PROTOCOLES RÉSEAU - PARTIE I

les mains dans le cambouis

Dernière mise à jour : 7 janvier 2024

Franck de Goër - teaching[at]udtq[.]fr

INTRODUCTION

Objectif du cours

Pour vous

Comprendre/apprendre des trucs en (sécurité) réseau.

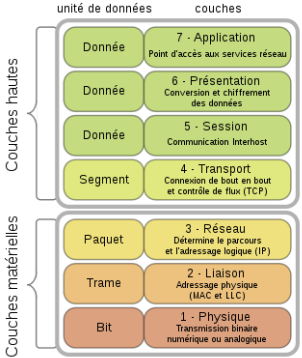
Manipuler.

Poser des questions.

Plan sur trois jours

- couches basses : protocoles et attaques
- couches applicatives : notions, attaques
- manipulation (configuration, fabrication de paquets avec scapy, attaques)
- un peu de scan
- (bonus) outils : masscan, masscanned, xdp, etc.

Les couches réseau



[https://fr.wikipedia.org/wiki/Mod%C3%A8le_OSI]

Les couches réseau

Les couches réseau des livres de réseau (ou sur Wikipédia) : modèle OSI

Les couches réseau

Les couches réseau des livres de réseau (ou sur Wikipédia) : modèle OSI

- **liaison** : *adressage physique*

Les couches réseau

Les couches réseau des livres de réseau (ou sur Wikipédia) : modèle OSI

- **liaison** : *adressage physique*
- **réseau** : *détermine le parcours des données et l'adressage logique*

Les couches réseau

Les couches réseau des livres de réseau (ou sur Wikipédia) : modèle OSI

- **liaison** : *adressage physique*
- **réseau** : *détermine le parcours des données et l'adressage logique*
- **transport** : *connexion de bout en bout, connectabilité et contrôle de flux*

Les couches réseau

Les couches réseau des livres de réseau (ou sur Wikipédia) : modèle OSI

- **liaison** : *adressage physique*
- **réseau** : *détermine le parcours des données et l'adressage logique*
- **transport** : *connexion de bout en bout, connectabilité et contrôle de flux*
- **...**

Les couches réseau

Les couches réseau des livres de réseau (ou sur Wikipédia) : modèle OSI

- **liaison** : *adressage physique*
- **réseau** : *détermine le parcours des données et l'adressage logique*
- **transport** : *connexion de bout en bout, connectabilité et contrôle de flux*
- **...**
- **application** : *point d'accès aux services réseau*

Les couches réseau

Les couches réseau par protocole :

- **liaison** :

Les couches réseau

Les couches réseau par protocole :

- **liaison** : *Ethernet*
- **réseau** :

Les couches réseau

Les couches réseau par protocole :

- **liaison** : *Ethernet*
- **réseau** : *IPv4, IPv6*
- **transport** :

Les couches réseau

Les couches réseau par protocole :

- **liaison** : *Ethernet*
- **réseau** : *IPv4, IPv6*
- **transport** : *TCP, UDP, SCTP*
- **...**
- **application** :

Les couches réseau

Les couches réseau par protocole :

- **liaison** : *Ethernet*
- **réseau** : *IPv4, IPv6*
- **transport** : *TCP, UDP, SCTP*
- **...**
- **application** : *HTTP, SMB, FTP*

Les couches réseau

Les couches réseau **en pratique** : *qui parle à qui ?*

- **liaison** : *des machines connectées à un même réseau*

Les couches réseau

Les couches réseau **en pratique** : *qui parle à qui ?*

- **liaison** : *des machines connectées à un même réseau*
- **réseau** : *des machines de réseaux différents*

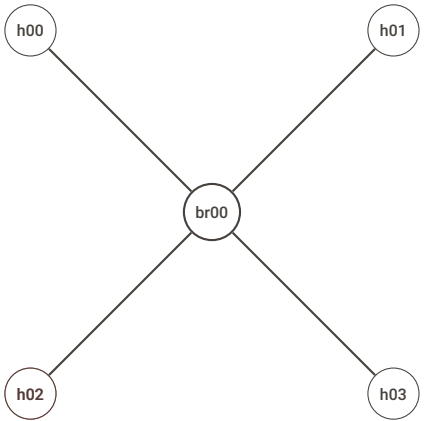
Les couches réseau

Les couches réseau **en pratique** : *qui parle à qui ?*

- **liaison** : *des machines connectées à un même réseau*
- **réseau** : *des machines de réseaux différents*
- **transport** : *des applications*

COUCHE LIAISON & MÉCANISMES ASSOCIÉS

Contexte



Adresses

Adresses MAC

MAC (*Media Access Control*)

Les adresses de la couche liaison sont appelées adresses **MAC**.

- liée à une interface réseau
- interface physique : fournie par le constructeur
- interface physique/logique : peut être changée par le système

Format

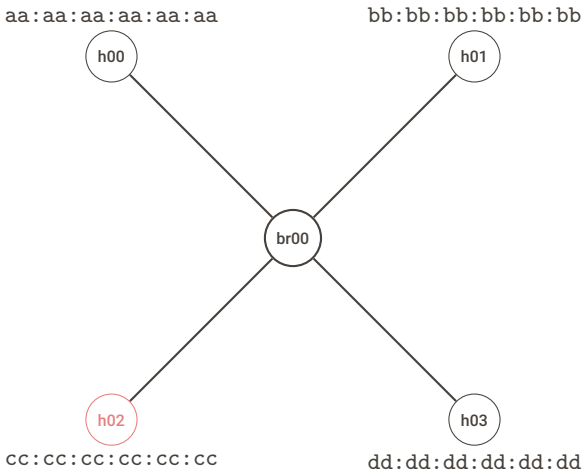
- **taille** : 48 bits (6 octets)
- **représentation** : `hh:hh:hh:hh:hh:hh` (exemple : `6e:3a:f6:4c:1f:63`)

```
$ ip link show enp0s31f6
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode DEFAULT group
      default qlen 1000
    link/ether 6e:3a:f6:4c:1f:63 brd ff:ff:ff:ff:ff:ff

# ip link set enp0s31f6 addr aa:bb:cc:dd:ee:ff

$ ip link show enp0s31f6
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN mode DEFAULT group
      default qlen 1000
    link/ether aa:bb:cc:dd:ee:ff brd ff:ff:ff:ff:ff:ff
```

Exemple


aa:aa:aa:aa:aa:aa
bb:bb:bb:bb:bb:bb
h00
h01
br00
h02
h03
cc:cc:cc:cc:cc:cc
dd:dd:dd:dd:dd:dd

Ethernet

Format

Ethernet est **le** protocole à connaitre de la couche liaison.

MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)
6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets
← 64–1522 octets →					

[https://en.wikipedia.org/wiki/Ethernet_frame]

Subtilité - le champ **EtherType** :

- valeur **[0, 1500]** : taille de la donnée (802.3)
- valeur **[1536, +]** : type de la trame (Ethernet II)
- entre les deux... ?

Longueur ou type ?

3.2.6 Length/Type field

This two-octet field takes one of two meanings, depending on its numeric value. For numerical evaluation, the first octet is the most significant octet of this field.

a) If the value of this field is less than or equal to 1500 decimal (05DC hexadecimal), then the Length/Type field indicates the number of MAC client data octets contained in the subsequent MAC Client Data field of the basic frame (Length interpretation).

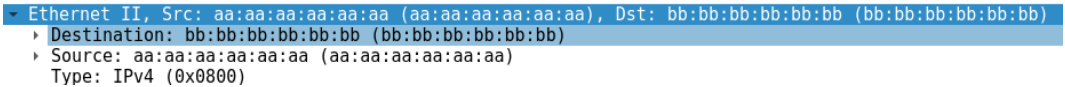
b) If the value of this field is greater than or equal to 1536 decimal (0600 hexadecimal), then the Length/Type field indicates the Ethertype of the MAC client protocol (Type interpretation).[28]
The Length and Type interpretations of this field are mutually exclusive.

[IEEE Std 802.3-2018, IEEE Standard for Ethernet - section 3.2.6]

Quelques types

EtherType	Description
`0x0800`	Internet Protocol version 4 (IPv4)
`0x0806`	Address Resolution Protocol (ARP)
`0x8100`	VLAN-tagged frame (IEEE 802.1Q)
`0x86DD`	Internet Protocol Version 6 (IPv6)
`0x888E`	EAP over LAN (IEEE 802.1X)

Wireshark


Ethernet II, Src: aa:aa:aa:aa:aa:aa (aa:aa:aa:aa:aa:aa), Dst: bb:bb:bb:bb:bb:bb (bb:bb:bb:bb:bb:bb)
Destination: bb:bb:bb:bb:bb:bb (bb:bb:bb:bb:bb:bb)
Source: aa:aa:aa:aa:aa:aa (aa:aa:aa:aa:aa:aa)
Type: IPv4 (0x0800)

Scapy

```
>>> Ether()
<Ether  |>
>>> Ether() / IP()
<Ether  type=IPv4 |<IP  |>>
>>> Ether() / IPv6()
<Ether  type=IPv6 |<IPv6  |>>
>>> Ether() / ARP()
<Ether  type=ARP |<ARP  |>>
>>> Ether(dst=ETHER_BROADCAST, src="aa:aa:aa:aa:aa:aa")
<Ether  dst=ff:ff:ff:ff:ff:ff src=aa:aa:aa:aa:aa:aa |>
>>> ls(_)
dst        : DestMACField                    = 'ff:ff:ff:ff:ff:ff' ('None')
src        : SourceMACField                  = 'aa:aa:aa:aa:aa:aa' ('None')
type       : XShortEnumField                 = 36864           ('36864')
>>> raw(Ether(dst=ETHER_BROADCAST, src="aa:aa:aa:aa:aa:aa"))
b'\xff\xff\xff\xff\xff\xff\xaa\xaa\xaa\xaa\xaa\xaa\x90\x00'
>>> (Ether() / ARP()).pdfdump()
```

ARP

Introduction

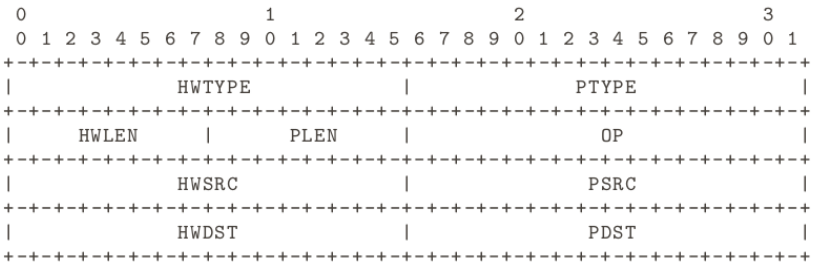
Un *paquet* a, la plupart du temps, besoin de deux adresses de destination :

- une adresse `MAC` (couche **liaison**)
- une adresse `IP` (couche **réseau**)

Address Resolution Protocol (ARP)

Le rôle d'`ARP` est de trouver l'adresse `MAC` correspondant à une adresse `IP`.

