Workshop OpenFlow

2. Routing and Monitoring

Akbari Indra Basuki
Pusat Penelitian Informatika, LIPI

Daftar Materi

Topik	Keterangan
Basic forwarding	Dasar-dasar OpenFlow
Routing & Monitoring	Program Controller: Shortest-path routing Monitor node and link status NetworkX integration
Packet Filtering (Firewall + Web Interface)	Program Controller: Bloom Filter, Flask integration
Load balancing	Group bucket and group tables Round robin load balancing Main-backup path protection
Rate limiting	Meter tables
Stateless vs Stateful data plane Stateful data plane	Jenis data plane dalam memproses paket OpenState SDN Arp handling Port Knocking

Routing

Basic forwarding menggunakan L2 switch terbatas pada satu jaringan

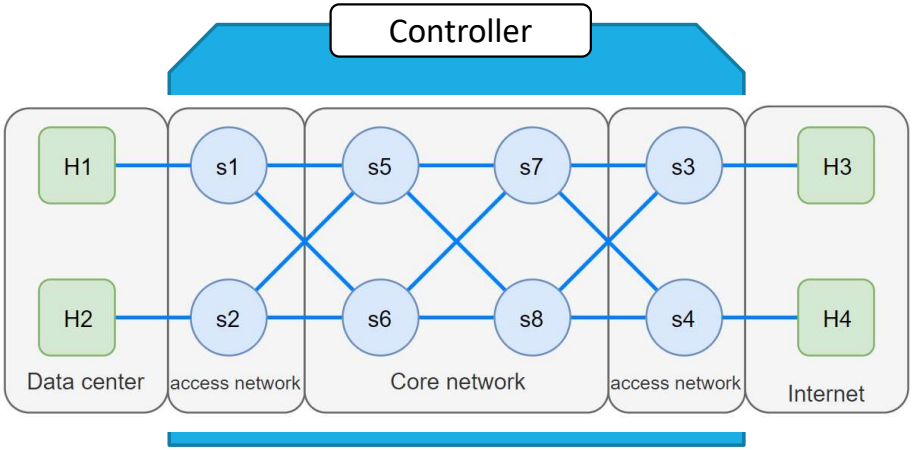
Untuk menghubungkan beberapa jaringan sehingga dapat berkomunikasi antara satu sama lain, perlu diterapkan aturan routing

Hal yang perlu disiapkan:

- Monitoring topologi jaringan
- Komputasi jalur routing dari antar switch
- Install flow rule untuk jalur routing tersebut

Studi kasus:

- Topologi data center dengan skema leaf and spine
- Terdiri dari access switch dan core switch
- Studi kasus 1: Pengiriman via Controller
- Studi kasus 2: Pengiriman via shortest path routing



Simplified spine and leaf topology

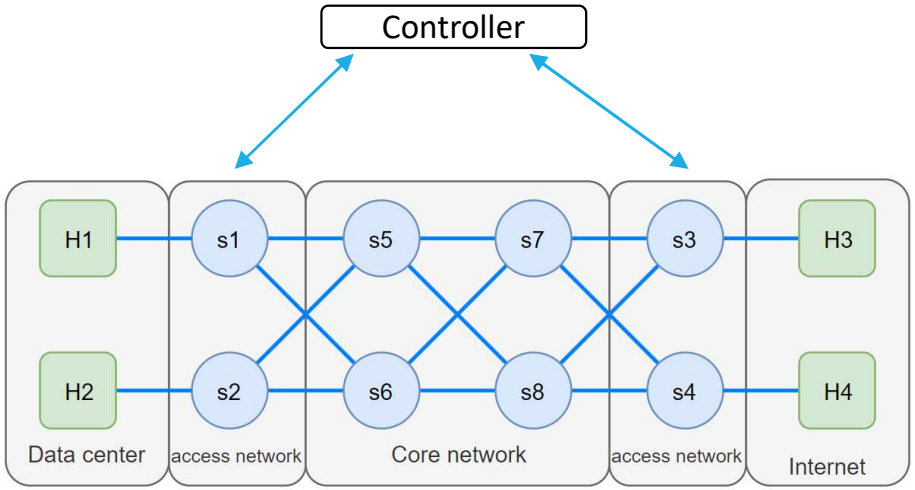
Studi kasus 1 - Routing via controller

Skenario:

