Laboratorium - wykorzystanie P4 do obsługi migrującej końcowki

Anna Gut, Tomasz Ukowski, Zuzanna Brzezińska

July 2023

1 Wprowadzenie teoretyczne

1.0.1 Migrująca końcówka

Z migrującą końcówką mamy do czynienia, kiedy urządzenie końcowe (takie jak komputer czy serwer) zmienia swoje położenie, zarówno fizycznie (np. przeniesienie maszyny wirtualnej między serwerami fizycznymi), jak i logicznie (zmiana adresu IP).

1.0.2 Migrująca końcówka w CISCO ACI

Cisco ACI to rozwiązanie będące siecią SDN (Software-Defined Networking - sieć oparta na koncepcji wydzielenia komponentów sterujących. Umożliwia to centralne konfigurowanie sieci i usług sieciowych.), wykorzystywane do budowy centrów przetwarzania danych i systemów chmurowych. Używa ono architektury leaf-spine (liść-grzbiet) - każdy przełącznik dostępu („liść") łączy się z każdym przełącznikiem agregacji („grzbiet"), co zapewnia rednundantość i wydajność. Do warstwy liści podłączane są urządzenia końcowe, takie jak: serwery, routery, firewalle, load-balancery.
W ACI dynamiczne reagowanie na migrujące końcówki odbywa się za pomocą kilku mechanizmów, w tym protokołu COOP (Council of Oracles Protocol) oraz "bounce entries".
COOP służy do przekazywania informacji o mapowaniu (lokalizacji i tożsamości - identity) do proxy grzbietu (spine). Leaf switch przekazuje informacje o adresie endpointu (urządzenia końcowego) do switcha rdzenia (spine switch) za pomocą Zero Message Queue (ZMQ). COOP działający na węzłach grzbietu będzie zapewniał, że wszystkie węzły grzbietu utrzymują spójną kopię informacji o adresach endpointów końcowych i lokalizacji.

Mechanizm ten działa w następujący sposób:

1. Liść (leaf switch) wykrywa nowy endpoint i aktualizuje bazę danych COOP o wpis o tym endpoincie
2. Jeśli w bazie jest już wpis o tym samym endpoincie, COOP rozpoznaje migrację końcówki (endpointu) i zgłasza to poprzedniemu liściowi (temu który początkowo zgłosił dołączenie endpointu)
3. Pierwotny liść tworzy *bounce entry* - wpis który wskazuje na liść do którego przemigrował endpoint. Dzięki temu, ruch kierowany do tego endpointu jest przekierowywany do jego nowej lokalizacji.
4. Pierwotny liść usuwa stary wpis dotyczący tej końcówki

W rezultacie, dzięki protokołowi COOP oraz mechanizmowi "bounce entries", ACI jest w stanie dynamicznie reagować na migracje endpointów, zapewniając nieprzerwane przekazywanie danych do i z tych endpointów.

1.0.3 P4

P4 (Programming Protocol-Independent Packet Processors) to język programowania zaprojektowany specjalnie do programowania urządzeń sieciowych, takich jak switche i routery. Umożliwia on developerom dostosowanie zachowania urządzeń poprzez zdefiniowanie sposobu przetwarzania przez nie pakietów. Pozwala to na stworzenie wydajnej i elastycznej sieci. Switche P4 to switche, które można programować za pomocą języka P4. Są one często wykorzystywane w omawianych wcześniej sieciach definiowanych programowo - SDNach.

1.1 Cel laboratorium

Celem tego laboratorium będzie zaimplementowanie rozwiązania wzorowanego na protokole COOP w ACI za pomocą switchy P4. Laboratorium będzie obejmowało stworzenie topologii w mininecie i uzupełnienie kodu kontrolerów zapewniających funkcjonalność obsługi migracji końcówki.

