RYU: Lab 1 Application NAT

Objet	contrôleur RYU avec NAT
Niveau requis	débutant, avisé
Débutant, à savoir	ryu-overview
Suivi	:DRAFT:

Objectif

Cette implémentation utilise Ryu NAT (https://github.com/John-Lin/nat) pour effectuer un comportement NAT afin de comprendre les opérations NAT.

Topologie expérimentale

Télécharger le code NAT depuis Github

```
$ git clone https://github.com/John-Lin/nat
$ cd nat
```

Initialiser les paramètres

Avant d'iniliaser la configuration modifier les paramètres par défaut dans la méthode nat_config_init en fonction de l'environnement d'expérimentation dans `snat.py`:

```
393  @route('nat_setings_init', urls.url_nat_config_init, methods='POST')
394  def nat_config_init(self, req, **kwargs):
395      save-dict = {}
396
397      # Default settings modify these value to fit your requirements
393      save_dict['wan_port'] = 1
399      save_dict['nat_public_ip'] = IPAddress('140.116.71.178')
400      save_dict['default_gateway'] = IPAddress('140.114.71.254')
401
402      network = '192.168.8.0/24'
403      save_dict['ip_network'] = IPNetwork(network)
404      save_dict['dhcp_gw_addr'] = IPNetwork(network)[1]
405
406      save_dict['broadcast_addr'] = IPAddress('192.168.8.255')
407      save_dict['dns_addr'] = IPAddress('8.8.8.8')
403
409      save_dict['dhcp_hw_addr'] = '08.00:27.8:0f:gd'
410      save_dict['MAC_ON_WAN'] = '00:0e:c6:67:ag:f6'
411      save_dict['MAC_ON_LAN'] = '00:0e:c6:87:ag:fa'
```

Wan_port	Cela correspond au port du commutateur et est défini sur le port externe au NAT.
Nat_public_ip	NAT divulgué publiquement
Default_gateway	L'adresse de passerelle du même domaine que l'adresse IP publique NAT
Network	Domaine LAN dans NAT / 24 (plus masque)
Ip_network	Domaine LAN dans NAT
Dhcp_gw_addr	Cet auteur doit faire un dhcp simple, il y a donc une adresse de passerelle dhcp
Broadcast_addr	Adresse de diffusion utilisée par dhcp
Dns_addr	Adresse DNS
Dhcp_hw_addr	adresse MAC Dhcp
MAC_ON_WAN	adresse MAC WAN
MAC_ON_LAN	adresse MAC LAN

- Lancer Ryu NAT

```
$ ryu-manager base.py l2switch.py dhcp.py snat.py --verbose
```

- Initialiser Ryu NAT

Cela initialisera la configuration NAT

```
curl --noproxy xx.xx.xxx.xxx -X POST -d '
{}' http://xx.xx.xxx.xxx:8383/api/nat_config_init
```

Lancer Ryu NAT

```
$ ryu-manager base.py l2switch.py dhcp.py snat.py --verbose
```

Mettre à jour les paramètres NAT

Méthode	PUT
URI	/api/nat_config_save

Cela mettra à jour la configuration NAT

```
$ curl --noproxy xx.xx.xxx.xxx -X PUT  -d '
{
  "wanPort": 4,
  "natPublicIp": "xx.xx.xxx.xxx",
  "defaultGateway": "xx.xx.xxx.x",
  "natPrivateNetwork": "192.168.8.0"
}' http://xx.xx.xxx.xxx:8383/api/nat_config_save
```

Expérimentation

Essayer d'exécuter les instructions Ping sur l'hôte 2 à partir de l'hôte 1.

`Ping H1: $ ping 192.168.1.10`

On peux voir que Ping ne peut pas passer.