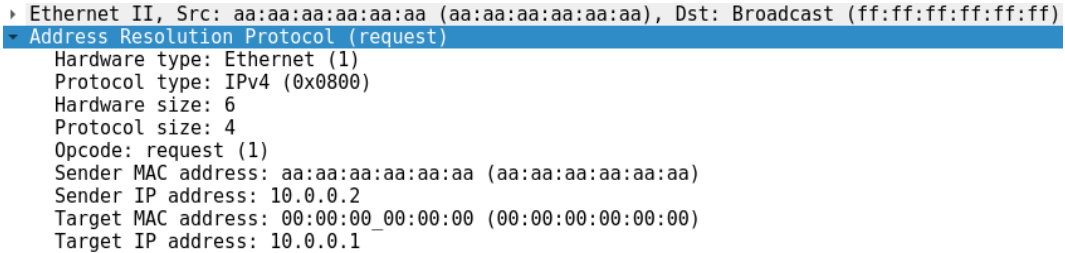
Format



RFC 826

- **`hwtype`** : type d'adresse **`hwsrc`**, **`hwdst`** (*e.g.*, Ethernet)
- **`ptype`** : type d'adresse **`psrc`**, **`pdst`** (*e.g.*, IPv4 ou IPv6)
- **`op`** : opération (*e.g.*, **`who-is`** ou **`is-at`**)

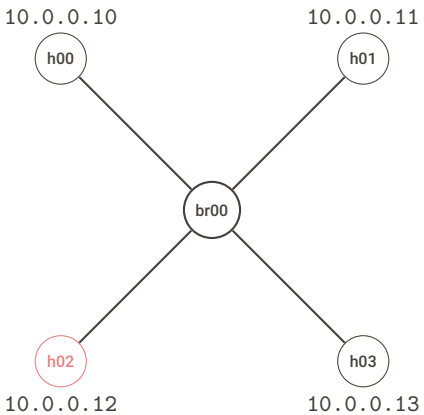
Wireshark


Ethernet II, Src: aa:aa:aa:aa:aa:aa (aa:aa:aa:aa:aa:aa), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
Hardware type: Ethernet (1)
Protocol type: IPv4 (0x0800)
Hardware size: 6
Protocol size: 4
Opcode: request (1)
Sender MAC address: aa:aa:aa:aa:aa:aa (aa:aa:aa:aa:aa:aa)
Sender IP address: 10.0.0.2
Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
Target IP address: 10.0.0.1

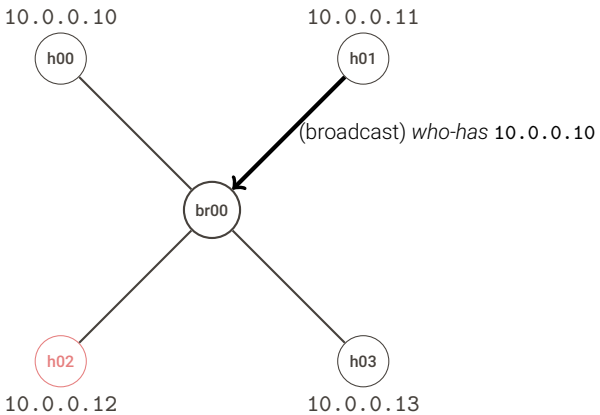
Scapy

```
>>> Ether(dst=ETHER_BROADCAST) / ARP(op=1, pdst="10.0.0.1")
<Ether  dst=ff:ff:ff:ff:ff:ff type=ARP |<ARP  op=who-has pdst=10.0.0.1 |>>
>>> ls(ARP)
hwtype     : XShortField                          = ('1')
ptype      : XShortEnumField                      = ('2048')
hwlen      : FieldLenField                        = ('None')
plen       : FieldLenField                        = ('None')
op         : ShortEnumField                       = ('1')
hwsrc      : MultipleTypeField (SourceMACField, StrFixedLenField) = ('None')
psrc       : MultipleTypeField (SourceIPField, SourceIP6Field, StrFixedLenField) = ('None')
hwdst      : MultipleTypeField (MACField, StrFixedLenField) = ('None')
pdst       : MultipleTypeField (IPField, IP6Field, StrFixedLenField) = ('None')
```

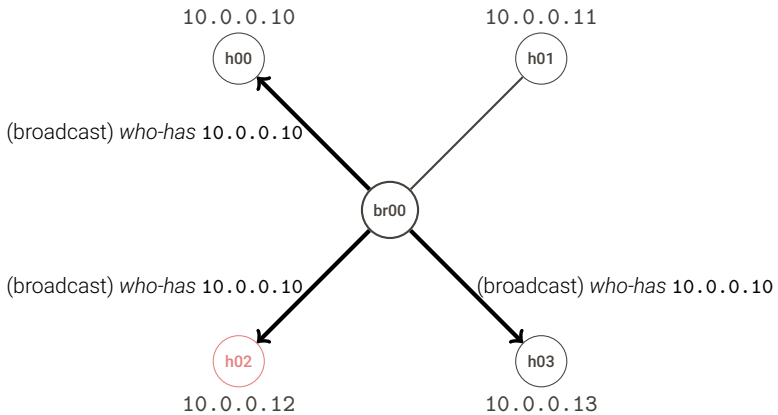
Exemple



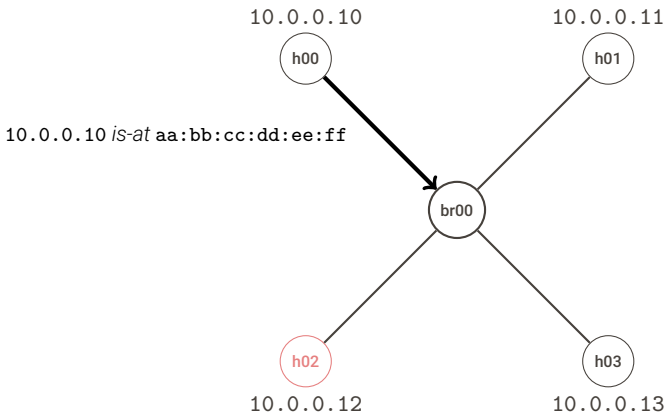
Exemple


10.0.0.10
10.0.0.11
h00
h01
(broadcast) *who-has* 10.0.0.10
br00
h02
h03
10.0.0.12
10.0.0.13

Exemple


10.0.0.10
h00
10.0.0.11
h01
(broadcast) who-has 10.0.0.10
br00
(broadcast) who-has 10.0.0.10
(broadcast) who-has 10.0.0.10
h02
h03
10.0.0.12
10.0.0.13

Exemple


10.0.0.10
10.0.0.11
h00
h01
10.0.0.10 is-at aa:bb:cc:dd:ee:ff
br00
h02
h03
10.0.0.12
10.0.0.13

Exemple

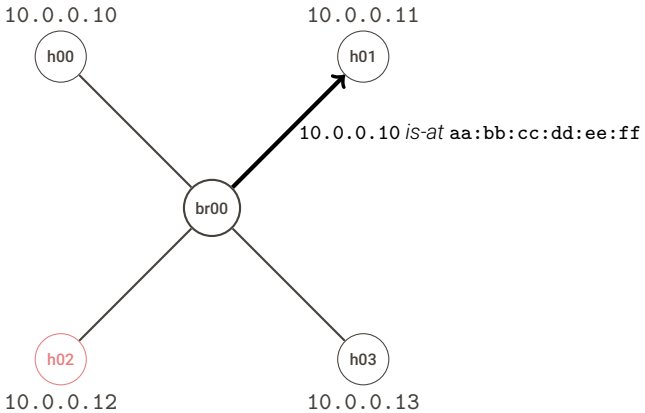

10.0.0.10
10.0.0.11
h00
h01
10.0.0.10 is-at aa:bb:cc:dd:ee:ff
br00
h02
h03
10.0.0.12
10.0.0.13

Table ARP

Une machine conserve les résolutions **ARP** de ses voisins dans une table **ARP**.