- ARP paket dari host dikirim ke controller oleh access switch
- Controller akan mengirim paket ke semua access switch yang lain

Implementasi:

- Controller memonitor topologi jaringan
- Controller mengisntall flow rule static untuk mengirim paket ke Controller di setiap access switch
- Controller akan meneruskan paket tersebut ke access switch selain access switch pengirim



Source code

Monitor topo dan konversi ke format graph

```
@set_ev_cls(event.EventSwitchEnter)
def get_topology_data(self, ev):
        def drawGraph():
                pos = nx.spring_layout(self.G)
                nx.draw_networkx_nodes(self.G, pos, node_size = 500)
                nx.draw_networkx_labels(self.G, pos)
                black_edges = [edge for edge in self.G.edges()]
                nx.draw_networkx_edges(self.G, pos, edgelist=black_edges, arrows=False)
                #plt.show() #it will halt the program, close the window to continue
                plt.savefig('topology.png')
                plt.clf()

        def convertToGraph():
                #self.G = nx.Graph()
                self.G.clear()
                for lk in self.lkListInfo:
                        self.G.add_edge(str(lk[0]), str(lk[1]))
                #drawGraph()

        #  ryu-manager <path/this_file> --observe-links
        self.swList = get_switch(self.topology_api_app, None)
        self.swListID =[switch.dp.id for switch in self.swList]
        self.lkList = get_link(self.topology_api_app, None)
        self.lkListInfo =[(link.src.dpid,link.dst.dpid,{'port':link.src.port_no}) for link in

        convertToGraph()
```

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
        msg = ev.msg
        in_port = msg.match['in_port']
        dp = msg.datapath
        ofproto = dp.ofproto
        dpid = dp.id
        pkt = packet.Packet(msg.data)

        print pkt
        pkt_ethernet = pkt.get_protocols(ethernet.ethernet)[0]
        #print pkt_ethernet.ethertype
        pkt_arp = pkt.get_protocols(arp.arp)

        if pkt_arp:
                pkt_arp = pkt.get_protocols(arp.arp)[0]
                print ("receiving arp packet")
                #ambil IP pengirim dan asosiasikan dengan switch dan input port.
                pkt_arp = pkt.get_protocol(arp.arp)
                sipv4 =  pkt_arp.src_ip
                self.hostDB[sipv4] = (dp.id, in_port)

                #kirim ke semua access switch yg lain
                self.sendPacketArp(dp, msg, pkt)

        pkt_ipv4 = pkt.get_protocols(ipv4.ipv4)
        if pkt_ipv4:
                pkt_ipv4 = pkt.get_protocols(ipv4.ipv4)[0]
        pkt_icmp = pkt.get_protocols(icmp.icmp)
        if pkt_icmp:
                pkt_icmp = pkt.get_protocols(icmp.icmp)[0]
                #kirim ke access switch yg sesuai
                if pkt_icmp.type == 0:#reply
                        dst_ip = pkt_ipv4.dst
                        print "DB:",self.hostDB[dst_ip]
                        self.sendPacketICMP(dp, msg, pkt, dst_ip)
                if pkt_icmp.type == 8:#request
                        dst_ip = pkt_ipv4.dst
                        print "DB:",self.hostDB[dst_ip]
                        self.sendPacketICMP(dp, msg, pkt, dst_ip)
```

Jika terima paket ARP panggil fungsi kirim ARP

Jika terima paket ICMP panggil fungsi kirim ICMP

```
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
        msg = ev.msg
        dp = msg.datapath
        ofproto = dp.ofproto
        print "config sw:",dp.id
        self.delflowintable(dp)
        self.swList.append(dp)  #daftar switch
        for sw in self.swList:
                self.swListDB[sw.id] = sw
                if sw.id < 5 and not(sw in self.swAcsList):
                        self.swAcsList.append(sw)
        print "Access sw:",len(self.swAcsList), self.swAcsList
        if dp.id < 5:
                match = dp.ofproto_parser.OFPMatch(eth_type=0x806)  #0x806 -> ARP ethertype
                actions = [dp.ofproto_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]
                self.add_flow(dp, match, actions)

                match = dp.ofproto_parser.OFPMatch(eth_type=0x800,ip_proto=0x01)  #0x800, proto 0x01 -> ICMP
                actions = [dp.ofproto_parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,ofproto.OFPCML_NO_BUFFER)]
                self.add_flow(dp, match, actions)
```

Paket ARP dan ICMP dari host kirim ke Controller

