00 (c2:01:06:bb:00:00)
> Internet Protocol Version 4, Src: 1.1.1.1, Dst: 2.2.2.2
> Generic Routing Encapsulation (NHRP)
v Next Hop Resolution Protocol (NHRP Traffic Indication)
  > NHRP Fixed Header
  v NHRP Mandatory Part
      Source Protocol Len: 4
      Destination Protocol Len: 4
      Source NBMA Address: 1.1.1.1
      Source Protocol Address: 10.10.10.1
      Destination Protocol Address: 192.168.2.100
  v Packet Causing Indication
    > Internet Protocol Version 4, Src: 192.168.2.100, Dst: 192.168.3.100
    > Internet Control Message Protocol
    > HiPerConTracer Trace Service
  > Forward Transit NHS Record Extension
  > Reverse Transit NHS Record Extension
  > Cisco NAT Address Extension
  > End of Extension
```

Figure 3: R1 sends NHRP redirect to R2

All phases use NHRP to establish and manage tunnels, but they have their differences. Phase 1 being the simplest utilizes a hub-to-spoke communication, this is improved in Phase 2 becoming spoke-to-spoke with the hub redirecting the communication, Phase 3 like is said before optimizes Phase 2.

Phase 3 Communication

When we first setup DMVPN Phase 3 the routing tables for the 3 Routers (R1, R2 and R3) is not yet complete (figure 4) and that also goes for the NHRP cache of R2 and R3 (Figures 5 and 6)

```
      1.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        1.1.1.0/24 is directly connected, Loopback0
L        1.1.1.1/32 is directly connected, Loopback0
      2.0.0.0/32 is subnetted, 1 subnets
O        2.2.2.2 [110/12] via 200.1.1.10, 00:00:45, GigabitEthernet0/0
      3.0.0.0/32 is subnetted, 1 subnets
O        3.3.3.3 [110/3] via 200.1.1.10, 00:00:15, GigabitEthernet0/0
      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.10.10.0/24 is directly connected, Tunnel0
L        10.10.10.1/32 is directly connected, Tunnel0
      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, GigabitEthernet0/1
L        192.168.1.1/32 is directly connected, GigabitEthernet0/1
R     192.168.2.0/24 [120/1] via 10.10.10.2, 00:00:20, Tunnel0
      200.1.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        200.1.1.0/24 is directly connected, GigabitEthernet0/0
L        200.1.1.1/32 is directly connected, GigabitEthernet0/0
O     200.2.2.0/24 [110/11] via 200.1.1.10, 00:01:35, GigabitEthernet0/0
O     200.3.3.0/24 [110/2] via 200.1.1.10, 00:01:35, GigabitEthernet0/0
```

Figure 4: R1 routing table

```
# Ent  Peer NBMA Addr Peer Tunnel Add State  UpDn Tm Attrb
----- --------------- --------------- ----- -------- -----
    1 1.1.1.1             10.10.10.1    UP 00:07:28     S
```

Figure 5: R2 NHRP cache

```
# Ent  Peer NBMA Addr Peer Tunnel Add State  UpDn Tm Attrb
----- --------------- --------------- ----- -------- -----
    1 1.1.1.1             10.10.10.1    UP 00:05:54     S
```

Figure 6: R3 NHRP cache

After a ping between PC2 and PC3 both the routing tables (Figure 7) and the NHRP cache (Figures 8 and 9) get updated.

```
      1.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        1.1.1.0/24 is directly connected, Loopback0
L        1.1.1.1/32 is directly connected, Loopback0
      2.0.0.0/32 is subnetted, 1 subnets
O        2.2.2.2 [110/12] via 200.1.1.10, 00:09:22, GigabitEthernet0/0
      3.0.0.0/32 is subnetted, 1 subnets
O        3.3.3.3 [110/3] via 200.1.1.10, 00:08:52, GigabitEthernet0/0
      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.10.10.0/24 is directly connected, Tunnel0
L        10.10.10.1/32 is directly connected, Tunnel0
      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.0/24 is directly connected, GigabitEthernet0/1
L        192.168.1.1/32 is directly connected, GigabitEthernet0/1
R     192.168.2.0/24 [120/1] via 10.10.10.2, 00:00:00, Tunnel0
R     192.168.3.0/24 [120/1] via 10.10.10.3, 00:00:18, Tunnel0
      200.1.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        200.1.1.0/24 is directly connected, GigabitEthernet0/0
L        200.1.1.1/32 is directly connected, GigabitEthernet0/0
O     200.2.2.0/24 [110/11] via 200.1.1.10, 00:10:12, GigabitEthernet0/0
O     200.3.3.0/24 [110/2] via 200.1.1.10, 00:10:12, GigabitEthernet0/0
```

Figure 7: R1 routing table after ping

```
# Ent  Peer NBMA Addr Peer Tunnel Add State  UpDn Tm Attrb
----- --------------- --------------- ----- -------- -----
    1 1.1.1.1             10.10.10.1    UP 00:11:09     S
    2 3.3.3.3             10.10.10.3    UP 00:02:49   DT1
                          10.10.10.3    UP 00:02:49   DT1
```

Figure 8: R2 NHRP cache after ping

```
# Ent  Peer NBMA Addr Peer Tunnel Add State  UpDn Tm Attrb
----- --------------- --------------- ----- -------- -----
    1 1.1.1.1             10.10.10.1    UP 00:10:56     S
    2 2.2.2.2             10.10.10.2    UP 00:03:07   DT1
                          10.10.10.2    UP 00:03:07   DT1
```

Figure 9: R3 NHRP Cache after ping

When a ping from PC2 to PC3 (Figure 10), R2 sends an ICMP Echo Request to R1 (Figure 11) and R1 forwards it to R3 (figure 12).

```
437.880214   192.168.2.100   192.168.3.100   ICMP   122 Echo (ping) request  id=0x65c0, seq=1/256, ttl=63
437.883363   192.168.2.100   192.168.3.100   ICMP   122 Echo (ping) request  id=0x65c0, seq=1/256, ttl=62
437.889518   1.1.1.1         2.2.2.2         NHRP   122 NHRP Traffic Indication
437.920924   192.168.3.100   192.168.2.100   ICMP   122 Echo (ping) reply    id=0x65c0, seq=1/256, ttl=63
437.925938   192.168.3.100   192.168.2.100   ICMP   122 Echo (ping) reply    id=0x65c0, seq=1/256, ttl=62
437.935798   1.1.1.1         3.3.3.3         NHRP   122 NHRP Traffic Indication
```

Figure 10: R1 to RA link when ping occurs

```
> Frame 267: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface -, id 0
> Ethernet II, Src: c2:01:06:bb:00:00 (c2:01:06:bb:00:00), Dst: 0c:ab:aa:7a:00:00 (0c:ab:aa:7a:00:00)
> Internet Protocol Version 4, Src: 2.2.2.2, Dst: 1.1.1.1
> Generic Routing Encapsulation (IP)
> Internet Protocol Version 4, Src: 192.168.2.100, Dst: 192.168.3.100
> Internet Control Message Protocol
```

Figure 11: R2 sends an ICMP Echo Request to R1

```
> Frame 268: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface -, id 0
> Ethernet II, Src: 0c:ab:aa:7a:00:00 (0c:ab:aa:7a:00:00), Dst: c2:01:06:bb:00:00 (c2:01:06:bb:00:00)
> Internet Protocol Version 4, Src: 1.1.1.1, Dst: 3.3.3.3
> Generic Routing Encapsulation (IP)
> Internet Protocol Version 4, Src: 192.168.2.100, Dst: 192.168.3.100
> Internet Control Message Protocol
```

Figure 12: R1 forwards it to R3

In our case we have R1 sending the NHRP Redirect to R2 (Figure 13) and we also have R1 forwarding the packet to R3 (Figure 14) but we don't have R2 sending a NHRP resolution to R1 and also the reply by R3 to the NHRP resolution. Still afterwards the ICMP request and replies are directly between R2 and R3 (Figure 15 and Figure 16).