- voir la table **ARP** :

```
$ ip neigh show
10.0.0.1 dev eth0 lladdr bb:bb:bb:bb:bb:bb REACHABLE
```

- vider la table **ARP** :

```
# ip neigh flush all
```

Switch

Fonctionnement d'un switch/commutateur

- à chaque port du switch est branché zéro, un ou plusieurs équipements (via un autre switch)
- garde en mémoire une table associative **(port, @MAC [1])** appelée CAM [2]
- table utilisée pour savoir sur quel port (re)transmettre une trame

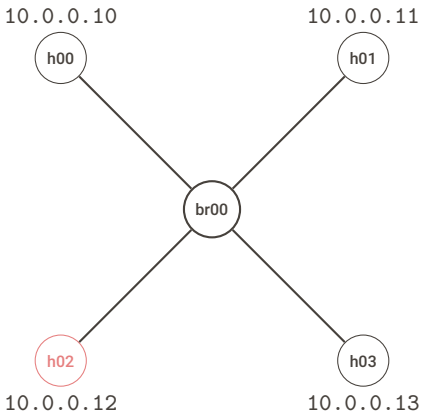
```
switch# show mac address-table
          Mac Address Table
------------------------------------------

Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    1cdb.5a46.8382    DYNAMIC     Et0/2
   1    5ac1.22d9.617d    DYNAMIC     Et0/0
   1    d905.09d7.6276    DYNAMIC     Et0/0
   1    e37b.ea1b.0b50    DYNAMIC     Et0/0
Total Mac Addresses for this criterion: 4
```

1. `MAC` : Media Access Control
2. CAM : Content-Addressable Memory

Manipulation

Le « lab »



Mise en place du « lab »

- récupérer/recopier `utils.sh`, `setup_lab_datalink.sh`
- exécuter `setup_lab_datalink.sh`
- vérifier

```
$ sudo su
# mkdir /root/lab
# cd /root/lab
# curl https://static.udtq.fr/UP78/lab.tgz | tar xzvf -
# ./setup_lab_datalink.sh
# ip netns show
lab_br00
lab_h03
lab_h02
lab_h01
lab_h00
```

Mise en place du « lab »

- éditer le fichier `/root/.bashrc` (ou `/root/.zshrc` pour Kali) et ajouter la ligne à la fin : `source /root/lab/utils.sh`
- vérifier **dans un nouveau terminal**

```
# enter_ns h00
h00> ip a
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: veth0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether ae:03:4c:ec:c5:34 brd ff:ff:ff:ff:ff:ff link-netns lab_br
    inet 10.0.0.10/24 scope global veth0
       valid_lft forever preferred_lft forever
    inet6 fe80::ac03:4cff:feec:c534/64 scope link
       valid_lft forever preferred_lft forever
h00> ping h01
```

Prise en main

- sniffer sur `h01` et `h02`
- envoyer un ping depuis `h00` vers `h01`
- observer les paquets capturés sur `h01` et `h02`
- inspecter les tables `ARP` de `h00`, `h01` et `h02`
- forger et envoyer avec scapy depuis `h02` vers `h00` :
  - un paquet `ARP who-has`
  - un paquet `ARP is-at`
- inspecter la table `ARP` sur `h00`
- (bonus) faire une fonction python (scapy) qui prend une IP en paramètre et qui retourne son adresse MAC (faire la résolution ARP « à la main » avec scapy, extraire le contenu)

Illustrations du cours

- sniffer sur `h01` et `h02`

```
h01> tcpdump -enli veth0
h02> tcpdump -enli veth0
```

- envoyer un ping depuis `h00` vers `h01`

```
h00> ping h01
```

- observer les paquets capturés sur `h01` et `h02`
- inspecter les tables `ARP` de `h00`, `h01` et `h02`

```
h0X> arp -a
```

- forger et envoyer avec scapy depuis `h02` vers `h00` :
  - un paquet `ARP who-has`

```
>>> sendp(Ether(dst=ETHER_BROADCAST)/ARP(op=1, pdst="10.0.0.10"), iface="veth0")
```

  - un paquet `ARP is-at`

```
>>> sendp(Ether(dst=ETHER_BROADCAST)/ARP(op=2, hwsrc="aa:bb:cc:cc:bb:aa", psrc="10.0.0.12", pdst="
    10.0.0.10"), iface="veth0")
```

Exercice : Attaque MitM [3]

Setup : un terminal sur `h00`, un sur `h01`, trois sur `h02`.

`h00>` Lancer un ping vers `h01`
`h01>` Lancer un ping vers `h00`
`h02>` Capturer le trafic
`h02>` Envoyer une requête `ARP` à `h00` depuis l'`IP` de `h01` et la `MAC` de `h02`
Que se passe-t-il ?
`h02>` Envoyer une requête `ARP` à `h01` depuis l'`IP` de `h00` et la `MAC` de `h02`
`h01>` Inspecter la table `ARP`
`h02>` Envoyer des ARP gratuits pour intercepter le trafic de `h00` vers `h01`
`h02>` Envoyer des ARP gratuits pour intercepter le trafic de `h01` vers `h00`

3. Man in the Middle

Scan

ARP

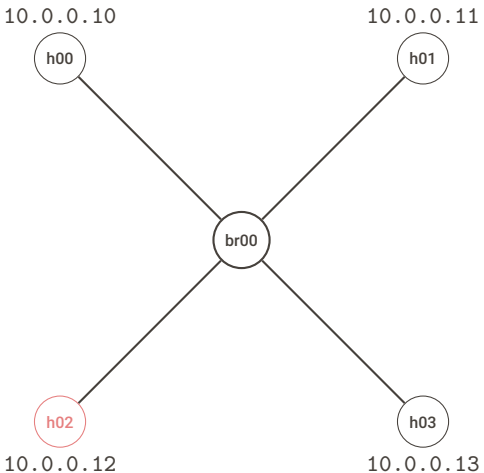
Objectif

Découvrir des machines accessibles sur un LAN

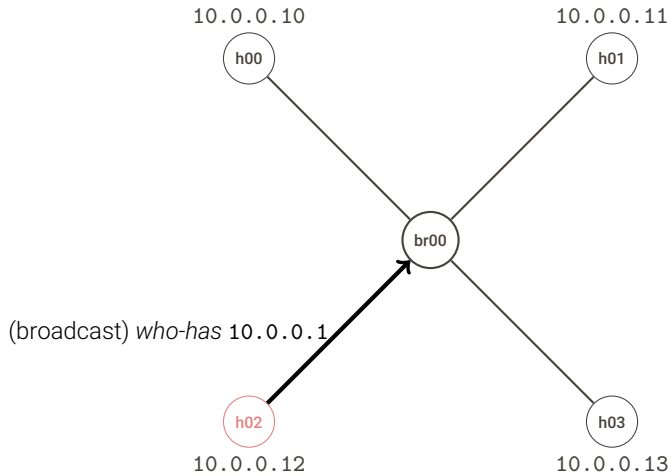
- **méthode :** envoi de requêtes `ARP`
- **outil :** `nmap -PR (-n -sn)` ou `arp-scan`

> The -P* options (which select ping types) can be combined. You can increase your odds of penetrating strict firewalls by sending many probe types using different TCP ports/flags and ICMP codes. **Also note that ARP/Neighbor Discovery (-PR) is done by default against targets on a local ethernet network even if you specify other -P* options, because it is almost always faster and more effective.**

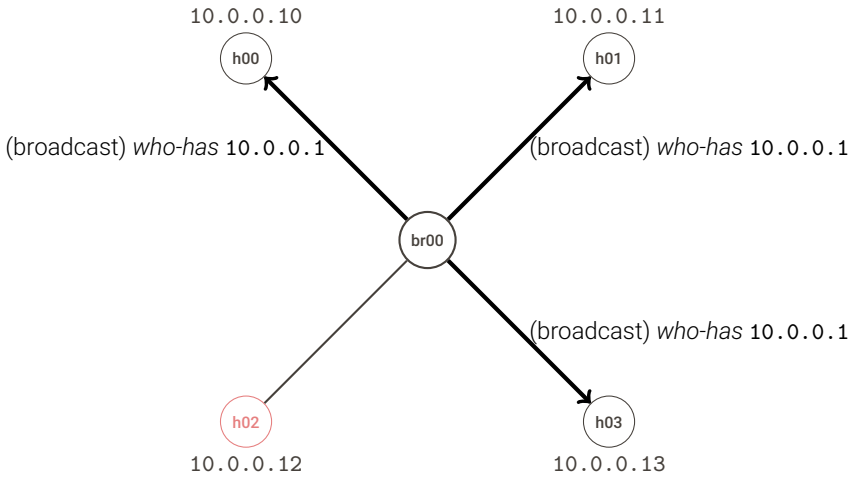
ARP - Exemple



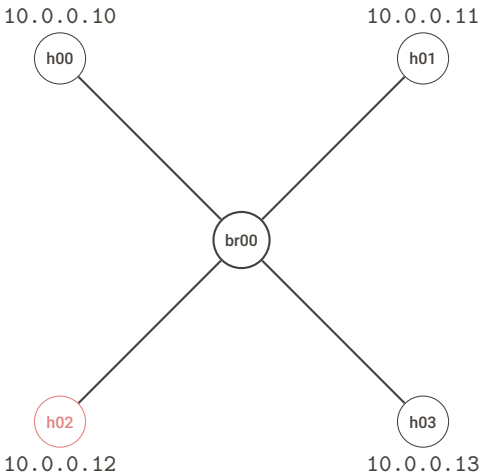
ARP - Exemple


10.0.0.10
h00
10.0.0.11
h01
br00
(broadcast) *who-has* 10.0.0.1
h02
h03
10.0.0.12
10.0.0.13

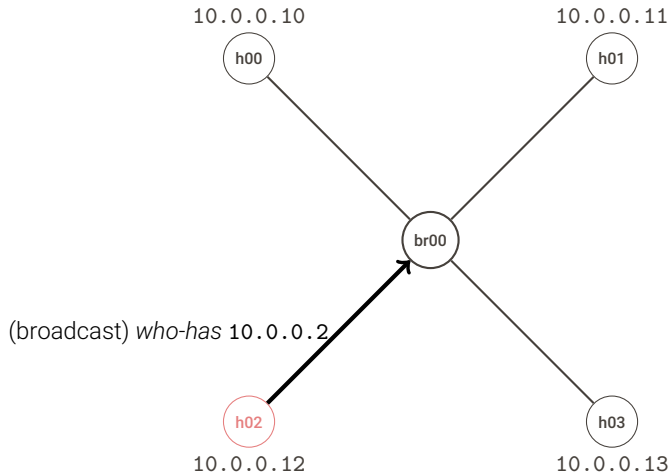
ARP - Exemple


10.0.0.10
h00
10.0.0.11
h01
(broadcast) who-has 10.0.0.1
(broadcast) who-has 10.0.0.1
br00
(broadcast) who-has 10.0.0.1
h02
h03
10.0.0.12
10.0.0.13