```
def sendPacketArp(self, datapath, msg, pkt):
        for dp in self.swAcsList:
                ofproto = dp.ofproto
                if dp.id != datapath.id:#bukan switch penerima arp dari host
                        actions = [dp.ofproto_parser.OFPActionOutput(1 , 0)]
                        data =msg.data
                        out = dp.ofproto_parser.OFPPacketOut(datapath=dp, buffer_id=msg.buffer_id,
                                                             in_port=ofproto.OFPP_CONTROLLER, actions=actions, data=data)
                        dp.send_msg(out)
                        print "sent to: S-", dp.id, " port:1"
```

Fungsi untuk mengirim ARP ke access switch

```
def sendPacketICMP(self, datapath, msg, pkt, dstIP):
        dstTpl = self.hostDB[dstIP]
        val  = int(dstTpl[0])
        print val
        dp = self.swListDB[val]
        portdst = 1#int(dstTpl[1])
        ofproto = dp.ofproto
        actions = [dp.ofproto_parser.OFPActionOutput(portdst , 0)]
        data =msg.data
        out = dp.ofproto_parser.OFPPacketOut(datapath=dp, buffer_id=msg.buffer_id,
                                             in_port=ofproto.OFPP_CONTROLLER, actions=actions, data=data)
        dp.send_msg(out)
        print "sent to: S-", dp.id, " port:", portdst
```

Fungsi untuk mengirim ICMP ke access switch

Langkah Pengujian

Pengujian:

- Jalankan aplikasi controller simpleWebFilter.py

  Ryu-manager simpleswitchL3.py
- Jalankan mininet, topologi: simpleTopo.py

  Sudo python core.py
- H4 menjalankan program sniffer (tcpdump)

  Tcpdump –ni h4-eth0
- Ping H4 dari H1

  Ping 10.0.0.4
- Untuk membuktikan bahwa paket dikirim via controller bukan core switch, mulai ulang percobaan dengan memutus semua link di core network

  Link s5 s7 down, link s5 s8 down, link s6 s7 down, link s6 s8 down
- Ping H4 dari H1 dan amati hasilnya.

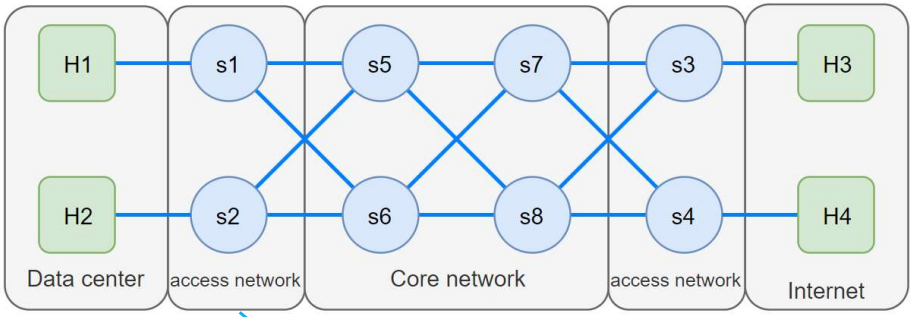
Shortest-path routing – Pendahuluan

Pendahuluan:

- Jaringan sesungguhnya → routing berbasis shortest path
- Controller mengkomputasi rute shortest path dari setiap ingress/access switch ke ingress switch yang lain
- Pengiriman paket ke IP tujuan di asosiasikan dengan ingress switch dari host pemilik IP tersebut. Misal h1 = ingress switch s1
- Shortest path di update setiap ada link down → controller menerima laporan dari switch

Implementasi:

- Controller memonitor topologi jaringan dan mencari shortest path dari setia ingress switch
- ARP reply valid → controller menginstall flow rule untuk IP tersebut berdasarkan switch ingress dan nomor port yang menerima IP tersebut.



End node shortest path routing

Shortest-path routing – Percobaan

Kisi-kisi:

- Untuk mendapatkan informasi topologi, pakai library *ryu.topology*

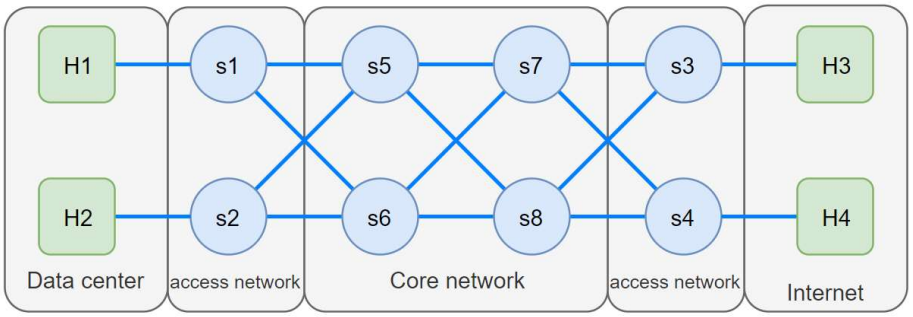
```
from ryu.topology import event, switches                    # library bawaan ryu untuk event dan switch
from ryu.topology.api import get_switch, get_link           # library bawaan ryu untuk informasi link dan switch
import networkx as nx                                       # library networkx untuk operasi graph