```
> Frame 269: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface -, id 0
> Ethernet II, Src: 0c:ab:aa:7a:00:00 (0c:ab:aa:7a:00:00), Dst: c2:01:06:bb:00:00 (c2:01:06:bb:00:00)
> Internet Protocol Version 4, Src: 1.1.1.1, Dst: 2.2.2.2
> Generic Routing Encapsulation (NHRP)
v Next Hop Resolution Protocol (NHRP Traffic Indication)
  > NHRP Fixed Header
  v NHRP Mandatory Part
      Source Protocol Len: 4
      Destination Protocol Len: 4
      Source NBMA Address: 1.1.1.1
      Source Protocol Address: 10.10.10.1
      Destination Protocol Address: 192.168.2.100
  v Packet Causing Indication
    > Internet Protocol Version 4, Src: 192.168.2.100, Dst: 192.168.3.100
    > Internet Control Message Protocol
    > HiPerConTracer Trace Service
  > Forward Transit NHS Record Extension
  > Reverse Transit NHS Record Extension
  > Cisco NAT Address Extension
  > End of Extension
```

Figure 13: R1 send NHRP Redirect to R2

```
> Frame 272: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface -, id 0
> Ethernet II, Src: 0c:ab:aa:7a:00:00 (0c:ab:aa:7a:00:00), Dst: c2:01:06:bb:00:00 (c2:01:06:bb:00:00)
> Internet Protocol Version 4, Src: 1.1.1.1, Dst: 3.3.3.3
> Generic Routing Encapsulation (NHRP)
v Next Hop Resolution Protocol (NHRP Traffic Indication)
  > NHRP Fixed Header
  v NHRP Mandatory Part
      Source Protocol Len: 4
      Destination Protocol Len: 4
      Source NBMA Address: 1.1.1.1
      Source Protocol Address: 10.10.10.1
      Destination Protocol Address: 192.168.3.100
  > Packet Causing Indication
  > Forward Transit NHS Record Extension
  > Reverse Transit NHS Record Extension
  > Cisco NAT Address Extension
  > End of Extension
```

Figure 14: R1 forwards the NHRP Resolution to R3

403	738.079467	192.168.2.100	192.168.3.100	ICMP	122	Echo (ping) request id=0x95c1, seq=1/256, ttl=63 (reply in 404)
404	738.084737	192.168.3.100	192.168.2.100	ICMP	122	Echo (ping) reply id=0x95c1, seq=1/256, ttl=63 (request in 403)
405	738.438399	0c:6c:d1:49:00:00	DEC-MOP-Remote-Cons…	0x6002	77	DEC DNA Remote Console
406	739.102743	192.168.2.100	192.168.3.100	ICMP	122	Echo (ping) request id=0x96c1, seq=2/512, ttl=63 (reply in 407)
407	739.105103	192.168.3.100	192.168.2.100	ICMP	122	Echo (ping) reply id=0x96c1, seq=2/512, ttl=63 (request in 406)
408	739.349717	0c:6c:d1:49:00:00	0c:6c:d1:49:00:00	LOOP	60	Reply
409	740.126936	192.168.2.100	192.168.3.100	ICMP	122	Echo (ping) request id=0x97c1, seq=3/768, ttl=63 (reply in 410)
410	740.129069	192.168.3.100	192.168.2.100	ICMP	122	Echo (ping) reply id=0x97c1, seq=3/768, ttl=63 (request in 409)
411	740.862237	0c:6c:d1:49:00:00	CDP/VTP/DTP/PAgP/UD…	CDP	360	Device ID: Router Port ID: GigabitEthernet0/0
412	741.146938	192.168.2.100	192.168.3.100	ICMP	122	Echo (ping) request id=0x98c1, seq=4/1024, ttl=63 (reply in 413)
413	741.148922	192.168.3.100	192.168.2.100	ICMP	122	Echo (ping) reply id=0x98c1, seq=4/1024, ttl=63 (request in 412)
414	742.172349	192.168.2.100	192.168.3.100	ICMP	122	Echo (ping) request id=0x99c1, seq=5/1280, ttl=63 (reply in 415)
415	742.175601	192.168.3.100	192.168.2.100	ICMP	122	Echo (ping) reply id=0x99c1, seq=5/1280, ttl=63 (request in 414)

Figure 15: R3 to RA link when ping occurs

```
> Frame 233: 122 bytes on wire (976 bits), 122 bytes captured (976 bits) on interface -, id 0
> Ethernet II, Src: 0c:b0:e8:81:00:00 (0c:b0:e8:81:00:00), Dst: c2:01:06:bb:00:01 (c2:01:06:bb:00:01)
> Internet Protocol Version 4, Src: 2.2.2.2, Dst: 3.3.3.3
> Generic Routing Encapsulation (IP)
> Internet Protocol Version 4, Src: 192.168.2.100, Dst: 192.168.3.100
> Internet Control Message Protocol
```

Figure 16: R2 sends ICMP Request directly to R3

VRFs and BGP-MPLS L3VPNs

Network structure

For this exercise we've created a network where there are 7 routers, 3 of them work as providers (routers P, PE1 and PE2 from Figure 4) and the other 4 are considered to be customers (routers Blue1, Blue2, Red1 and Red2 from Figure 1).

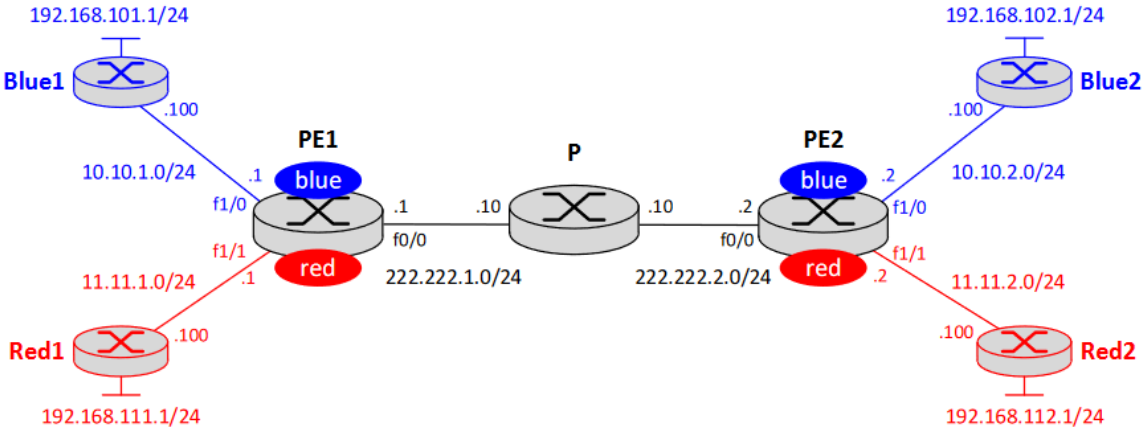


Figure 1: Network structure and IP distribution

The overall idea of this exercise is to configure a BGP-MPLS Layer 3 VPN with the use of VRF (Virtual Routing and Forwarding). As we can see in Figure 1, we've configured two VRFs (Red and Blue) that allow customers from one VRF to communicate between each other while being isolated from the other VRF's customers and vice versa. The providers form a network that is shared between both VRFs and works as a VPN tunnel hub through the use of MPLS label stacking (see Figure 2). This means that we have a VPN for each of the VRFs, equipped with its exclusive routing table (see example for VRF Blue in Figure 3).

```
> MultiProtocol Label Switching Header, Label: 17, Exp: 0, S: 0, TTL: 254
> MultiProtocol Label Switching Header, Label: 18, Exp: 0, S: 1, TTL: 254
v Internet Protocol Version 4, Src: 192.168.101.1, Dst: 192.168.102.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 100
    Identification: 0x0005 (5)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 254
    Protocol: ICMP (1)
    Header Checksum: 0x7040 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.101.1
    Destination Address: 192.168.102.1
```

Figure 2: Ping from Blue1 to Blue2

```
PE1#show ip route vrf blue

Routing Table: blue
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override

Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.10.1.0/24 is directly connected, FastEthernet1/0
L        10.10.1.1/32 is directly connected, FastEthernet1/0
S     192.168.101.0/24 [1/0] via 10.10.1.100
B     192.168.102.0/24 [200/0] via 2.2.2.2, 00:05:52
PE1#
```

Figure 3: Routing table for VRF Blue

Something to note on the previously mentioned Figure 2 is that Blue1's IP address is its loopback address instead of being 10.10.1.100/24. This is because this method uses the customers' loopback addresses to create "networks" and these are considered as the endpoints of their respective VRFs (as seen in Figure 4).

```
PE1#show ip route vrf blue

Routing Table: blue
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       + - replicated route, % - next hop override