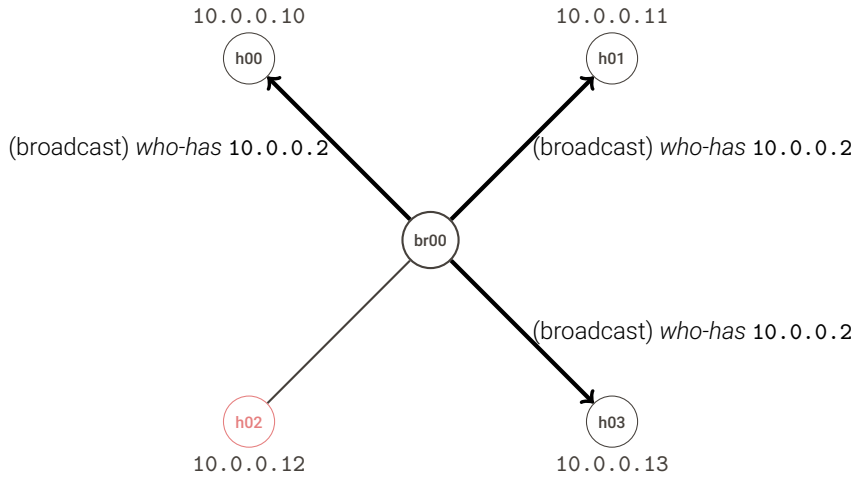
ARP - Exemple



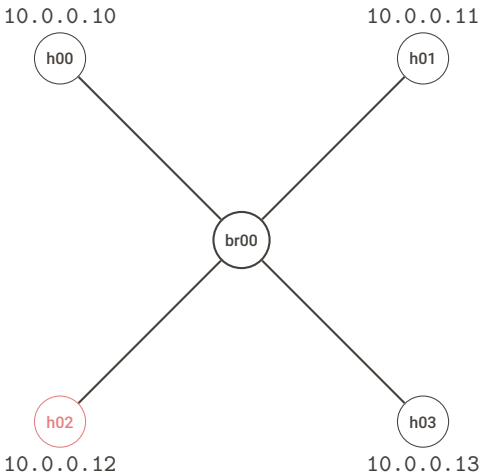
ARP - Exemple


10.0.0.10
h00
10.0.0.11
h01
br00
(broadcast) *who-has* 10.0.0.2
h02
h03
10.0.0.12
10.0.0.13

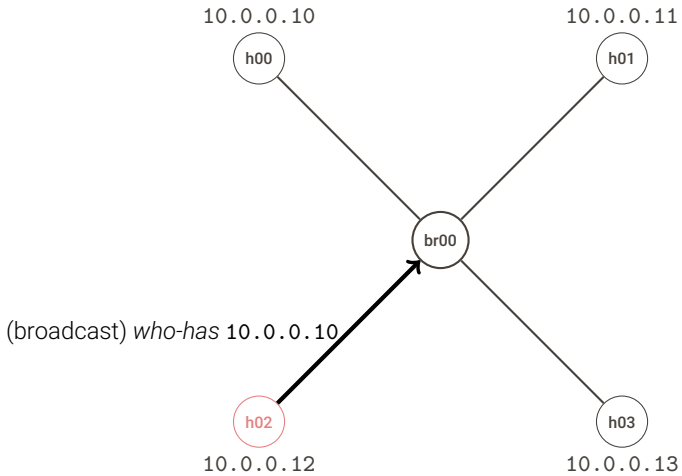
ARP - Exemple


10.0.0.10
h00
10.0.0.11
h01
(broadcast) who-has 10.0.0.2
(broadcast) who-has 10.0.0.2
br00
(broadcast) who-has 10.0.0.2
h02
h03
10.0.0.12
10.0.0.13

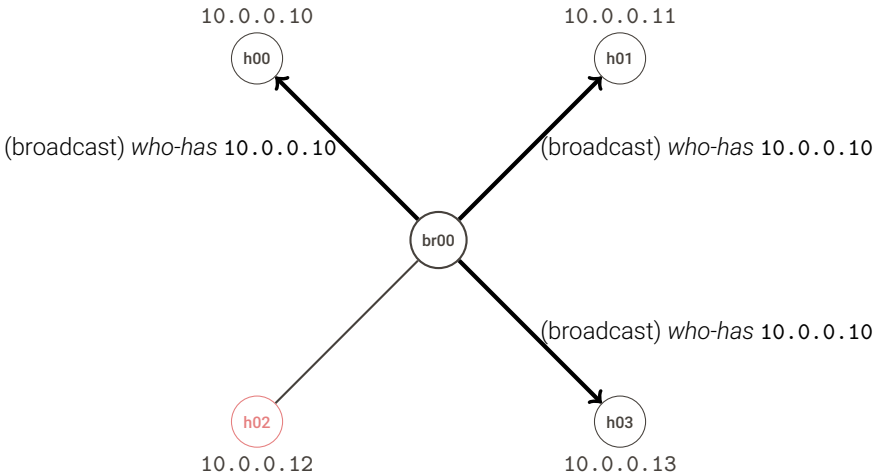
ARP - Exemple



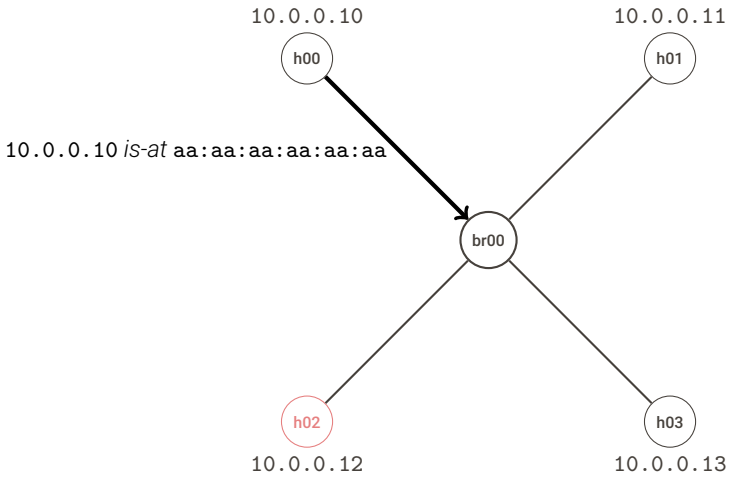
ARP - Exemple


10.0.0.10
h00
10.0.0.11
h01
br00
(broadcast) *who-has* 10.0.0.10
h02
10.0.0.12
h03
10.0.0.13

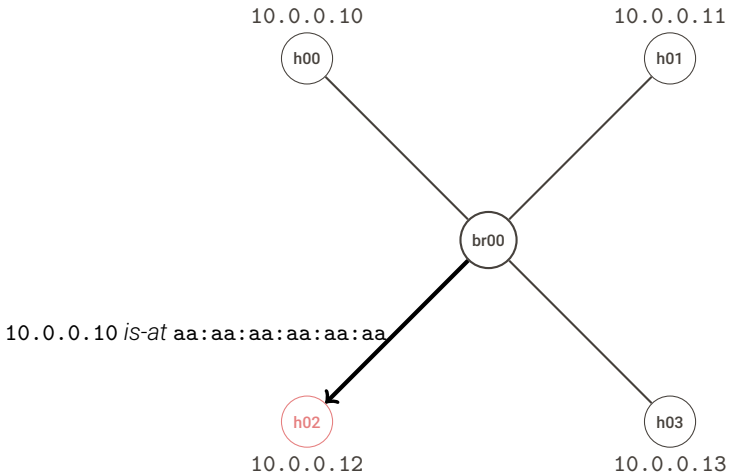
ARP - Exemple


10.0.0.10
h00
10.0.0.11
h01
(broadcast) who-has 10.0.0.10
(broadcast) who-has 10.0.0.10
br00
(broadcast) who-has 10.0.0.10
h02
h03
10.0.0.12
10.0.0.13

ARP - Exemple


10.0.0.10
10.0.0.11
h00
h01
10.0.0.10 is-at aa:aa:aa:aa:aa:aa
br00
h02
h03
10.0.0.12
10.0.0.13

ARP - Exemple


10.0.0.10
10.0.0.11
h00
h01
br00
10.0.0.10 is-at aa:aa:aa:aa:aa:aa
h02
h03
10.0.0.12
10.0.0.13

Filtrage

arptables

arptables - administration tool for arp packet filtering

Permet de filtrer :

- sur les adresses `MAC` (source, dest.)
- sur les adresses de niveau 3 (source, dest.)
- sur le type d'adresses de niveau 3

Exemples

Quelques commandes :

- Bloquer l'ensemble des requêtes **ARP** en entrée :

```
# arptables -P INPUT DROP
```

- Bloquer les requêtes **ARP** provenant de l'adresse **MAC** `aa:aa:aa:aa:aa:aa` :

```
# arptables -A INPUT --source-mac aa:aa:aa:aa:aa:aa -j DROP
```

- Bloquer les requêtes **ARP** provenant de la plage d'adresses **IP** `10.0.0.0/24` :

```
# arptables -A INPUT -s 10.0.0.0/24 -j DROP
```

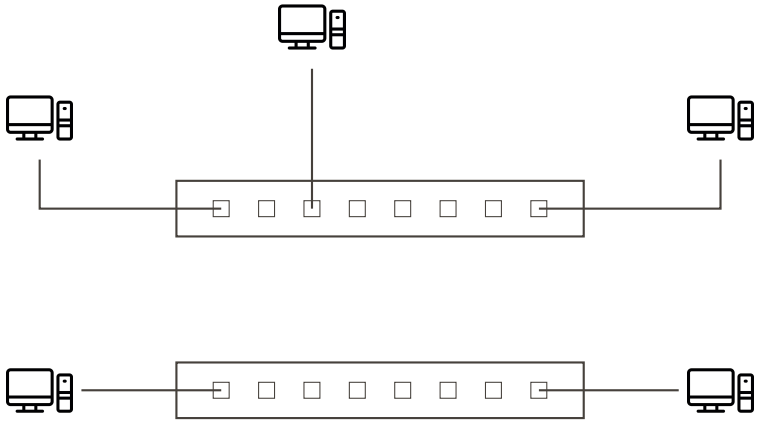
Exercice

`h00>` Récupérer l'adresse `MAC` de `h03`
`h00>` Interdire l'`IP` de `h02` et la `MAC` de `h03` à résoudre en `ARP`
`h01>` Flusher la table `ARP`, essayer de résoudre l'adresse `MAC` de `h00`
`h02>` Flusher la table `ARP`, essayer de résoudre l'adresse `MAC` de `h00`
Contourner la restriction
`h03>` Flusher la table `ARP`, essayer de résoudre l'adresse `MAC` de `h00`
Contourner la restriction