@set_ev_cls(event.EventSwitchEnter)                         # Event ketika switch terkoneksi ke Controller Ryu
def get_topology_data(self, ev):
    switch_list = get_switch(self.topology_api_app, None)   # informasi switch
    switches = [switch.dp.id for switch in switch_list]     # array berisi switch ID
    links_list = get_link(self.topology_api_app, None)      # informasi link, format tuple (S1, S2)
    # selanjutnya konversi topologi menjadi informasi graph untuk libray networkx
    # contoh; G.add_nodes_from([2, 3])  --> menambahkan switch ID 2 dan 3 beserta link 2-3 kedalam graph
```

- Untuk komputasi shortest path, pakai library python network: pip install networkx
- Letakkan kode untuk autentifikasi ARP reply di fungsi: def _packet_in_handler(self, ev)
- Jika terautentifikasi, hitung shortest path dari setiap ingress switch ke switch tujuan (switch pengirim ev)
- Ulangi untuk setiap host baru yang ter-autentifikasi dan ketika ada port yang down atau link terputus
- Untuk mendengarkan informasi link down:

  https://ryu.readthedocs.io/en/latest/ofproto_v1_3_ref.html#port-status-message

Langkah pengujian



1. Jalankan emulator jaringan: Sudo python core.py
2. Jalankan controller: ryu-manager shortestPath.py
3. Buka jendela host: Xterm h1 h4
4. Cek koneksi dari host 1 ke host 4 (jendela h1) ping 10.0.0.4
5. Cek shortest path terpilih dengan menjalankan program sniffer di S1 dan S4, port 1

   tcpdump –n –i S1-eth1 dan tcpdump –n –i S4-eth1
6. Putus koneksi dari host 1 ke switch terpilih (misal: link S1-S5 down) dan amati berapa lama controller memperbaharui koneksi.

Latihan Akhir

Pengembangan lanjutan:

1. Routing dengan skema protection
   - Untuk menjamin realiabilitas routing → jalur routing ganda (disjoint path algorithm)
   - Gunakan fast rerouting group table untuk memilih jalur mana yang aktif (Bab 4)

2. Routing dengan skema stateful (bab 6)
   - Tidak perlu menghubungi controller untuk mengkomputasi ulang shorthest path

Bandingkan waktu pemulihan jaringan antara skema shorteat path biasa dengan skema kedua skema diatas

Daftar pustaka

- RyuBook. https://osrg.github.io/ryu-book/en/Ryubook.pdf
- Mininet. http://mininet.org/walkthrough/
- Networkx. https://networkx.github.io/documentation/stable/tutorial.html