Gateway of last resort is not set

      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.10.1.0/24 is directly connected, FastEthernet1/0
L        10.10.1.1/32 is directly connected, FastEthernet1/0
S     192.168.101.0/24 [1/0] via 10.10.1.100
B     192.168.102.0/24 [200/0] via 2.2.2.2, 00:05:52
PE1#show mpls forwarding-table
Local      Outgoing   Prefix           Bytes Label   Outgoing   Next Hop
Label      Label      or Tunnel Id     Switched      interface
17         17         2.2.2.2/32       0             Fa0/0      222.222.1.10
18         Pop Label  222.222.2.0/24   0             Fa0/0      222.222.1.10
19         No Label   192.168.101.0/24[V]   \
                                       2280          Fa1/0      10.10.1.100
20         No Label   192.168.111.0/24[V]   \
                                       1140          Fa1/1      11.11.1.100
PE1#
```

Figure 4: Routing and MPLS tables from router PE1

BGP Messages

When the BGP session is established there is a flow of BGP Open and BGP Update packets (as seen in Figure 5). BGP Open advertises the capabilities that are necessary for the VPN to function (Figure 6), while BGP Update advertises the Route Target, Route Distinguisher and the MPLS labels used by each of the VRFs that have been configured (Figure 7).

bgp

No.	Time	Source	Destination	Protocol	Length	Info
190	221.894489	1.1.1.1	2.2.2.2	BGP	119	OPEN Message
192	222.101840	2.2.2.2	1.1.1.1	BGP	115	OPEN Message
193	222.113536	1.1.1.1	2.2.2.2	BGP	77	KEEPALIVE Message
194	222.113638	2.2.2.2	1.1.1.1	BGP	73	KEEPALIVE Message
250	273.772669	1.1.1.1	2.2.2.2	BGP	77	KEEPALIVE Message
253	274.695367	2.2.2.2	1.1.1.1	BGP	73	KEEPALIVE Message
254	274.705622	2.2.2.2	1.1.1.1	BGP	234	UPDATE Message, UPDATE Message
256	274.716097	2.2.2.2	1.1.1.1	BGP	77	UPDATE Message
257	275.297217	1.1.1.1	2.2.2.2	BGP	238	UPDATE Message, UPDATE Message
258	275.297289	1.1.1.1	2.2.2.2	BGP	81	UPDATE Message
261	276.015748	1.1.1.1	2.2.2.2	BGP	87	UPDATE Message
267	280.998546	2.2.2.2	1.1.1.1	BGP	83	UPDATE Message

Figure 5: BGP session initialization

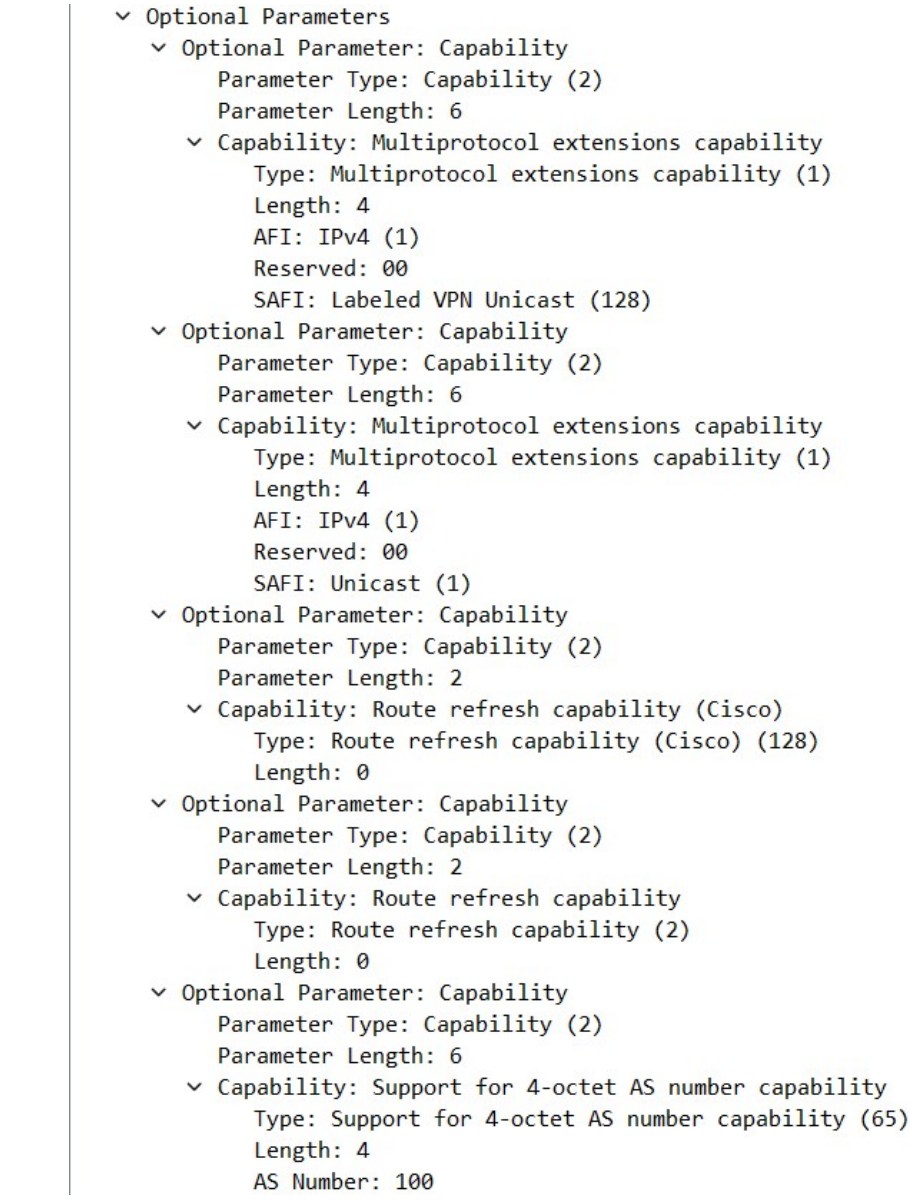


Figure 6: BGP Open packet example

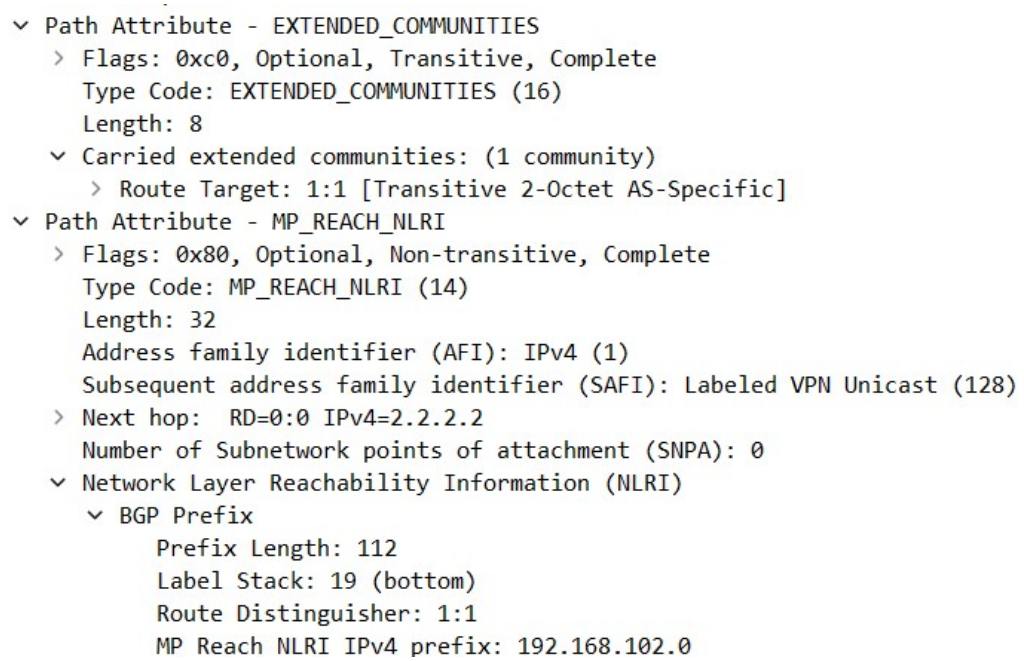


Figure 7: BGP Update packet example

Looking again at Figure 5, we can affirm that for this kind of VPN the BGP peers need to have multiprotocol extensions for IPv4, route refresh capability and support for 4-octet AS numbers.

If we observe Figure 6 we can spot several parameters of particular interest. Route Target determines how to import or export VPN routes from the VRFs and Route Distinguisher is used to distinguish between the subnets of different customers. As for the MPLS labels, we have both an inner and outer label. The inner label distinguishes the customers and the outer label is used for routing in the provider's network.

Connecting network namespaces to external networks

Network structure

For this exercise we have a simple network (shown in Figure 1), where we have two Ubuntu Server appliances connected to a switch. The goal here is to configure network namespaces on one of them that can connect to an external network. All of the following configurations were performed in the ub2 device, while we used ub1 as a member of an external network.

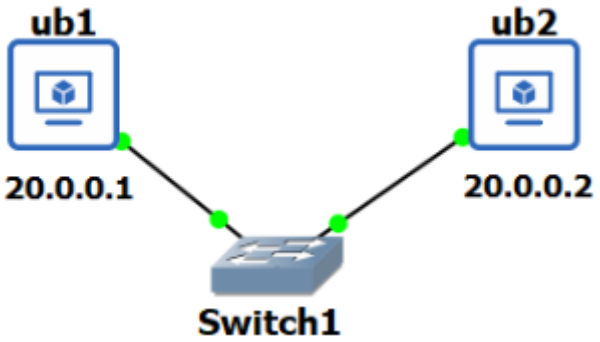


Figure 1: Network topology

Namespace setup

We started by creating two namespaces - red and blue - and a bridge interface. Said interface can be seen in Figure 2. We've also created "cables" to connect the bridge interface to each of the namespaces (Figure 3). We've assigned an IP address to each of these new interfaces, 11.0.0.1/24 for the red cable and 11.0.0.2/24 for the blue cable.

```
root@gns3:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 0c:a7:d6:f0:00:00 brd ff:ff:ff:ff:ff:ff
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 0c:a7:d6:f0:00:01 brd ff:ff:ff:ff:ff:ff
4: br1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/ether e2:56:80:28:f4:13 brd ff:ff:ff:ff:ff:ff
root@gns3:~#
```

Figure 2: Bridge interface creation

```
root@gns3:~# ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 0c:a7:d6:f0:00:00 brd ff:ff:ff:ff:ff:ff
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 0c:a7:d6:f0:00:01 brd ff:ff:ff:ff:ff:ff
4: br1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default qlen 1000
    link/ether 2e:af:c4:f0:d4:e1 brd ff:ff:ff:ff:ff:ff
5: iredbr1@if6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop master br1 state DOWN mode DEFAULT group default qlen 1000
    link/ether 2e:af:c4:f0:d4:e1 brd ff:ff:ff:ff:ff:ff link-netnsid 0
7: ibluebr1@if8: <BROADCAST,MULTICAST> mtu 1500 qdisc noop master br1 state DOWN mode DEFAULT group default qlen 1000
    link/ether fa:f4:b7:98:ef:d4 brd ff:ff:ff:ff:ff:ff link-netnsid 1
root@gns3:~#
```

Figure 3: Cable interface creation

After this initial setup, we performed some tests in order to verify if we could ping between namespaces and from the host to a namespace. Although we could achieve this for the former (as seen in Figure 4), the same cannot be said for the latter. To solve this issue, we assigned an IP address - 11.0.0.25/24 - to the bridge interface. After this addition we could finally achieve connection from the host to the namespaces (example in Figure 5).