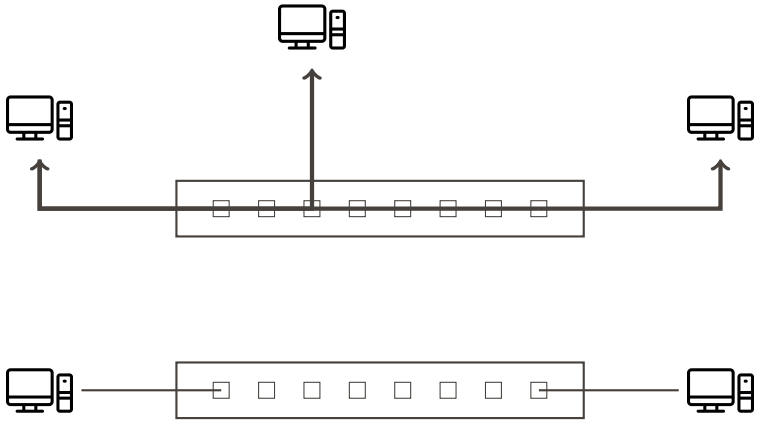
SEGMENTATION

VLAN

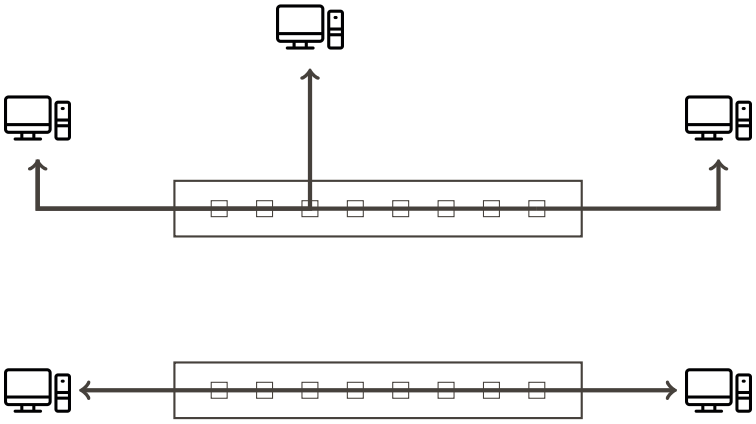
Contexte



Contexte



Contexte



VLAN

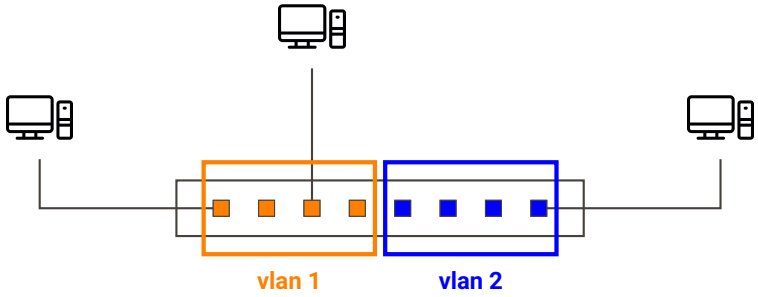
Deux fonctions (utiles !) du `VLAN` :

- faire **plusieurs** switchs avec **un** switch
- faire **un** switch avec **plusieurs** switchs

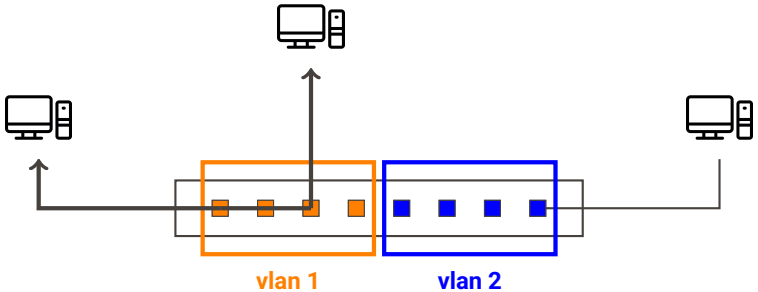
En un mot

Les `VLAN` permettent de séparer la topologie au niveau 2 du matériel (switchs) physique.

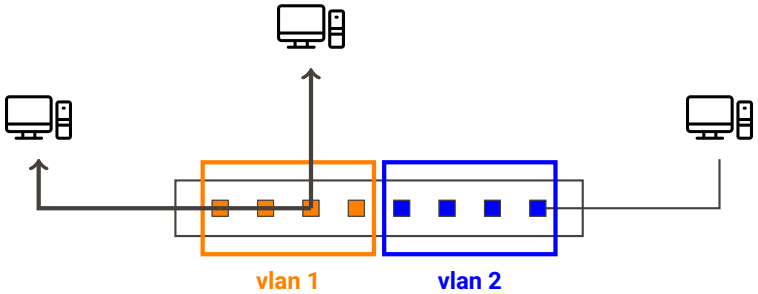
VLAN - plus avec moins



VLAN - plus avec moins

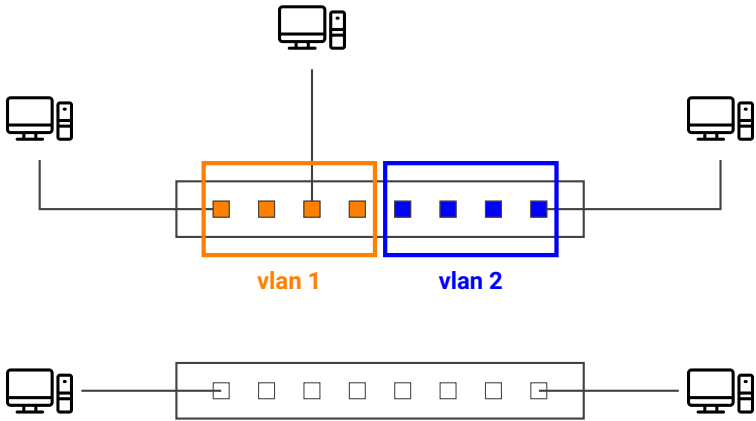


VLAN - plus avec moins

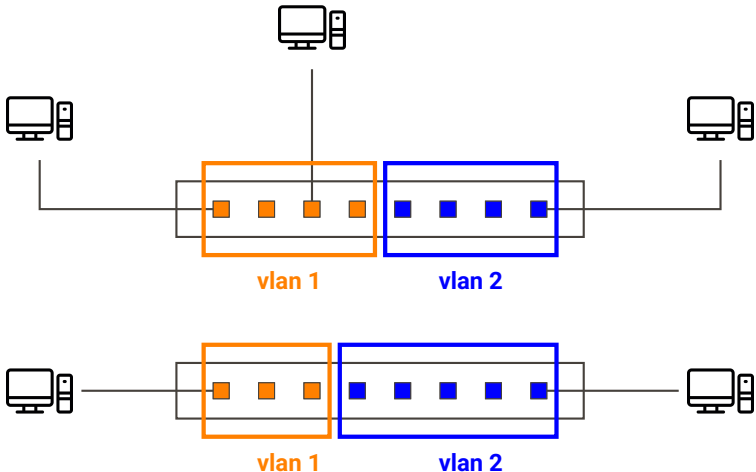


La machine de droite est désormais isolée au niveau 2.