1.2 Plan laboratorium

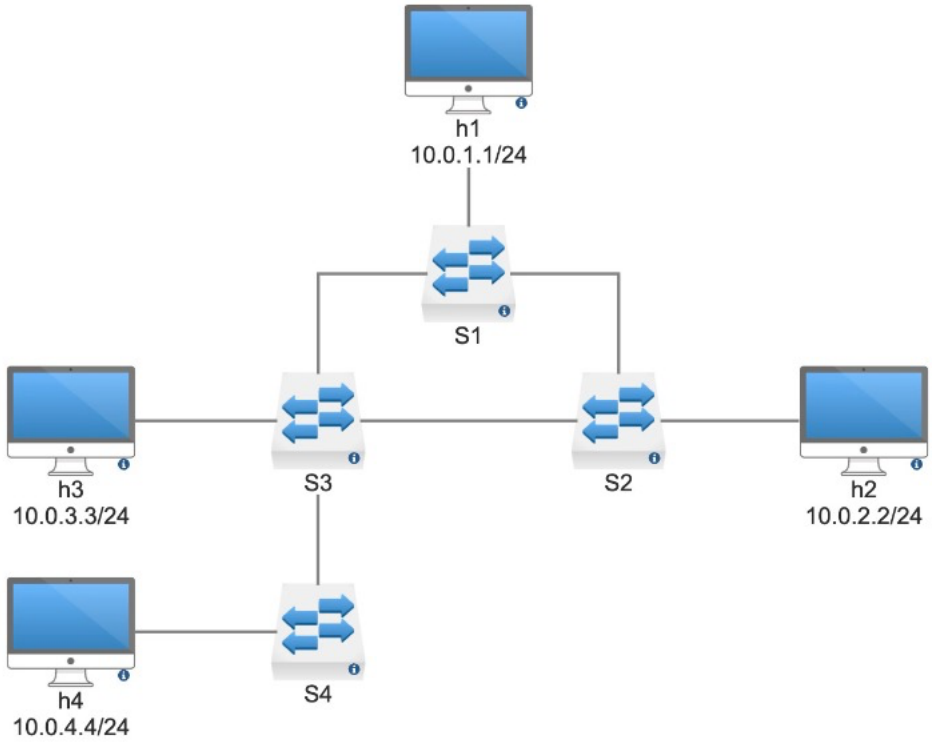
- Przygotowanie laboratorium
  - Stworzenie dokumentacji z wytłumaczeniem zagadnienia, omówieniem języka P4, topologii sieci i mechanizmów obsługujących migrujące końcówki
  - Przygotowanie kodu kontrolerów z lukami do uzupełnienia przez studentów.
  - Opracowanie instrukcji krok po kroku, jak wprowadzić zmiany w kodzie kontrolerów.
- Wprowadzenie teoretyczne:
  - Omówienie zagadnienia migrujących końcówek oraz języka P4.
  - Przedstawienie topologii sieci, która będzie używana w laboratorium
- Część praktyczna:
  - Studenci przeglądają dostarczony kod kontrolerów i zapoznają się z lukami do uzupełnienia.
  - Na podstawie instrukcji krok po kroku, studenci wprowadzają zmiany w kodzie kontrolerów, aby obsłużyć migrujące końcówki.
  - Studenci testują zmodyfikowany kod na przygotowanej topologii sieci, obserwując zachowanie sieci podczas migracji końcówek
- Podsumowanie i dyskusja:
  - Omówienie poprawnych rozwiązań i porównanie z wynikami uzyskanymi przez studentów.
  - Dyskusja na temat napotkanych problemów, wyzwań i możliwych ulepszeń.

.