```
root@gns3:~# ip netns exec red ping 11.0.0.2
PING 11.0.0.2 (11.0.0.2) 56(84) bytes of data.
64 bytes from 11.0.0.2: icmp_seq=1 ttl=64 time=1.00 ms
64 bytes from 11.0.0.2: icmp_seq=2 ttl=64 time=0.033 ms
64 bytes from 11.0.0.2: icmp_seq=3 ttl=64 time=0.032 ms
64 bytes from 11.0.0.2: icmp_seq=4 ttl=64 time=0.033 ms
^C
--- 11.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3042ms
rtt min/avg/max/mdev = 0.032/0.274/1.000/0.419 ms
root@gns3:~#
```

Figure 4: Ping test from Red to Blue

```
root@gns3:~# ping 11.0.0.1
PING 11.0.0.1 (11.0.0.1) 56(84) bytes of data.
64 bytes from 11.0.0.1: icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from 11.0.0.1: icmp_seq=2 ttl=64 time=0.035 ms
64 bytes from 11.0.0.1: icmp_seq=3 ttl=64 time=0.030 ms
64 bytes from 11.0.0.1: icmp_seq=4 ttl=64 time=0.029 ms
^C
--- 11.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3031ms
rtt min/avg/max/mdev = 0.029/0.040/0.067/0.016 ms
root@gns3:~#
```

Figure 5: Ping test from host to Red

In order for the namespaces to connect to an external network we did a couple more configurations on the host. We set the bridge's IP address as the default gateway for both namespaces and we also configured NAT for masquerading all traffic from the namespaces

(Figure 6). The result was a successful connection from a namespace to an external network, as can be seen in Figure 7.

```
root@gns3:~# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  11.0.0.0/24          anywhere
root@gns3:~#
```

Figure 6: Firewall NAT configuration

```
root@gns3:~# ip netns exec blue ping 20.0.0.1
PING 20.0.0.1 (20.0.0.1) 56(84) bytes of data.
64 bytes from 20.0.0.1: icmp_seq=1 ttl=63 time=3.76 ms
64 bytes from 20.0.0.1: icmp_seq=2 ttl=63 time=0.853 ms
64 bytes from 20.0.0.1: icmp_seq=3 ttl=63 time=1.03 ms
^C
--- 20.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 0.853/1.884/3.763/1.331 ms
root@gns3:~#
```

Figure 7: Ping test from Blue to ub1

Macvlan networks with VLANs

Network Structure

We used the following network topology in this exercice to set up two isolated macvlan networks, where the isolation is performed via IEEE 802.1q trunking (VLANs)

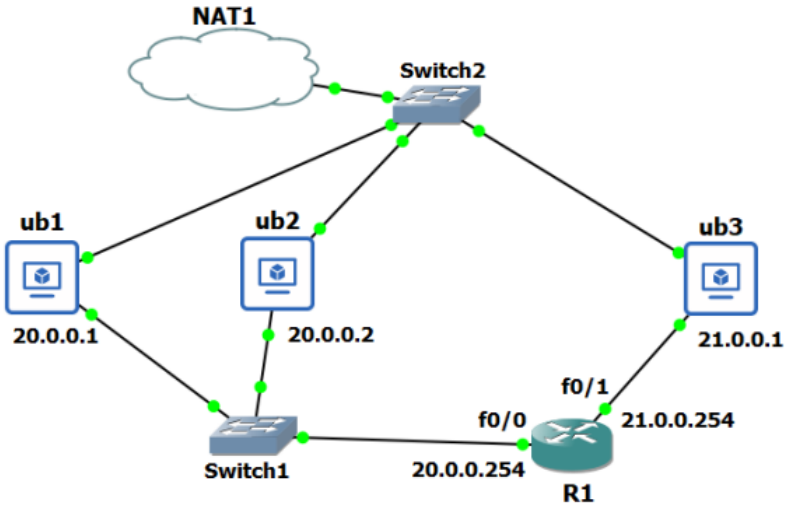


Figure 1: Network topology

Configuration

Macvlan networks and VLANs are two different concepts that can be used together to provide network isolation and segmentation in a virtualized environment. In certain scenarios we may want to combine Macvlan networks with VLANs to achieve both Layer 2 and Layer 3 network segregation. This can be useful in environments where we need to isolate not only the Layer 2 traffic based on MAC addresses but also the Layer 3 traffic based on IP addresses. To combine these two, we created VLAN subinterfaces on the physical network interface and associated each VLAN subinterface with a separate Macvlan network. This allowed us to achieve both network segregation at Layer 2 using Macvlan and further isolate traffic within each Macvlan network using VLAN tags. This permits to create a more granular and secure network architecture, allowing for flexible network segmentation and isolation at both Layer 2 and Layer 3.
We configured two Macvlan networks with VLANs for ub1 and ub2. We called the two networks, my-8021q-macvlan-net and my-8021q-macvlan-second-net with the subnet 172.16.86.0/24 with VLAN id 10 and 172.16.87.0/24 with VLAN id 20 respectively.
After this implementation we could ping from ub1 to ub2 inside the same VLAN network as we can observe in the figure below, but we could not communicate between containers attached to different networks as we can see in figure 3.

```
root@gns3:~# docker exec my-second-macvlan-alpine ping 172.16.86.4
PING 172.16.86.4 (172.16.86.4): 56 data bytes
64 bytes from 172.16.86.4: seq=0 ttl=64 time=3.210 ms
64 bytes from 172.16.86.4: seq=1 ttl=64 time=1.450 ms
64 bytes from 172.16.86.4: seq=2 ttl=64 time=1.478 ms
64 bytes from 172.16.86.4: seq=3 ttl=64 time=1.488 ms
64 bytes from 172.16.86.4: seq=4 ttl=64 time=1.973 ms
```

Figure 2: Ping between containers attached to the same network

```
root@gns3:~# docker exec my-second-macvlan-alpine ping 172.16.87.5
PING 172.16.87.5 (172.16.87.5): 56 data bytes
```

Figure 3: Ping between containers attached to the different networks

Tagging

In a docker environment, when we create a network with a specific VLAN id, Docker automatically adds VLAN tagging to the packets transmitted between the containers attached to that network. This tagging allows the underlying network infrastructure, such as switches and routers, to correctly handle and route the traffic based on the VLAN id.

By using the same VLAN id for containers within a network, we can ensure that their communication remains isolated and separate from other containers or networks with different VLAN ids. This allows for better network segmentation and control over the traffic flow within the Docker environment.

To further analyze the VLAN tagging and traffic within the containers, we examined the Wireshark captures. We looked for the VLAN tags in the Ethernet headers of the captured packets. The VLAN ID was visible in the "802.1Q Virtual LAN" field within the Ethernet frame as we can see in the next two figures.

By inspecting the VLAN tags, we could verify that the VLAN ID is consistent for the containers within the same network. Additionally, we can observe how the VLAN-tagged packets are being forwarded and processed by the network infrastructure, including switches and routers, to ensure the intended segregation and isolation are being maintained.

```
> Frame 926: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface -, id 0
> Ethernet II, Src: 02:42:ac:10:57:05 (02:42:ac:10:57:05), Dst: 02:42:ac:10:57:02 (02:42:ac:10:57:02)
v 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 20
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = DEI: Ineligible
    .... 0000 0001 0100 = ID: 20
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.16.87.5, Dst: 172.16.87.2
> Internet Control Message Protocol
```

Figure 4: 172.16.87.0/24 with VLAN id 20