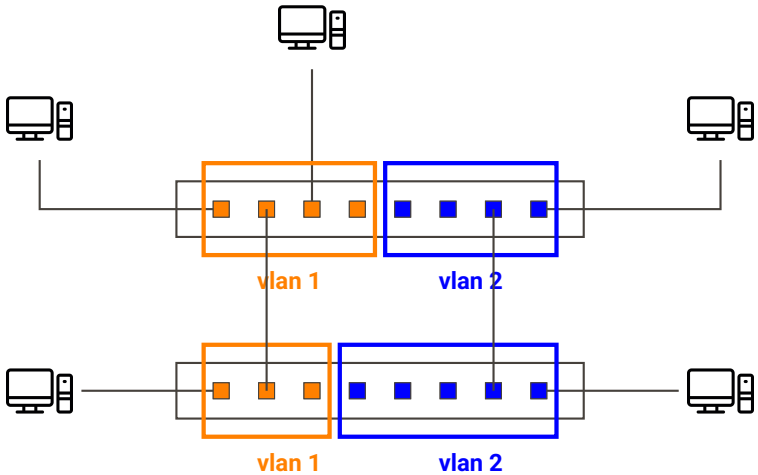
VLAN - moins avec plus



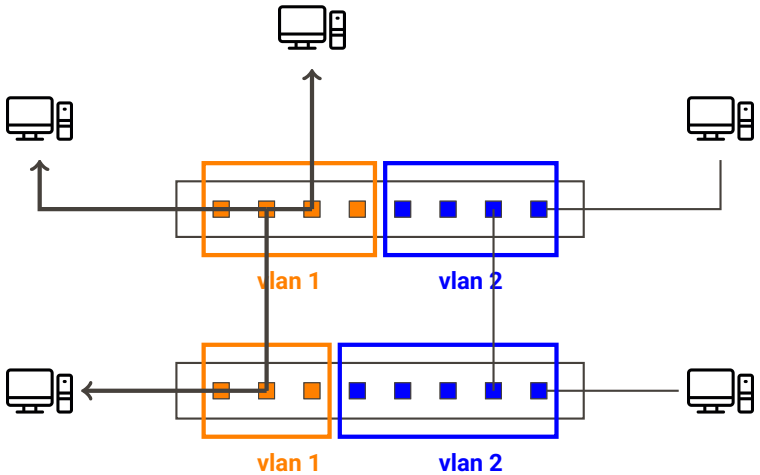
VLAN - moins avec plus



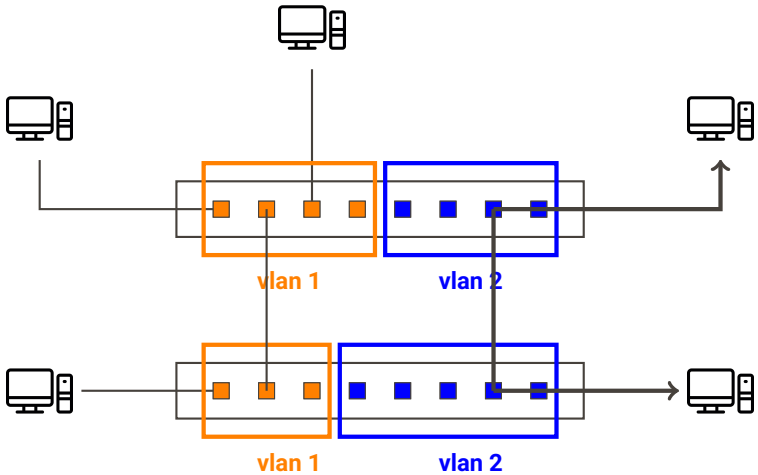
VLAN - moins avec plus



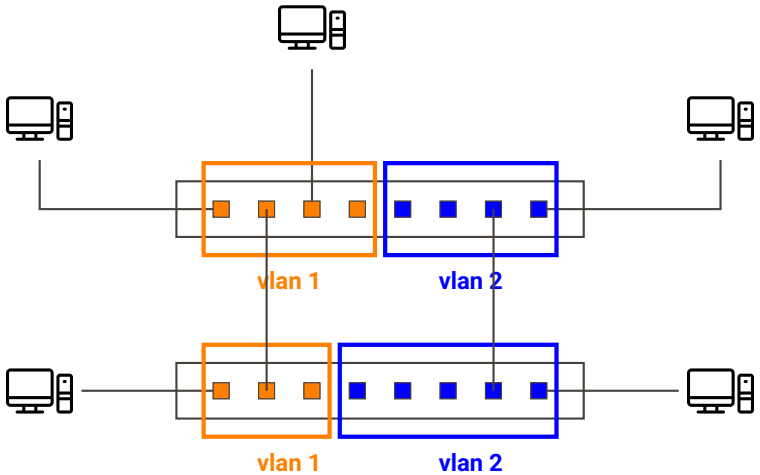
VLAN - moins avec plus



VLAN - moins avec plus

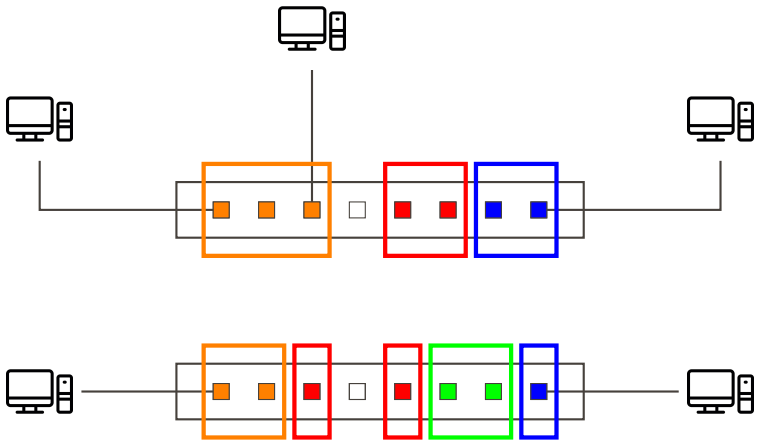


VLAN - moins avec plus

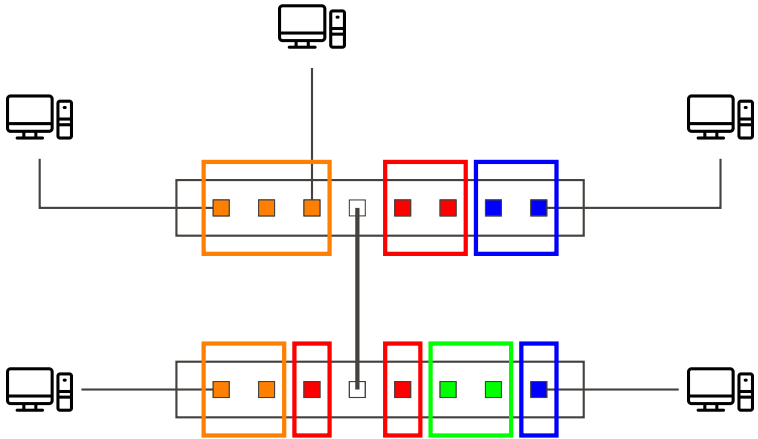


Jusqu'ici, pas besoin de tags (logique interne aux switchs).

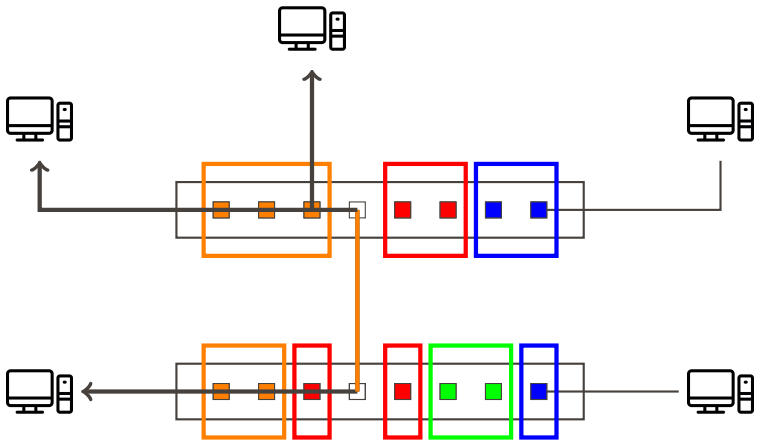
VLAN - le trunk



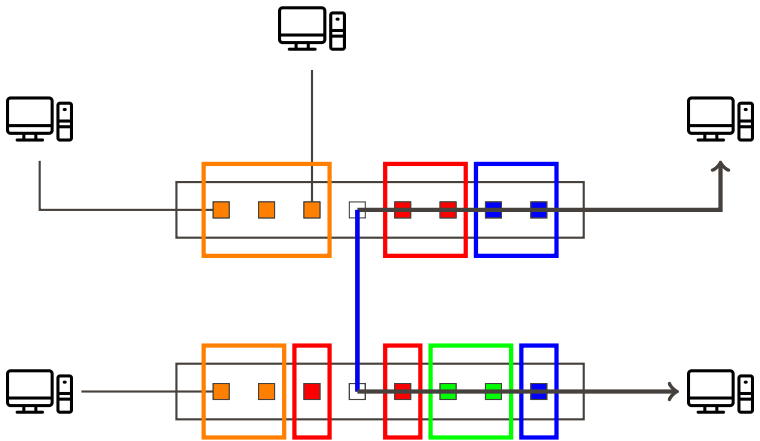
VLAN - le trunk



VLAN - le trunk



VLAN - le trunk



`VLAN` - Les tags

Les tags sont **nécessaires** sur les liens pluri-`VLAN`.

- les ports reliés à des machines n'ont pas besoin de tag (ports `untagged`, `access`)
- les ports entre switchs (ou entre switchs et routeurs) ont besoin de marquer les paquets (ports `tagged`, `trunk`)

Attention

- un tag est ajouté **uniquement** lorsqu'un paquet est transmis sur un port **tagged** (ou **trunk**)
- le tag est supprimé dès réception sur un port **tagged** (ou **trunk**)

`VLAN` natif

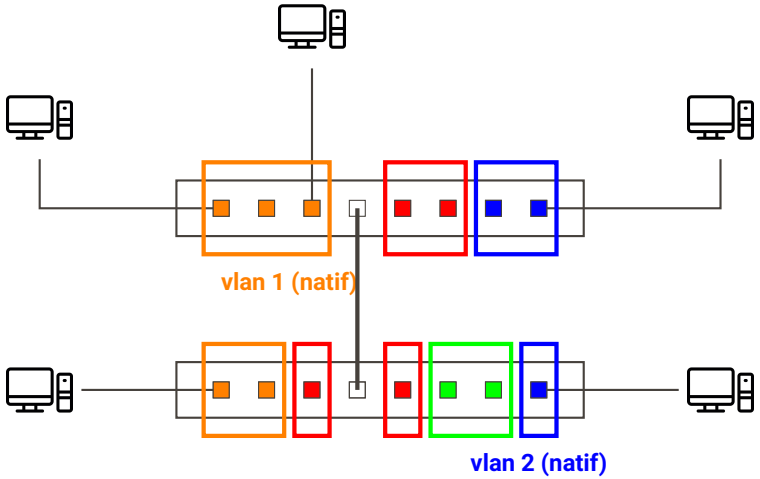
Le `VLAN` natif est utilisé dans deux cas :

- un paquet appartenant au `VLAN` natif envoyé sur un port **tagged** (ou **trunk**) **n'est pas tagué**
- un paquet qui arrive non tagué sur un port **tagged** (ou **trunk**) est tagué avec le `VLAN` natif.

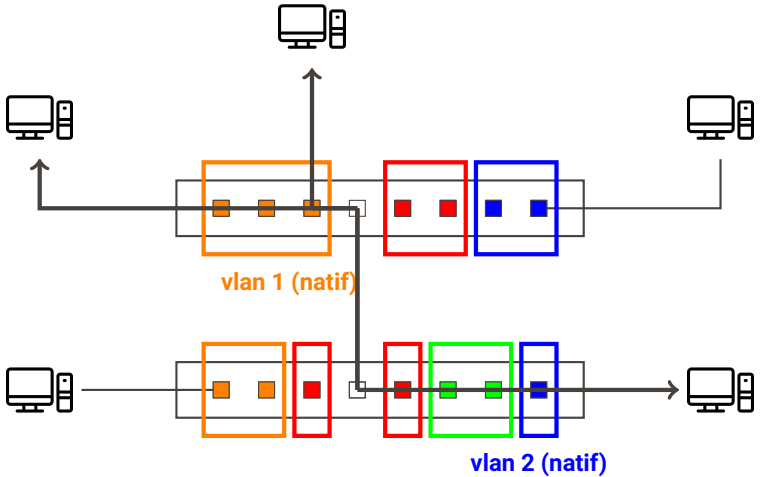
Attention

Il est crucial que les switchs interconnectés utilisent le même `VLAN` natif.