Analyse du code dans snat.py

Le traitement des paquets ICMP a été prévu mais qu'il n'a pas été implémenté:

```
292    @set_evcls (ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
293    def_packet_in_handler (self, ev).
294       msg = ev.msg
295       datapath = msg.datapath
296       # ofproto = datapath.ofproto
297       # parser = datapath.ofproto_parser
293       in_port = msg.match['rin_port']
299       pkt = packet.Packet(msg.data)
300
301       pkt_ethernet = pkt_protocol(ethernet.ethernet)
302       pkt_arp = pkt.get_protocol(arp.arp)
303       # pkt_icmp = pkt.get_protoco(icmp.icmp)
304       pkt_ip = pkt.get_protocol(ipv4.ipv4)
305       pkt_tcp = pkt.get_protocol(tcp.tcp)
306       pkt_udp = pkt.get_protocol(udp.udp)
```

```
        if pkt_tcp:
            # print "@@@ Install TCP Flow Entry @@@"
            tcp_src = pkt_tcp.src_port
            tcp_dst = pkt_tcp.dst_port

            match = parser.OFPMatch(in_port=in_port,
                                    eth_type=ether.ETH_TYPE_IP,
                                    ip_proto=inet.IPPROTO_TCP,
                                    ipv4_src=ipv4_src,
                                    ipv4_dst=ipv4_dst,
                                    tcp_src=tcp_src,
                                    tcp_dst=tcp_dst)

            actions =
[parser.OFPActionSetField(eth_dst=IP_TO_MAC_TABLE[target_ip]),
parser.OFPActionSetField(ipv4_src=self.nat_public_ip),
                         parser.OFPActionSetField(tcp_src=nat_port),
                         parser.OFPActionOutput(out_port)]
```

```
            match_back = parser.OFPMatch(eth_type=ether.ETH_TYPE_IP,
                                         ip_proto=inet.IPPROTO_TCP,
                                         ipv4_src=ipv4_dst,
                                         ipv4_dst=self.nat_public_ip,
                                         tcp_src=tcp_dst,
                                         tcp_dst=nat_port)

            actions_back = [parser.OFPActionSetField(eth_dst=eth_src),
                            parser.OFPActionSetField(ipv4_dst=ipv4_src),
                            parser.OFPActionSetField(tcp_dst=tcp_src),
                            parser.OFPActionOutput(in_port)]
        elif pkt_udp:
            # print "@@@ Install UDP Flow Entry @@@"
            udp_src = pkt_udp.src_port
            udp_dst = pkt_udp.dst_port

            match = parser.OFPMatch(in_port=in_port,
                                    eth_type=ether.ETH_TYPE_IP,
                                    ip_proto=inet.IPPROTO_UDP,
                                    ipv4_src=ipv4_src,
                                    ipv4_dst=ipv4_dst,
                                    udp_src=udp_src,
                                    udp_dst=udp_dst)

            actions =
[parser.OFPActionSetField(eth_dst=IP_TO_MAC_TABLE[target_ip]),
parser.OFPActionSetField(ipv4_src=self.nat_public_ip),
                        parser.OFPActionSetField(udp_src=nat_port),
                        parser.OFPActionOutput(out_port)]

            match_back = parser.OFPMatch(eth_type=ether.ETH_TYPE_IP,
                                         ip_proto=inet.IPPROTO_UDP,
                                         ipv4_src=ipv4_dst,
                                         ipv4_dst=self.nat_public_ip,
                                         udp_src=udp_dst,
                                         udp_dst=nat_port)

            actions_back = [parser.OFPActionSetField(eth_dst=eth_src),
                            parser.OFPActionSetField(ipv4_dst=ipv4_src),
                            parser.OFPActionSetField(udp_dst=udp_src),
                            parser.OFPActionOutput(in_port)]
        else:
            pass
```

Par conséquent, lorsque l'instruction Flow Entry est exécutée, il n'y a pas de partie ICMP. Pour faire passer la commande Ping efficacement, Il faut donc activer la résolution de la partie ICMP.