```
> Frame 934: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface -, id 0
> Ethernet II, Src: 02:42:ac:10:56:02 (02:42:ac:10:56:02), Dst: 02:42:ac:10:56:04 (02:42:ac:10:56:04)
v 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 10
    000. .... .... .... = Priority: Best Effort (default) (0)
    ...0 .... .... .... = DEI: Ineligible
    .... 0000 0000 1010 = ID: 10
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.16.86.2, Dst: 172.16.86.4
> Internet Control Message Protocol
```

Figure 5: 172.16.86.0/24 with VLAN id 10

Linux VXLAN with multicast routing

Network Structure

In this exercise we used the network (Figure 1) and the objective of this exercise is to create 2 overlay networks using VXLAN and multicast routing

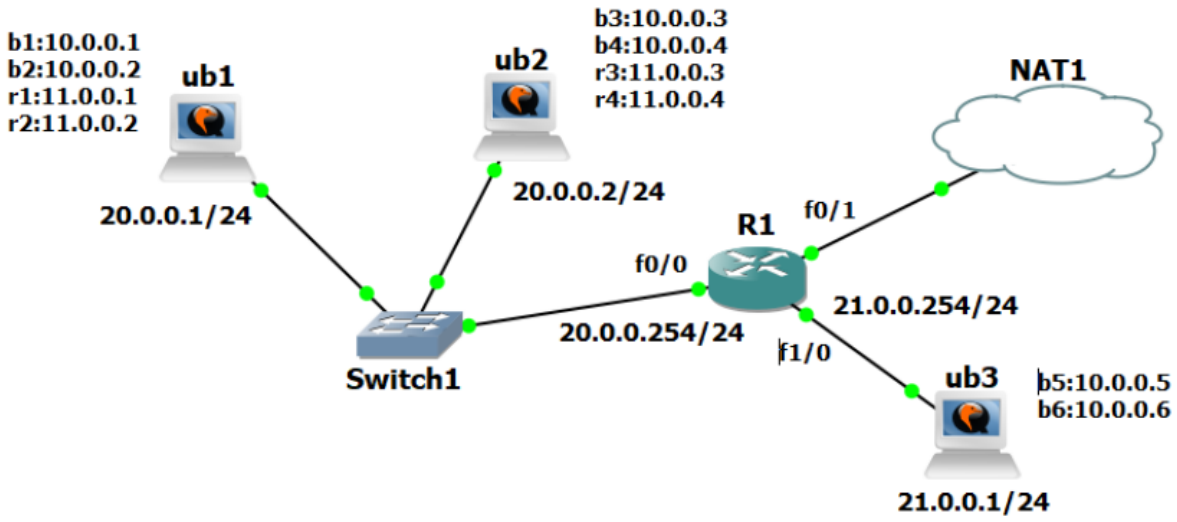


Figure 1: Network structure and IP distribution

VXLAN configuration

After setting up the VXLAN and multicast routing, in ub1 we can see 2 addresses that refer to the VXLAN (Figure 2), and also by looking at the IGMP groups at R1 we can see that both VXLAN are set up (Figure 3), and that the 3 ub's are sending IGMP packets to R1 (Figure 4 and Figure 5).

```
root@ub1:~# ip add
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 0c:1a:e7:c5:00:00 brd ff:ff:ff:ff:ff:ff
    inet 20.0.0.1/24 brd 20.0.0.255 scope global ens3
       valid_lft forever preferred_lft forever
    inet6 fe80::e1a:e7ff:fec5:0/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:2b:15:95:ff brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
4: blue: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 1e:32:a3:b5:55:f1 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.101/24 scope global blue
       valid_lft forever preferred_lft forever
    inet6 fe80::1c32:a3ff:feb5:55f1/64 scope link
       valid_lft forever preferred_lft forever
5: red: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether 96:fa:8b:92:40:42 brd ff:ff:ff:ff:ff:ff
    inet 11.0.0.101/24 scope global red
       valid_lft forever preferred_lft forever
    inet6 fe80::94fa:8bff:fe92:4042/64 scope link
       valid_lft forever preferred_lft forever
```

Figure 2: IP addresses at ub1

```
R1#show ip igmp groups
IGMP Connected Group Membership
Group Address    Interface                Uptime    Expires   Last Reporter   Group Accounted
239.1.1.3        FastEthernet0/1          00:05:00  00:02:53  21.0.0.1
239.1.1.3        FastEthernet0/0          00:05:03  00:02:54  20.0.0.2
239.1.1.6        FastEthernet0/0          00:05:01  00:02:54  20.0.0.1
224.0.1.40       FastEthernet0/0          00:05:03  00:02:52  20.0.0.254
R1#show ip igmp groups
IGMP Connected Group Membership
Group Address    Interface                Uptime    Expires   Last Reporter   Group Accounted
239.1.1.3        FastEthernet0/1          00:06:14  00:02:41  21.0.0.1
239.1.1.3        FastEthernet0/0          00:06:16  00:02:38  20.0.0.2
239.1.1.6        FastEthernet0/0          00:06:15  00:02:42  20.0.0.2
224.0.1.40       FastEthernet0/0          00:06:16  00:02:37  20.0.0.254
```

Figure 3: IGMP groups at R1

```
      9 35.522137    20.0.0.254      224.0.0.1       IGMPv2     60 Membership Query, general
     10 36.674362    20.0.0.1        239.1.1.3       IGMPv2     46 Membership Report group 239.1.1.3
     11 38.067968    20.0.0.2        224.0.0.251     IGMPv2     46 Membership Report group 224.0.0.251
     12 38.690928    20.0.0.1        239.1.1.6       IGMPv2     46 Membership Report group 239.1.1.6
     14 45.517696    20.0.0.254      224.0.1.40      IGMPv2     60 Membership Report group 224.0.1.40
     24 95.521523    20.0.0.254      224.0.0.1       IGMPv2     60 Membership Query, general
     25 95.569402    20.0.0.2        239.1.1.6       IGMPv2     46 Membership Report group 239.1.1.6
     27 96.876329    20.0.0.1        224.0.0.251     IGMPv2     46 Membership Report group 224.0.0.251
     28 98.521707    20.0.0.254      224.0.1.40      IGMPv2     60 Membership Report group 224.0.1.40
     31 104.400574   20.0.0.2        239.1.1.3       IGMPv2     46 Membership Report group 239.1.1.3
```

Figure 4: IGMP messages by ub1 and ub2

6	24.077177	21.0.0.254	224.0.0.1	IGMPv2	60 Membership Query, general
7	26.054986	21.0.0.1	239.1.1.3	IGMPv2	46 Membership Report group 239.1.1.3
8	28.871412	21.0.0.1	224.0.0.251	IGMPv2	46 Membership Report group 224.0.0.251

Figure 5: IGMP messages by ub3

The objective of setting up multicast routing protocol is to allow and efficient distribution of multicast traffic (Figure 6), enabling an efficient use of network resources as the same packet can be delivered to multiple destinations without the need to separate unicast transmissions

```
(*, 239.1.1.3), 00:24:48/00:02:11, RP 0.0.0.0, flags: DC
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/1, Forward/Dense, 00:24:46/00:00:00
    FastEthernet0/0, Forward/Dense, 00:24:48/00:00:00

(*, 239.1.1.6), 00:24:47/00:02:09, RP 0.0.0.0, flags: DC
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Dense, 00:24:48/00:00:00

(*, 224.0.1.40), 00:24:50/00:02:03, RP 0.0.0.0, flags: DCL
  Incoming interface: Null, RPF nbr 0.0.0.0
  Outgoing interface list:
    FastEthernet0/0, Forward/Dense, 00:24:50/00:00:00
```

Figure 6: Multicast route at R1

To test the VXLAN we did pings from b2 to b6 (figure 7 and figure 8), we also checked the MAC address of b2 to see if it is correct (Figure 9)

```
> Frame 102: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface -, id 0
> Ethernet II, Src: 0c:1a:e7:c5:00:00 (0c:1a:e7:c5:00:00), Dst: c2:01:0e:12:00:00 (c2:01:0e:12:00:00)
> Internet Protocol Version 4, Src: 20.0.0.1, Dst: 21.0.0.1
> User Datagram Protocol, Src Port: 44785, Dst Port: 4789
> Virtual eXtensible Local Area Network
> Ethernet II, Src: 02:42:0a:00:00:02 (02:42:0a:00:00:02), Dst: 02:42:0a:00:00:06 (02:42:0a:00:00:06)
> Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.6
> Internet Control Message Protocol
```

Figure 7: ICMP request from b2 to b6

```
> Frame 103: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface -, id 0
> Ethernet II, Src: c2:01:0e:12:00:00 (c2:01:0e:12:00:00), Dst: 0c:1a:e7:c5:00:00 (0c:1a:e7:c5:00:00)
> Internet Protocol Version 4, Src: 21.0.0.1, Dst: 20.0.0.1
> User Datagram Protocol, Src Port: 57633, Dst Port: 4789
v Virtual eXtensible Local Area Network
  > Flags: 0x0800, VXLAN Network ID (VNI)
    Group Policy ID: 0
    VXLAN Network Identifier (VNI): 33
    Reserved: 0
> Ethernet II, Src: 02:42:0a:00:00:06 (02:42:0a:00:00:06), Dst: 02:42:0a:00:00:02 (02:42:0a:00:00:02)
> Internet Protocol Version 4, Src: 10.0.0.6, Dst: 10.0.0.2
> Internet Control Message Protocol
```

Figure 8: ICMP reply from b6 to b2