1.3 Instrukcja

1.3.1 Stworzenie topologii w mininecie

1. Na samym początku będziemy potrzebować maszyny wirtualnej z Mininetem - można ją pobrać pod tym linkiem: http://mininet.org/download/.
2. Pobrany obraz importujemy do VirtualBoxa/VMware i uruchamiamy maszynę
3. Tworzymy plik *topology.json* definiujący topologię naszej sieci. Nasza sieć wygląda w następujący sposób:



Rysunek 1: Topologia sieci

Do pliku json kopiujemy poniższą konfigujację topologii:

```
{
    "hosts": {
        "h1": {"ip": "10.0.1.1/24", "mac": "08:00:00:00:01:11",
               "commands":["route add default gw 10.0.1.10 dev eth0",
                           "arp -i eth0 -s 10.0.1.10 08:00:00:00:01:00"]},
        "h2": {"ip": "10.0.2.2/24", "mac": "08:00:00:00:02:22",
               "commands":["route add default gw 10.0.2.20 dev eth0",
                           "arp -i eth0 -s 10.0.2.20 008:00:00:00:02:00"]},
        "h3": {"ip": "10.0.3.3/24", "mac": "08:00:00:00:03:33",
               "commands":["route add default gw 10.0.3.30 dev eth0",
                           "arp -i eth0 -s 10.0.3.30 08:00:00:00:03:00"]},
        "h4": {"ip": "10.0.4.4/24", "mac": "08:00:00:00:04:44",
               "commands":["route add default gw 10.0.4.40 dev eth0",
                           "arp -i eth0 -s 10.0.4.40 08:00:00:00:04:00"]}
    },
    "switches": {
        "s1": {"runtime_json": "s1-runtime.json"},
        "s2": {"runtime_json": "s2-runtime.json"},
        "s3": {"runtime_json": "s3-runtime.json"},
        "s4": {"runtime_json": "s4-runtime.json"}
    },
    "links": [
        ["h1", "s1-p1"], ["s1-p2", "s2-p2"], ["s1-p3", "s3-p2"],
        ["s3-p3", "s2-p3"], ["h2", "s2-p1"], ["h3", "s3-p1"],
        ["s3-p4", "s4-p2"], ["h4", "s4-p1"]
    ]
}
```

1.3.2 Tworzenie mechanizmów obsługi migrującej końcówki

Na tym etapie będziemy wypełniać kod kontrolera w celu zaobserwowania, w jako sposób jego mechanizm radzi sobie z migrowaniem endpointu.
Scenariusz migracji jest następujący:

1. Host h2 ma adres IP 10.0.2.2 i jest pierwotnie podłączony do przełącznika S2.
2. Podczas migracji fizycznej, host h2 zostaje odłączony od przełącznika s2 i przenoszony do innego miejsca w sieci.
3. Po przeniesieniu hosta h2, zostaje on podłączony do przełącznika S1
4. Adres IP hosta h2 (10.0.2.2) pozostaje niezmieniony, ale teraz host h2 znajduje się w podsieci obsługiwanej przez przełącznik S1.

1. Otwieramy plik controller.py, zawierający kod kontrolera, za pomocą którego odbywać się będzie wykrywanie migracji i tworzenie bounce entry

2. W funkcji monitorHostLocation, odpowiadającej za ciągłe odczytywanie tablic z wpisami o endpointach i porównywanie ich z aktualnym stanem bazy, musimy uzupełnić brakujący kod, oznaczony komentarzami todo:

```
def monitorHostLocation(p4info_helper, switch, added_entries_shared, removed_entries_shared):
"""
Monitors the host location by continuously reading the table entries and
comparing them with the expected rules.

:param p4info_helper: the P4Info helper
:param switch: the switch connection
"""

it = 0
expected_table_entries = []
while True:
    current_table_entries = []
    for response in switch.ReadTableEntries():
        for entity in response.entities:
            entry = entity.table_entry
            current_table_entries.append(entry)

    if current_table_entries != expected_table_entries and it != 0:
        # Distinguish whether a new host was added or an old one was removed

        added_entries = [entry for entry in current_table_entries if not entry in expected_table_entries]
        removed_entries = [entry for entry in expected_table_entries if not entry in current_table_entries]
        for entry in added_entries:
            ip_in_bytes = get_ipv4_dst_address(entry)
            # TODO - convert ip_in_bytes to dotted-decimal notation
            # TODO - put new switch name to added_entries_shared under proper index
            print('A new entry has been added. IP: {}'.format(ip_in_dotted_decimal))

        for entry in removed_entries:
            ip_in_bytes = get_ipv4_dst_address(entry)
            # TODO - convert ip_in_bytes to dotted-decimal notation
            # TODO - put new switch name to removed_entries_shared under proper index

            print('An entry has been removed. IP: {}'.format(ip_in_dotted_decimal))
        expected_table_entries = current_table_entries

    if it == 0:
        expected_table_entries = current_table_entries
```

```
sleep(2)
it += 1
```