VLAN natif - exemple de problème



VLAN natif - exemple de problème



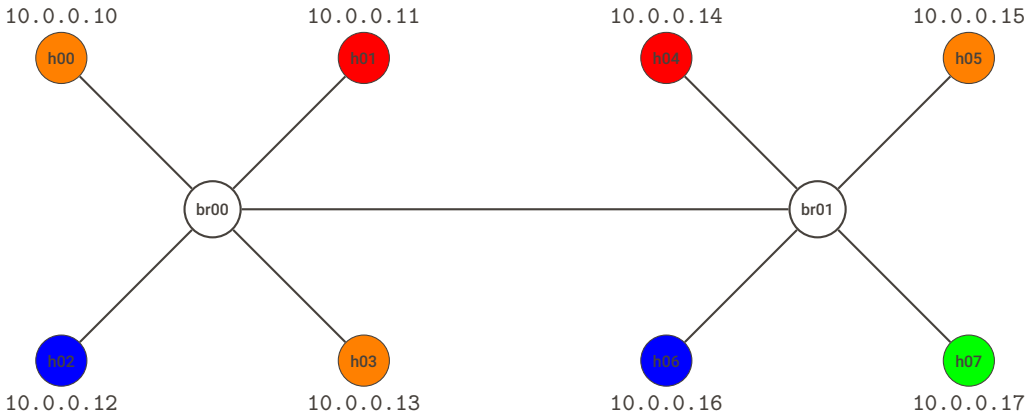
VLAN - Résumé

multiplicité	type de lien	dénomination
un seul VLAN	machine <> switch	untagged/access
plusieurs VLAN	switch <> switch	tagged/trunk

dénomination	type de lien	VLAN	paquet sortant
untagged/access	machine <> switch	*	non tagué
tagged/trunk	switch <> switch	natif	non tagué
tagged/trunk	switch <> switch	sauf natif	tagué

Manipulation

Le « lab »


10.0.0.10
h00
10.0.0.11
h01
10.0.0.14
h04
10.0.0.15
h05
br00
br01
h02
10.0.0.12
h03
10.0.0.13
h06
10.0.0.16
h07
10.0.0.17

Mise en place du « lab »

```
# cd /root/lab
# ./setup_lab_vlan.sh
# ip netns show
lab_br01
lab_h07
lab_h06
lab_h05
lab_h04
lab_br00
lab_h03
lab_h02
lab_h01
lab_h00
```

Illustrations du cours

`h00>` Faire un scan `ARP` sur `10.0.0.0/24`
Quelles machines appartiennent au même VLAN ?
`h00>` Lancer un ping vers `h05`
`h00>` Capturer le trafic sur `veth0`
`br00>` Capturer le trafic sur `veth_br01`
Qu'observe-t-on ?
`h07>` Capturer le trafic sur `veth0`
`br00>` Forger et envoyer avec scapy un paquet sur l'interface `veth_br01` à destination de `h07`
`h00>` Envoyer le même paquet depuis `h00`
Où le paquet s'arrête-t-il ?

Protected Port

VLAN extrêmiste

Idée

Isoler toutes les machines au niveau 2 (sauf la passerelle).

Avec les VLAN :

- un VLAN par machine
- un /31 (voire un /30) par machine
- une interface, une IP par machine sur le routeur
- limité à 4095 machines (ou alors 802.1ad - QinQ)

Protected port

Principe

Un port « protégé » ne peut pas parler avec un autre port « protégé ».

Si tous les ports « machine » sont marqués « protégés » $\Rightarrow$ les machines ne peuvent plus parler qu'avec la passerelle

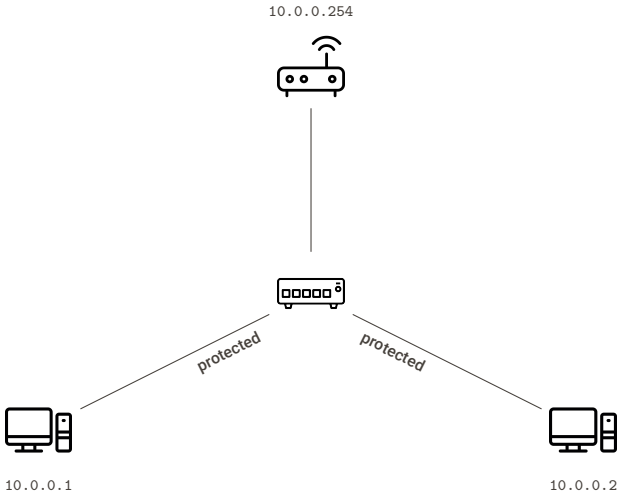
Conséquence

Plus d'attaques sur la couche liaison entre les machines.

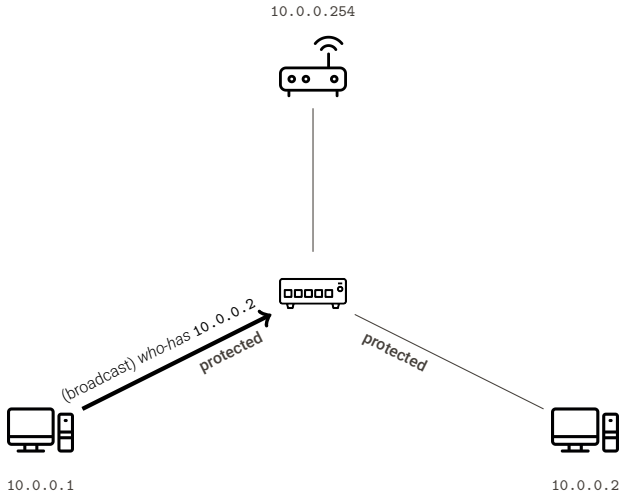
Protected port

- ***A protected port does not forward any traffic*** *(unicast, multicast, or broadcast)* ***to any other port that is also a protected port.*** *Data traffic cannot be forwarded between protected ports at Layer 2; only control traffic, such as PIM packets, is forwarded because these packets are processed by the CPU and forwarded in software. All data traffic passing between protected ports must be forwarded through a Layer 3 device.*
- ***Forwarding behavior between a protected port and a nonprotected port proceeds as usual.***