```
root@ub1:~# docker exec b1 arp
? (10.0.0.3) at 02:42:0a:00:00:03 [ether]  on eth0
b2.bluenet (10.0.0.2) at 02:42:0a:00:00:02 [ether]  on eth0
root@ub1:~# docker exec b2 arp
? (10.0.0.6) at 02:42:0a:00:00:06 [ether]  on eth0
b1.bluenet (10.0.0.1) at 02:42:0a:00:00:01 [ether]  on eth0
```

Figure 9: MAC addresses of b1 and b2

If the ub1 doesn't know b6 (in this case we deleted ub1 knowledge of 10.0.0.6) and a ping between b2 and b6 occurs there is an exchange of ARP messages in order for ub1 to have the knowledge of b6 (Figure 10 and Figure 11). Analyzing the images we can check that the VXLAN identifier is 33 that refers to the blue VXLAN.

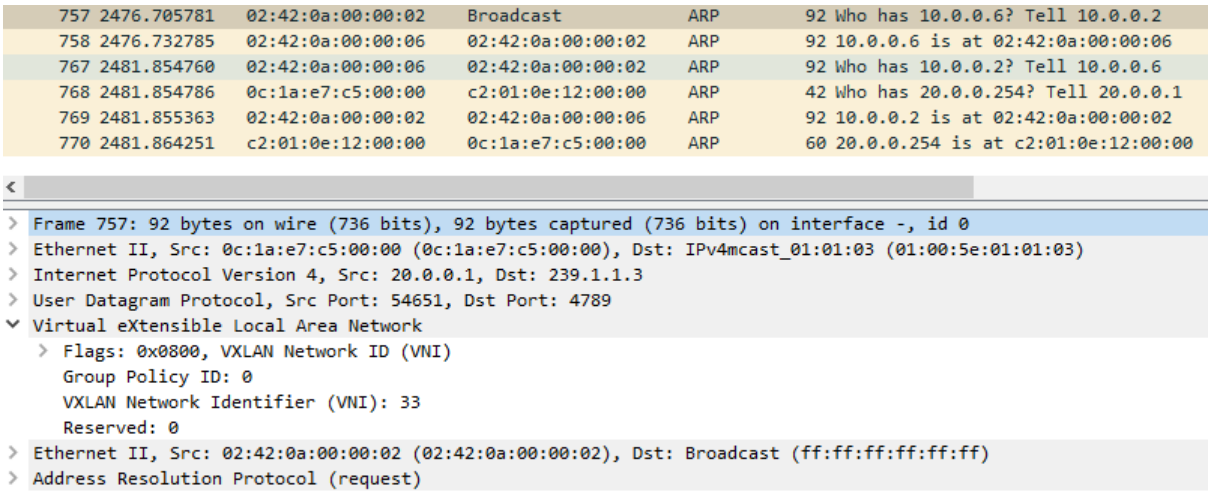


Figure 10: ARP request

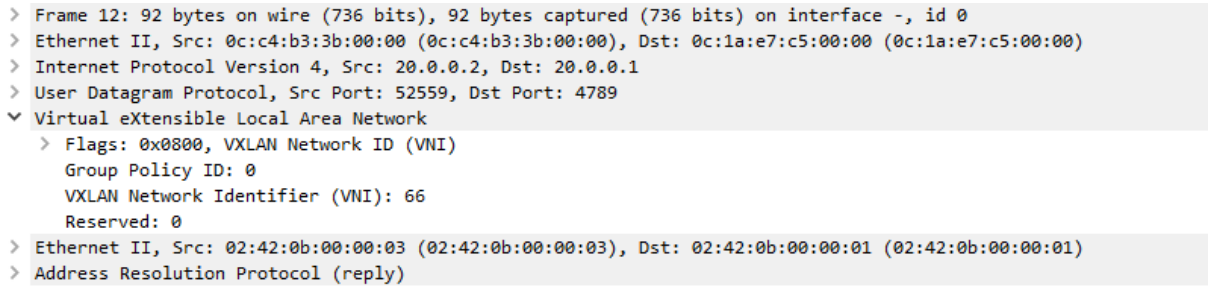


Figure 11: ARP reply

To check if the red VXLAN was also working correctly we did a similar test as above but the pings occur from r1 to r3 (Figure 12 and Figure 13) and deleting ub1 knowledge of r3 and checking the ARP messages again (Figure 14 and Figure 15). Looking at the VNI we see it's 66 that refers to the red VXLAN.

```
> Frame 13: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface -, id 0
> Ethernet II, Src: 0c:1a:e7:c5:00:00 (0c:1a:e7:c5:00:00), Dst: 0c:c4:b3:3b:00:00 (0c:c4:b3:3b:00:00)
> Internet Protocol Version 4, Src: 20.0.0.1, Dst: 20.0.0.2
> User Datagram Protocol, Src Port: 58589, Dst Port: 4789
v Virtual eXtensible Local Area Network
    > Flags: 0x0800, VXLAN Network ID (VNI)
      Group Policy ID: 0
      VXLAN Network Identifier (VNI): 66
      Reserved: 0
> Ethernet II, Src: 02:42:0b:00:00:01 (02:42:0b:00:00:01), Dst: 02:42:0b:00:00:03 (02:42:0b:00:00:03)
> Internet Protocol Version 4, Src: 11.0.0.1, Dst: 11.0.0.3
> Internet Control Message Protocol
```

Figure 12: ICMP request from r1 to r3

```
> Frame 14: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface -, id 0
> Ethernet II, Src: 0c:c4:b3:3b:00:00 (0c:c4:b3:3b:00:00), Dst: 0c:1a:e7:c5:00:00 (0c:1a:e7:c5:00:00)
> Internet Protocol Version 4, Src: 20.0.0.2, Dst: 20.0.0.1
> User Datagram Protocol, Src Port: 44140, Dst Port: 4789
v Virtual eXtensible Local Area Network
    > Flags: 0x0800, VXLAN Network ID (VNI)
      Group Policy ID: 0
      VXLAN Network Identifier (VNI): 66
      Reserved: 0
> Ethernet II, Src: 02:42:0b:00:00:03 (02:42:0b:00:00:03), Dst: 02:42:0b:00:00:01 (02:42:0b:00:00:01)
> Internet Protocol Version 4, Src: 11.0.0.3, Dst: 11.0.0.1
> Internet Control Message Protocol
```

Figure 13: ICMP response from r3 to r1

```
    11 59.239800    02:42:0b:00:00:01   Broadcast           ARP       92 Who has 11.0.0.3? Tell 11.0.0.1
    12 59.240438    02:42:0b:00:00:03   02:42:0b:00:00:01   ARP       92 11.0.0.3 is at 02:42:0b:00:00:03
    13 59.240873    11.0.0.1            11.0.0.3            ICMP     148 Echo (ping) request  id=0x001f, seq=0/0, ttl=64 (reply in 14)
    14 59.241281    11.0.0.3            11.0.0.1            ICMP     148 Echo (ping) reply    id=0x001f, seq=0/0, ttl=64 (request in 13)
    15 60.247895    11.0.0.1            11.0.0.3            ICMP     148 Echo (ping) request  id=0x001f, seq=1/256, ttl=64 (reply in 16)
    16 60.248920    11.0.0.3            11.0.0.1            ICMP     148 Echo (ping) reply    id=0x001f, seq=1/256, ttl=64 (request in 15)
    17 61.250245    11.0.0.1            11.0.0.3            ICMP     148 Echo (ping) request  id=0x001f, seq=2/512, ttl=64 (reply in 18)
    18 61.251288    11.0.0.3            11.0.0.1            ICMP     148 Echo (ping) reply    id=0x001f, seq=2/512, ttl=64 (request in 17)
    19 62.250827    11.0.0.1            11.0.0.3            ICMP     148 Echo (ping) request  id=0x001f, seq=3/768, ttl=64 (reply in 20)
    20 62.251268    11.0.0.3            11.0.0.1            ICMP     148 Echo (ping) reply    id=0x001f, seq=3/768, ttl=64 (request in 19)
    21 64.428688    02:42:0b:00:00:03   02:42:0b:00:00:01   ARP       92 Who has 11.0.0.1? Tell 11.0.0.3
    22 64.429181    0c:c4:b3:3b:00:00   0c:1a:e7:c5:00:00   ARP       42 Who has 20.0.0.1? Tell 20.0.0.2
    23 64.429724    0c:1a:e7:c5:00:00   0c:c4:b3:3b:00:00   ARP       42 20.0.0.1 is at 0c:1a:e7:c5:00:00
    24 64.429745    02:42:0b:00:00:01   02:42:0b:00:00:03   ARP       92 11.0.0.1 is at 02:42:0b:00:00:01
    25 64.445436    0c:1a:e7:c5:00:00   0c:c4:b3:3b:00:00   ARP       42 Who has 20.0.0.2? Tell 20.0.0.1
    26 64.445773    0c:c4:b3:3b:00:00   0c:1a:e7:c5:00:00   ARP       42 20.0.0.2 is at 0c:c4:b3:3b:00:00
    27 72.779673    20.0.0.254          224.0.0.1           IGMPv2    60 Membership Query, general

> Frame 11: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface -, id 0
> Ethernet II, Src: 0c:1a:e7:c5:00:00 (0c:1a:e7:c5:00:00), Dst: IPv4mcast_01:01:06 (01:00:5e:01:01:06)
> Internet Protocol Version 4, Src: 20.0.0.1, Dst: 239.1.1.6
> User Datagram Protocol, Src Port: 54651, Dst Port: 4789
> Virtual eXtensible Local Area Network
> Ethernet II, Src: 02:42:0b:00:00:01 (02:42:0b:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
> Address Resolution Protocol (request)
```

Figure 14: ARP quest after deletion of ub1 knowledge of r3

```
> Frame 12: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface -, id 0
> Ethernet II, Src: 0c:c4:b3:3b:00:00 (0c:c4:b3:3b:00:00), Dst: 0c:1a:e7:c5:00:00 (0c:1a:e7:c5:00:00)
> Internet Protocol Version 4, Src: 20.0.0.2, Dst: 20.0.0.1
> User Datagram Protocol, Src Port: 52559, Dst Port: 4789
v Virtual eXtensible Local Area Network
    > Flags: 0x0800, VXLAN Network ID (VNI)
      Group Policy ID: 0
      VXLAN Network Identifier (VNI): 66
      Reserved: 0
> Ethernet II, Src: 02:42:0b:00:00:03 (02:42:0b:00:00:03), Dst: 02:42:0b:00:00:01 (02:42:0b:00:00:01)
> Address Resolution Protocol (reply)
```

Figure 15: ARP response

Docker Swarm services

Overview

A Docker Swarm service is a service that is hosted on multiple machines at the same time using replicas in containers. This allows this kind of service to naturally possess load balancing capabilities due to the nature of Docker Swarm. It can also provide good availability due to its self-healing properties and scalability due to its ease of replica deployment.

We've started this exercise by configuring a Docker Swarm service with 5 replicas in the blue network (see Figure 1). This service will later be used to perform tests that allow us to study a Docker Swarm service's functionalities.