3. Następnie przechodzimy do funkcji checkForMigration i uzupełniamy jej kod

```
 def checkForMigration(added_entries_shared, removed_entries_shared):
     while True:
         for host in list(added_entries_shared.keys()):
             if host in list(removed_entries_shared.keys()):
                 print(f"Migrating endpoint, host:{host} from switch
                 {removed_entries_shared[host]} to switch {added_entries_shared[host]}")

                 # Remove the entries from the shared dicts to avoid repeat notifications
                 new_sw_name = added_entries_shared[host]
                 old_sw_name = removed_entries_shared[host]
                 added_entries_shared.pop(host)
                 removed_entries_shared.pop(host)

                 # TODO - Start a new thread to handle bounce entry creation and removal

         time.sleep(1)
```

4. Następnie przechodzimy do funkcji handleBounceEntry

```
    def handleBounceEntry(p4info_helper, host, new_switch_name, old_switch_name, switches):
    """
    Function to handle bounce entry creation and removal
    """
    print(f"Creating bounce entry for migrating host: {host}")

    for switch in switches:
        if switch.name == old_switch_name:
            table_entry = p4info_helper.buildTableEntry(
                table_name="MyIngress.ipv4_lpm",
                match_fields={
                    "hdr.ipv4.dstAddr": (host, 32)
                },
                action_name="MyIngress.ipv4_forward",
                action_params = {
                # TODO - to fields "dstAddr" and "port" assign proper
                # elements form switch_mac_table and switch_ports"
                })
            switch.WriteTableEntry(table_entry)
            print("Installed bounce entry for host {} on switch {}".format(host, old_switch_name))
```

```
break
```

Host IP	MAC Address	Switch Port
10.0.2.2	08:00:00:00:02:22	p1
10.0.1.1	08:00:00:00:01:11	p2
10.0.3.3	08:00:00:00:03:33	p3

Tabela 1: Switching Table S2 przed instalacją Bounce Entry

Host IP	MAC Address	Switch Port
10.0.2.2	08:00:00:00:02:22	p2
10.0.1.1	08:00:00:00:01:11	p2
10.0.3.3	08:00:00:00:03:33	p3

Tabela 2: Switching Table S2 po instalacji Bounce Entry

Host IP	MAC Address	Switch Port
10.0.1.1	08:00:00:00:01:11	p2
10.0.3.3	08:00:00:00:03:33	p3

Tabela 3: Switching Table S2 po wygaśnięciu Bounce Entry

5. Uruchamiamy kod za pomocą komendy make
6. Sprawdzamy komunikację między hostami (*h1 ping h2*)
7. W drugim oknie w terminalu uruchamiamy plik kontrolera (*python my-controller.py*) i obserwujemy w logach programu, w jaki sposób kontroler radzi sobie z migracją endpointu
8. Sprawdzamy komunikację między hostami po przeprowadzeniu migracji hosta (*h1 ping h2*)

1.4 Bibliografia

- https://www.cisco.com/c/en/us/td/docs/switches/datacenter/aci/apic/sw/5-x/security/cisco-apic-security-configuration-guide-50x/m_coop_authentication.pdf
- https://www.cisco.com/c/dam/global/pl_pl/solutions/data-center-virtualization/pdfs/InfographicDCHeroSDN_PL_ebkdc009383.pdf
- https://www.thenetworkdna.com/2023/02/cisco-aci-control-plane-components.html