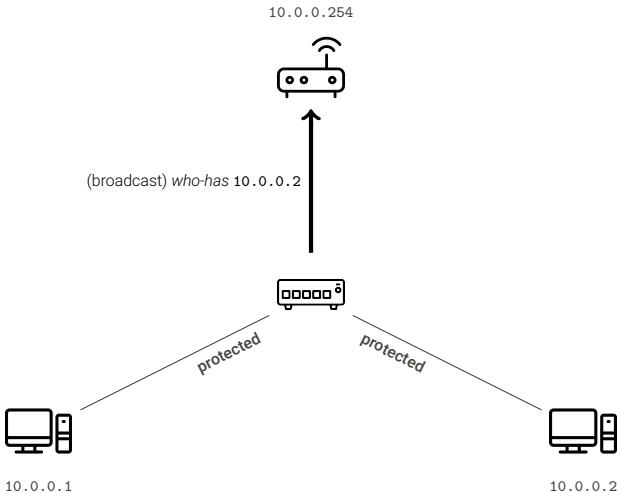
Exemple



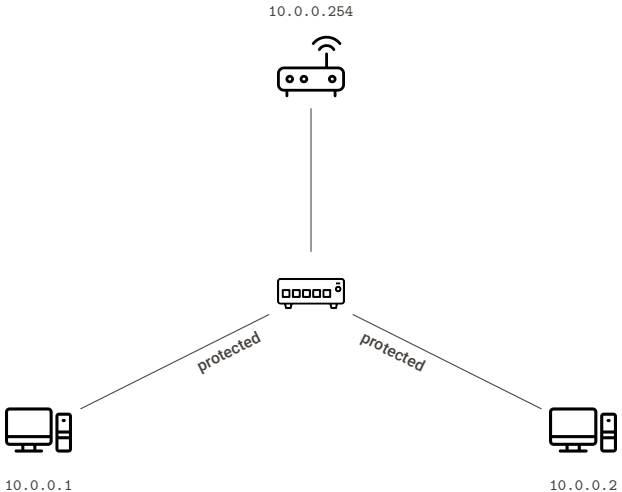
Exemple


10.0.0.254
(broadcast) who-has 10.0.0.2
protected
protected
10.0.0.1
10.0.0.2

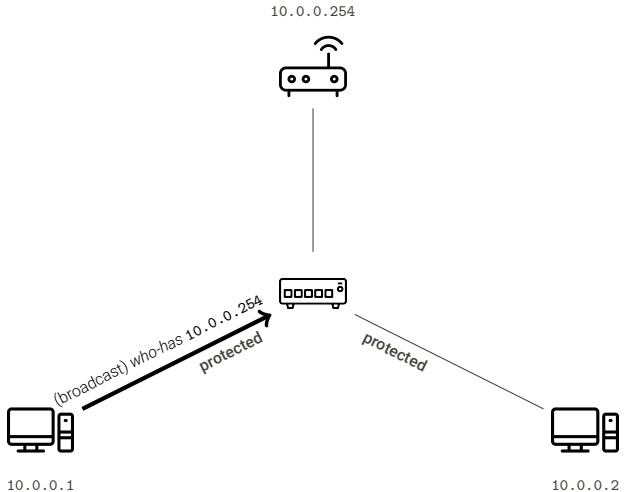
Exemple


10.0.0.254
(broadcast) *who-has* 10.0.0.2
protected
protected
10.0.0.1
10.0.0.2

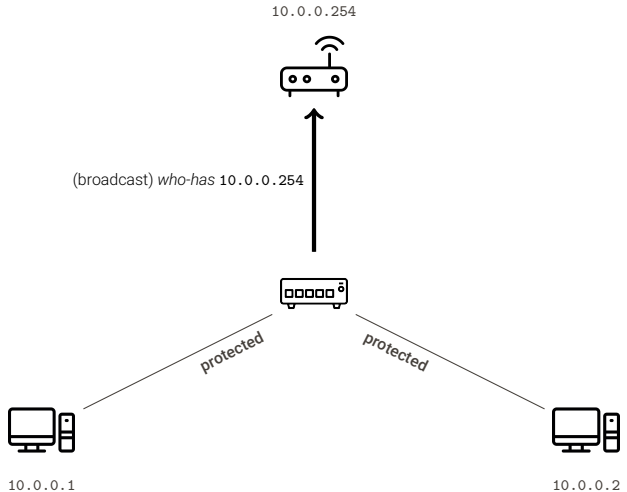
Exemple



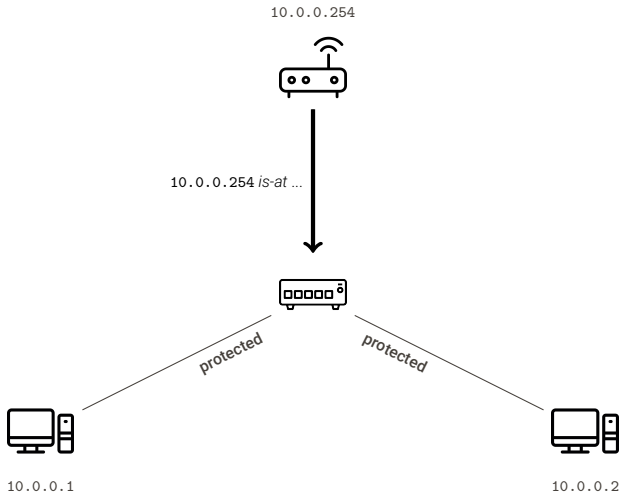
Exemple


10.0.0.254
(broadcast) who-has 10.0.0.254
protected
protected
10.0.0.1
10.0.0.2

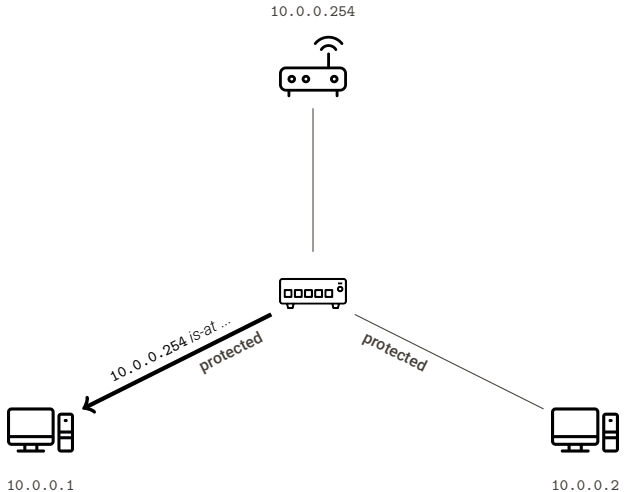
Exemple


10.0.0.254
(broadcast) *who-has* 10.0.0.254
protected
protected
10.0.0.1
10.0.0.2

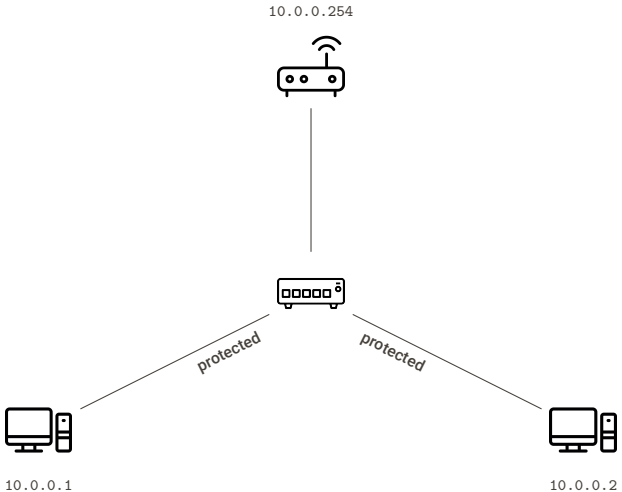
Exemple



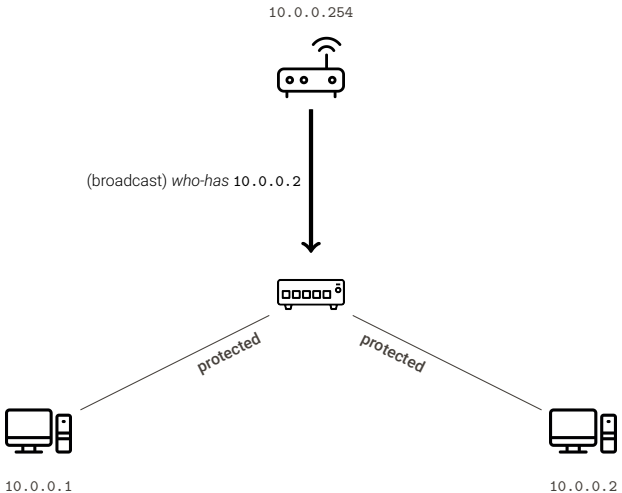
Exemple



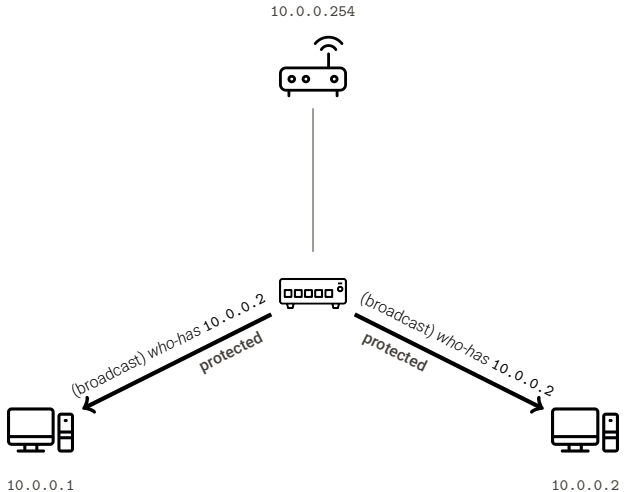
Exemple



Exemple


10.0.0.254
(broadcast) *who-has* 10.0.0.2
protected
protected
10.0.0.1
10.0.0.2

Exemple

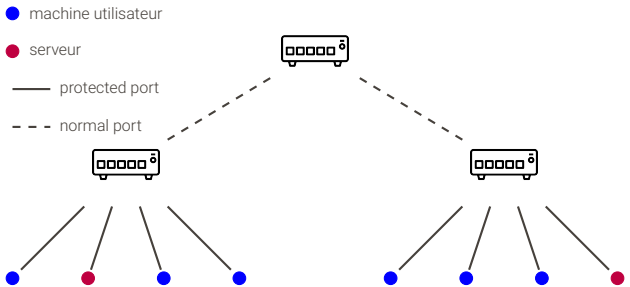

10.0.0.254
(broadcast) who-has 10.0.0.2
protected
(broadcast) who-has 10.0.0.2
protected
10.0.0.1
10.0.0.2

Limitation

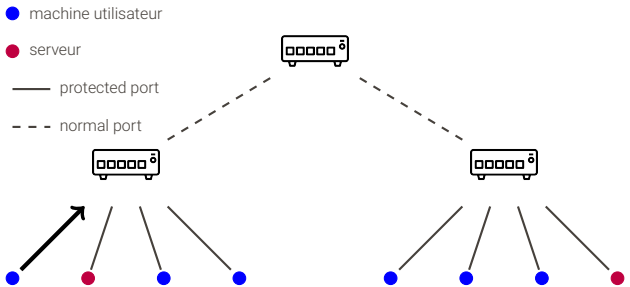
Problème

Ne permet pas l'isolation au-delà d'un seul switch.

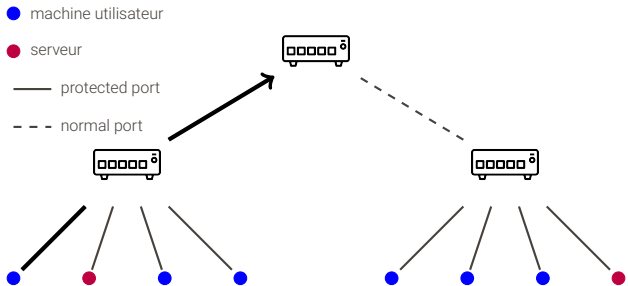
Limitation



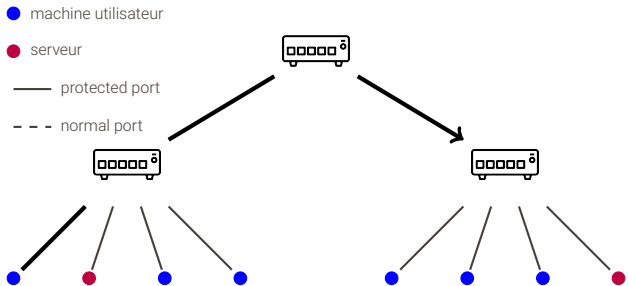
Limitation



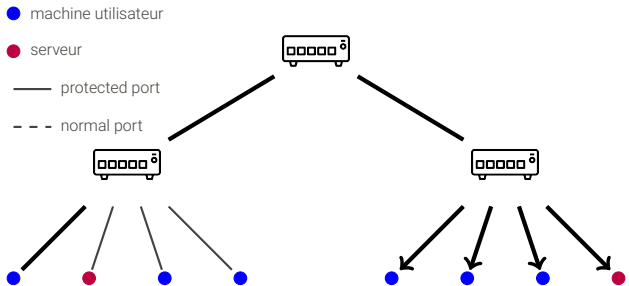
Limitation



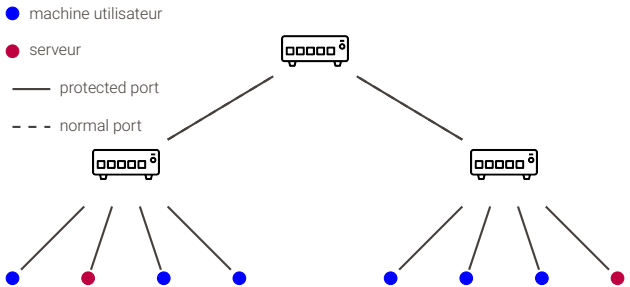
Limitation



Limitation



Limitation



OUTRODUCTION

QUESTIONS?