```
publish 3000:80 ruivaladas/ws-simple-2etwork blue --name bluesvc --replicas 5 --p
l13d0odegfe057onqlx1xp07l
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service converged
root@gns3:~# docker service ls
ID             NAME      MODE         REPLICAS   IMAGE                          PORTS
l13d0odegfe0   bluesvc   replicated   5/5        ruivaladas/ws-simple-2:latest   *:3000->80/tcp
root@gns3:~# docker service ps bluesvc
ID             NAME           IMAGE                          NODE      DESIRED STATE   CURRENT STATE            ERROR
afxijgg48chw   bluesvc.1      ruivaladas/ws-simple-2:latest   gns3      Running         Running 30 seconds ago
zpyg3upcs6xh    \_ bluesvc.1   ruivaladas/ws-simple-2:latest   gns3      Shutdown        Rejected 5 minutes ago   "No such image: ruivaladas/ws-…"
jif2aqc2to7y   bluesvc.2      ruivaladas/ws-simple-2:latest   gns3      Running         Running 4 minutes ago
fd9iv33mzuwk   bluesvc.3      ruivaladas/ws-simple-2:latest   gns3      Running         Running 2 minutes ago
0bxq3aht55rv    \_ bluesvc.3   ruivaladas/ws-simple-2:latest   gns3      Shutdown        Rejected 5 minutes ago   "No such image: ruivaladas/ws-…"
qu3mu2z617nt   bluesvc.4      ruivaladas/ws-simple-2:latest   gns3      Running         Running 31 seconds ago
eg41m1is7ycz   bluesvc.5      ruivaladas/ws-simple-2:latest   gns3      Running         Running 4 minutes ago
root@gns3:~#
```

Figure 1: Docker Swarm service creation in the blue network

Capabilities

We performed a series of tests in order to understand the behavior of a Docker Swarm service's capabilities, namely its self-healing, load balancing and isolation via overlay.

For the self-healing testing, we started by stopping one of ub2's service containers. The result, as seen in Figure 2, is that ub3 immediately created a container for the same replica of the service. The active services before shutdown, after shutdown and after restarting the stopped container in ub2 can be observed in Figure 3.

R1 ub1 ub2 ub3

```
root@gns3:~# docker container ls
CONTAINER ID   IMAGE                                COMMAND          CREATED          STATUS          PORTS     NAMES
2d963fc27862   ruivaladas/ws-simple-2:latest        "./ws_simple"    5 minutes ago    Up 5 minutes              bluesvc.5.nftxvhlwr3jb5oq3grybxapfl
1b7d8529afa9   ruivaladas/ws-simple-2:latest        "./ws_simple"    16 minutes ago   Up 15 minutes             bluesvc.3.fd9iv33mzuwkjpprhowatixx3
39ba1aed02e5   alpine                               "/bin/sh"        22 minutes ago   Up 22 minutes             r3
6638308ad7fe   alpine                               "/bin/sh"        22 minutes ago   Up 7 minutes              b3
root@gns3:~#
```

Figure 2: Docker containers on ub3 after bluesvc.5 container's shutdown in ub2

```
root@gns3:~# docker service ps bluesvc
ID             NAME            IMAGE                                NODE      DESIRED STATE   CURRENT STATE            ERROR                              PORTS
afxijgg48chw   bluesvc.1       ruivaladas/ws-simple-2:latest        gns3      Running         Running 7 minutes ago
zpyg3upcs6xh    \_ bluesvc.1   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 12 minutes ago  "No such image: ruivaladas/ws-…"
jif2aqc2to7y   bluesvc.2       ruivaladas/ws-simple-2:latest        gns3      Running         Running 11 minutes ago
fd9iv33mzuwk   bluesvc.3       ruivaladas/ws-simple-2:latest        gns3      Running         Running 9 minutes ago
0bxq3aht55rv    \_ bluesvc.3   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 12 minutes ago  "No such image: ruivaladas/ws-…"
qu3mu2z617nt   bluesvc.4       ruivaladas/ws-simple-2:latest        gns3      Running         Running 7 minutes ago
eg41m1is7ycz   bluesvc.5       ruivaladas/ws-simple-2:latest        gns3      Running         Running 11 minutes ago
root@gns3:~# docker service ps bluesvc
ID             NAME            IMAGE                                NODE      DESIRED STATE   CURRENT STATE            ERROR                              PORTS
afxijgg48chw   bluesvc.1       ruivaladas/ws-simple-2:latest        gns3      Running         Running 8 minutes ago
zpyg3upcs6xh    \_ bluesvc.1   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 13 minutes ago  "No such image: ruivaladas/ws-…"
jif2aqc2to7y   bluesvc.2       ruivaladas/ws-simple-2:latest        gns3      Running         Running 12 minutes ago
fd9iv33mzuwk   bluesvc.3       ruivaladas/ws-simple-2:latest        gns3      Running         Running 10 minutes ago
0bxq3aht55rv    \_ bluesvc.3   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 13 minutes ago  "No such image: ruivaladas/ws-…"
qu3mu2z617nt   bluesvc.4       ruivaladas/ws-simple-2:latest        gns3      Running         Running 8 minutes ago
nftxvhlwr3jb   bluesvc.5       ruivaladas/ws-simple-2:latest        gns3      Running         Ready 4 seconds ago
eg41m1is7ycz    \_ bluesvc.5   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Failed 5 seconds ago     "task: non-zero exit (2)"
root@gns3:~# docker service ps bluesvc
ID             NAME            IMAGE                                NODE      DESIRED STATE   CURRENT STATE            ERROR                              PORTS
afxijgg48chw   bluesvc.1       ruivaladas/ws-simple-2:latest        gns3      Running         Running 8 minutes ago
zpyg3upcs6xh    \_ bluesvc.1   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 13 minutes ago  "No such image: ruivaladas/ws-…"
jif2aqc2to7y   bluesvc.2       ruivaladas/ws-simple-2:latest        gns3      Running         Running 13 minutes ago
fd9iv33mzuwk   bluesvc.3       ruivaladas/ws-simple-2:latest        gns3      Running         Running 11 minutes ago
0bxq3aht55rv    \_ bluesvc.3   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 13 minutes ago  "No such image: ruivaladas/ws-…"
qu3mu2z617nt   bluesvc.4       ruivaladas/ws-simple-2:latest        gns3      Running         Running 9 minutes ago
nftxvhlwr3jb   bluesvc.5       ruivaladas/ws-simple-2:latest        gns3      Running         Running 19 seconds ago
eg41m1is7ycz    \_ bluesvc.5   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Failed 29 seconds ago    "task: non-zero exit (2)"
root@gns3:~#
```

Figure 3: Reaction to failure and restart of bluesvc.5

Afterwards, we tried to observe what would happen if an entire node crashed. For that end, we removed ub3 from the Docker Swarm (result in Figure 4) and observed that ub2 had inherited all of ub3's service-related containers (Figure 5). Even when the latter was re-added to the Swarm, it didn't recover its previous service containers and ub2 kept them permanently instead.