Activation de la section ICMP

La première étape consiste donc à obtenir les informations d'en-tête ICMP en activant la prise en

charge de protocole ICMP Pkt_icmp = pkt.get_protocol (icmp.icmp) dans le code :

```
set_ev_cls(ofp_event.EventoFPPacketIn, MAIN_DISPATCHER)
ef        (self, ev),
  msg = ev.msg
  datapath = msg.datapath
  ofproto = datapath.ofproto
  parser = datapath.ofproto_parser
  in_port = msg.matchr['in_port']
  pkt = packet.Packet(msg.data)

  pkt_ethernet = pkt.get_protocol(ethernet.ethernet)
  pkt_arp = pkt.get_protoco/(arp.arp)
  Pkt_icmip= pkt.get_protocol(icmp.icmp)
  pkt_ip = pkt.get_protocol(ipv4.ipv4)
  pkt_tcp = pkt.get_protocol(tcp.tcp)
  pkt_udp = pkt.get_protocol(udp.udp)
```

Ensuite, lorsqu'on a déterminé qu'il s'agit d'un paquet ICMP, les instructions doivent être données au commutateur.

```
elif pkt_icmp:
            print "@@@ Install ICMP Flow Entry @@@"

            match = parser.OFPMatch(in_port=in_port,
                                    eth_type=ether.ETH_TYPE_IP,
                                    ip_proto=inet.IPPROTO_ICMP,
                                    ipv4_src=ipv4_src,
                                    ipv4_dst=ipv4_dst,
                                    )

            actions =
[parser.OFPActionSetField(eth_dst=IP_TO_MAC_TABLE[target_ip]),
parser.OFPActionSetField(ipv4_src=self.nat_public_ip),
                        parser.OFPActionOutput(out_port)]

            match_back = parser.OFPMatch(eth_type=ether.ETH_TYPE_IP,
                                         ip_proto=inet.IPPROTO_ICMP,
                                         ipv4_src=ipv4_dst,
                                         ipv4_dst=self.nat_public_ip,
                                         )

            actions_back = [parser.OFPActionSetField(eth_dst=eth_src),
                            parser.OFPActionSetField(ipv4_dst=ipv4_src),
                            parser.OFPActionOutput(in_port)]
 else:
            pass
```

Ensuite, il faut déterminer si l'adresse cible est dans ARP TABLE.

```
if pkt_tcp:
    if target_ip in IP_TO_MAC_TABLE:
        self._private_to-public(datapath=datapath,
                                buffer_id=msg.buffer_id,
                                data=msg.data,
                                in_port=in_Port,
                                out-port=self.wan_Port,
                                pkt_ethernet=pkt_ethernet,
                                pkt_ip=pkt_ip,
                                pkt_tcp=pkt_tcp)
    elif pkt_udp:
        if target_ip in IP_TO_MAC_TABLE:
            self._private_to_public(datapath=datapath,
                                buffer_id=msg.buffer_id,
                                data=msg.data,
                                in_port=in_Port,
                                out-port=self.wan_Port,
                                pkt_ethernet=pkt_ethernet,
                                pkt_ip=pkt_ip,
                                pkt_udp=pkt_udp)
    elif pkt_icmp:
        if target_ip in IP_TO_MAC_TABLE:
            self._private_to_public(datapath=datapath,
                                buffer_id=msg.buffer_id,
                                data=msg.data,
                                in_port=in_Port,
                                out-port=self.wan_Port,
                                pkt_ethernet=pkt_ethernet,
                                pkt_ip=pkt_ip,
                                pkt_udp=pkt_icmp)
```

Relancer Ryu NAT

```
$ ryu-manager base.py l2switch.py dhcp.py snat.py --verbose
```

Essayer à nouveau d'utiliser la commande Ping de H1 à H2.

```
H1:
$ ping 192.168.1.10
```

Peut être cette fois, cela passera