```
root@gns3:~# docker service ps bluesvc
ID             NAME            IMAGE                                NODE      DESIRED STATE   CURRENT STATE            ERROR                              PORTS
afxijgg48chw   bluesvc.1       ruivaladas/ws-simple-2:latest        gns3      Running         Running 18 minutes ago
zpyg3upcs6xh    \_ bluesvc.1   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 23 minutes ago  "No such image: ruivaladas/ws-…"
jif2aqc2to7y   bluesvc.2       ruivaladas/ws-simple-2:latest        gns3      Running         Running 22 minutes ago
gei4dbgdfczg   bluesvc.3       ruivaladas/ws-simple-2:latest        gns3      Running         Running 38 seconds ago
fd9iv33mzuwk    \_ bluesvc.3   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Running 20 minutes ago
0bxq3aht55rv    \_ bluesvc.3   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 23 minutes ago  "No such image: ruivaladas/ws-…"
qu3mu2z617nt   bluesvc.4       ruivaladas/ws-simple-2:latest        gns3      Running         Running 18 minutes ago
wkmosxic1xcx   bluesvc.5       ruivaladas/ws-simple-2:latest        gns3      Running         Running 38 seconds ago
nftxvhlwr3jb    \_ bluesvc.5   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Running 9 minutes ago
eg41m1is7ycz    \_ bluesvc.5   ruivaladas/ws-simple-2:latest        gns3      Shutdown        Failed 10 minutes ago    "task: non-zero exit (2)"
root@gns3:~#
```

Figure 4: Node shutdown in this case ub3

```
root@gns3:~# docker container ls
CONTAINER ID   IMAGE                                COMMAND          CREATED              STATUS              PORTS     NAMES
11f25da308e6   ruivaladas/ws-simple-2:latest        "./ws_simple"    About a minute ago   Up About a minute             bluesvc.5.wkmosxic1xcxmlwnilnr3m1gj
1ab3fe0c23c0   ruivaladas/ws-simple-2:latest        "./ws_simple"    About a minute ago   Up About a minute             bluesvc.3.gei4dbgdfczg1bf3x9bsyh0b5
13efd18f2b21   ruivaladas/ws-simple-2:latest        "./ws_simple"    23 minutes ago       Up 10 minutes                 bluesvc.5.eg41m1is7yczceco47uuo24c3
937c87946e0e   ruivaladas/ws-simple-2:latest        "./ws_simple"    23 minutes ago       Up 23 minutes                 bluesvc.2.jif2aqc2to7yn6kp9hxoafvpx
eba1c4ea6523   alpine                               "/bin/sh"        28 minutes ago       Up 28 minutes                 r2
48225a96301b   alpine                               "/bin/sh"        28 minutes ago       Up 28 minutes                 b2
root@gns3:~#
```

Figure 5: Reaction to the node shutdown

To test load balancing we used a simple Python script that accesses the HTTP service provided by the swarm 10 times. From what we can see in Figure 6, we hit the same sequence of 5 container IDs twice. We've also verified that these IDs don't belong to only one node, instead some of the IDs are from ub1's containers and the others are from ub2's containers (Figures 7 and 8).

```
root@ruivaladas-saar-tools-1:/home# python3 script.py 20.0.0.1 3000 10
You've hit 1ab3fe0c23c0
You've hit 11f25da308e6
You've hit 2d2d591fd77d
You've hit b9d4f0625a70
You've hit 937c87946e0e
You've hit 1ab3fe0c23c0
You've hit 11f25da308e6
You've hit 2d2d591fd77d
You've hit b9d4f0625a70
You've hit 937c87946e0e
root@ruivaladas-saar-tools-1:/home#
```

Figure 6: Load balancing test with a script

```
root@gns3:~# docker container ls
CONTAINER ID   IMAGE                                COMMAND          CREATED          STATUS          PORTS     NAMES
2d2d591fd77d   ruivaladas/ws-simple-2:latest        "./ws_simple"    26 minutes ago   Up 26 minutes             bluesvc.1.afxijgg48chw6lvit1gyiyn11
b9d4f0625a70   ruivaladas/ws-simple-2:latest        "./ws_simple"    26 minutes ago   Up 26 minutes             bluesvc.4.qu3mu2z617ntguuye8i6qlr37
90bb3ee76171   alpine                               "/bin/sh"        36 minutes ago   Up 36 minutes             r1
2f614895e4f9   alpine                               "/bin/sh"        37 minutes ago   Up 37 minutes             b1
root@gns3:~#
```

Figure 7: ub1 container IDs

```
root@gns3:~# docker container ls
CONTAINER ID   IMAGE                                COMMAND          CREATED          STATUS          PORTS     NAMES
11f25da308e6   ruivaladas/ws-simple-2:latest        "./ws_simple"    3 minutes ago    Up 3 minutes              bluesvc.5.wkmosxic1xcxmlwnilnr3m1gj
1ab3fe0c23c0   ruivaladas/ws-simple-2:latest        "./ws_simple"    3 minutes ago    Up 3 minutes              bluesvc.3.gei4dbgdfczg1bf3x9bsyh0b5
13efd18f2b21   ruivaladas/ws-simple-2:latest        "./ws_simple"    25 minutes ago   Up 12 minutes             bluesvc.5.eg41m1is7yczceco47uuo24c3
937c87946e0e   ruivaladas/ws-simple-2:latest        "./ws_simple"    25 minutes ago   Up 25 minutes             bluesvc.2.jif2aqc2to7yn6kp9hxoafvpx
eba1c4ea6523   alpine                               "/bin/sh"        30 minutes ago   Up 30 minutes             r2
48225a96301b   alpine                               "/bin/sh"        31 minutes ago   Up 31 minutes             b2
root@gns3:~#
```

Figure 8: ub2 container IDs

For the isolation via overlay testing we decided to first scale down the services to only 3 replicas (Figure 9 shows which replicas remained). Then, we used the same Python script from before to send 3 HTTP packets to ub1, whose service is listening on port 3000.

```
root@gns3:~# docker service ps bluesvc
ID             NAME             IMAGE                                NODE      DESIRED STATE   CURRENT STATE            ERROR                              PORTS
afxijgg48chw   bluesvc.1        ruivaladas/ws-simple-2:latest        gns3      Running         Running 29 minutes ago
zpyg3upcs6xh    \_ bluesvc.1    ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 33 minutes ago   "No such image: ruivaladas/ws-…"
jif2aqc2to7y   bluesvc.2        ruivaladas/ws-simple-2:latest        gns3      Running         Running 33 minutes ago
gei4dbgdfczg   bluesvc.3        ruivaladas/ws-simple-2:latest        gns3      Running         Running 11 minutes ago
fd9iv33mzuwk    \_ bluesvc.3    ruivaladas/ws-simple-2:latest        gns3      Shutdown        Running 31 minutes ago
0bxq3aht55rv    \_ bluesvc.3    ruivaladas/ws-simple-2:latest        gns3      Shutdown        Rejected 33 minutes ago   "No such image: ruivaladas/ws-…"
nftxvhlwr3jb   bluesvc.5        ruivaladas/ws-simple-2:latest        gns3      Shutdown        Running 20 minutes ago
eg41m1is7ycz    \_ bluesvc.5    ruivaladas/ws-simple-2:latest        gns3      Shutdown        Failed 20 minutes ago     "task: non-zero exit (2)"
root@gns3:~#
```

Figure 9: Scaling service down to 3 replicas

233	21.994313	20.0.0.3	20.0.0.1	HTTP	100	GET / HTTP/1.1
235	21.994513	10.0.0.2	10.0.0.17	HTTP	150	GET / HTTP/1.1
242	22.167350	10.0.0.17	10.0.0.2	HTTP	116	HTTP/1.0 200 OK (text/html)
245	22.167866	20.0.0.1	20.0.0.3	HTTP	66	HTTP/1.0 200 OK (text/html)
259	22.974954	20.0.0.3	20.0.0.1	HTTP	100	GET / HTTP/1.1
262	23.083248	20.0.0.1	20.0.0.3	HTTP	66	HTTP/1.0 200 OK (text/html)
275	23.975691	20.0.0.3	20.0.0.1	HTTP	100	GET / HTTP/1.1
277	23.975876	10.0.0.2	10.0.0.7	HTTP	150	GET / HTTP/1.1
282	24.085268	10.0.0.7	10.0.0.2	HTTP	116	HTTP/1.0 200 OK (text/html)
284	24.085634	20.0.0.1	20.0.0.3	HTTP	66	HTTP/1.0 200 OK (text/html)

Figure 10: Service load distribution behavior

As we can see from Figure 10, when the master node distributes work to the worker nodes, that communication is done using VXLAN and the VXLAN used is always the same (Figures 11 and 12). As such, the IP addresses used in those same communications are the ones for the node's ingress interfaces (Figure 13).

```
> Internet Protocol Version 4, Src: 20.0.0.1, Dst: 21.0.0.1
> User Datagram Protocol, Src Port: 49825, Dst Port: 4789
v Virtual eXtensible Local Area Network
    > Flags: 0x0800, VXLAN Network ID (VNI)
      Group Policy ID: 0
      VXLAN Network Identifier (VNI): 4096
      Reserved: 0
> Ethernet II, Src: 02:42:0a:00:00:02 (02:42:0a:00:00:02), Ds
> Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.17
```

Figure 11: VXLAN from ub1 to ub3

```
> Internet Protocol Version 4, Src: 20.0.0.1, Dst: 20.0.0.2
> User Datagram Protocol, Src Port: 51364, Dst Port: 4789
v Virtual eXtensible Local Area Network
    > Flags: 0x0800, VXLAN Network ID (VNI)
      Group Policy ID: 0
      VXLAN Network Identifier (VNI): 4096
      Reserved: 0
> Ethernet II, Src: 02:42:0a:00:00:02 (02:42:0a:00:00:02), D
> Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.7
```

Figure 12: VXLAN from ub1 to ub2

Figure 13: VXLAN VNI and ingress interface IP-MAC for ub1, ub2 and ub3 respectively

We also created another service with 3 replicas in the red network in order to test out if the services could operate isolated from one another (Figure 14). After running the Python script for the third time, we saw that the results were the exact same as the ones in Figure 10, which means the service from the blue network is completely isolated from the service in the red network.

Figure 14: Docker Swarm service creation in the red network