

A decorative graphic on the left side of the slide, consisting of several thin, curved lines in shades of blue and grey, and a solid black arrow pointing to the right.

Separação de fluxos TCP e UDP

Utilizando controlador POX e OpenFlow

Francisco Carlos Baddini

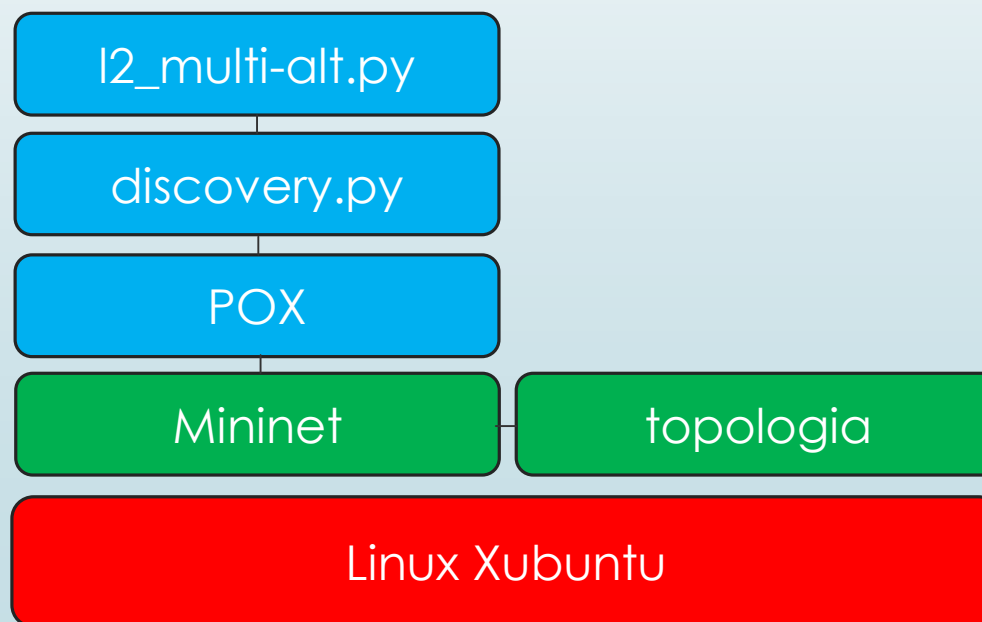
Reinaldo do Valle Júnior

Descrição do problema

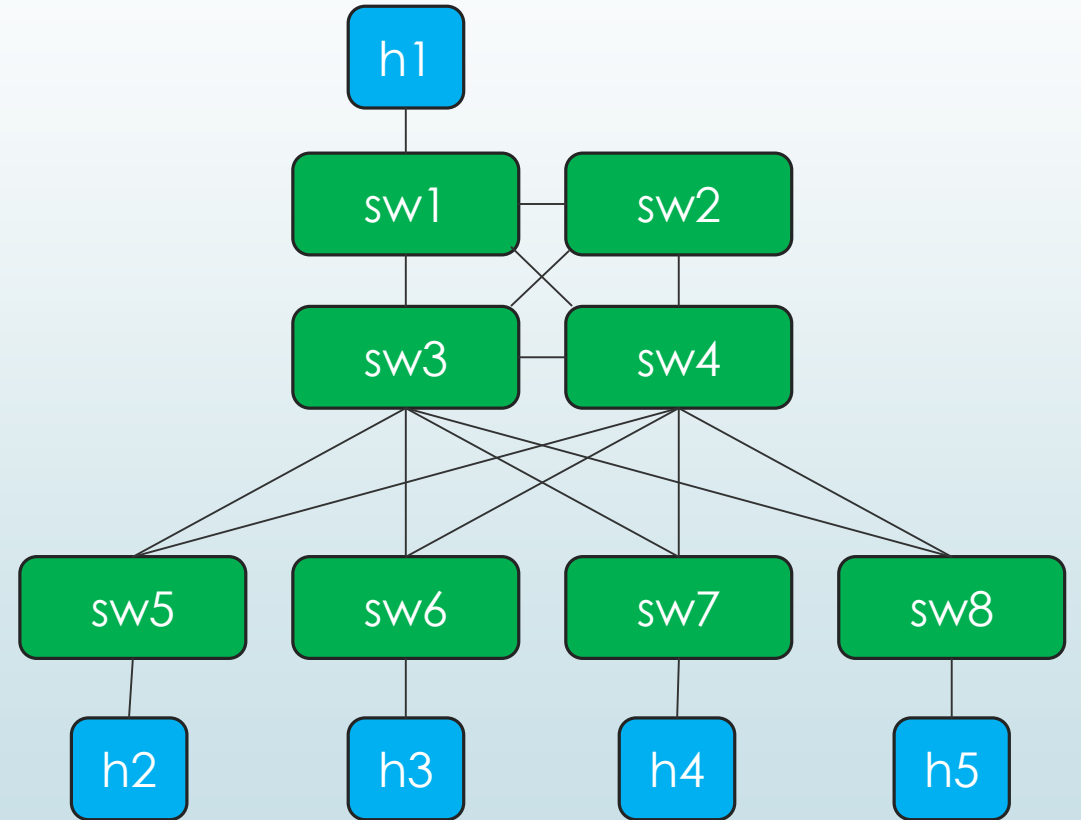
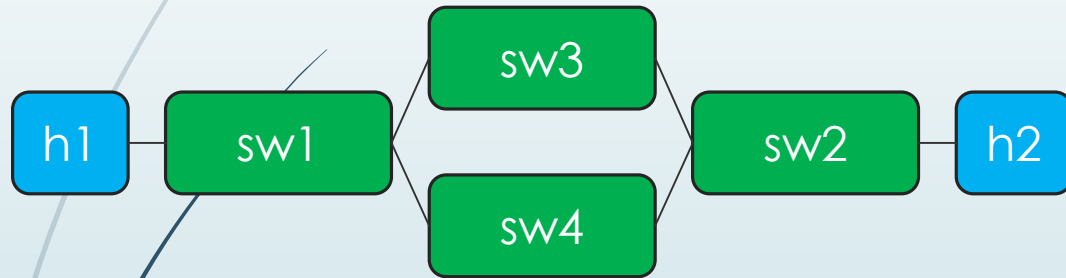
- ▶ Separar o tráfego de dois protocolos diferentes
 - ▶ Os fluxos desses protocolos não devem utilizar os mesmos caminhos
 - ▶ Nesse estudo os protocolos são TCP e UDP
- ▶ Porque separar o tráfego
 - ▶ A separação de tráfego em grandes redes é uma necessidade
 - ▶ Aumento da eficácia
 - ▶ Racionalização de recursos da infraestrutura
 - ▶ Controle do tráfego
 - ▶ Redução de custos
- ▶ O que se espera
 - ▶ A solução deve atender a topologias genéricas (lineares, árvore e híbridas)

Solução

- Arquitetura e Tecnologias Utilizadas
 - SO Linux Xubuntu 14.04 LTS
 - Mininet 2.1.0 – virtualização de rede
 - Controlador POX – linguagem Python 2.7.5
 - Uso dos módulos Discovery e l2_multi do controlador



2 topologias adotadas nos testes



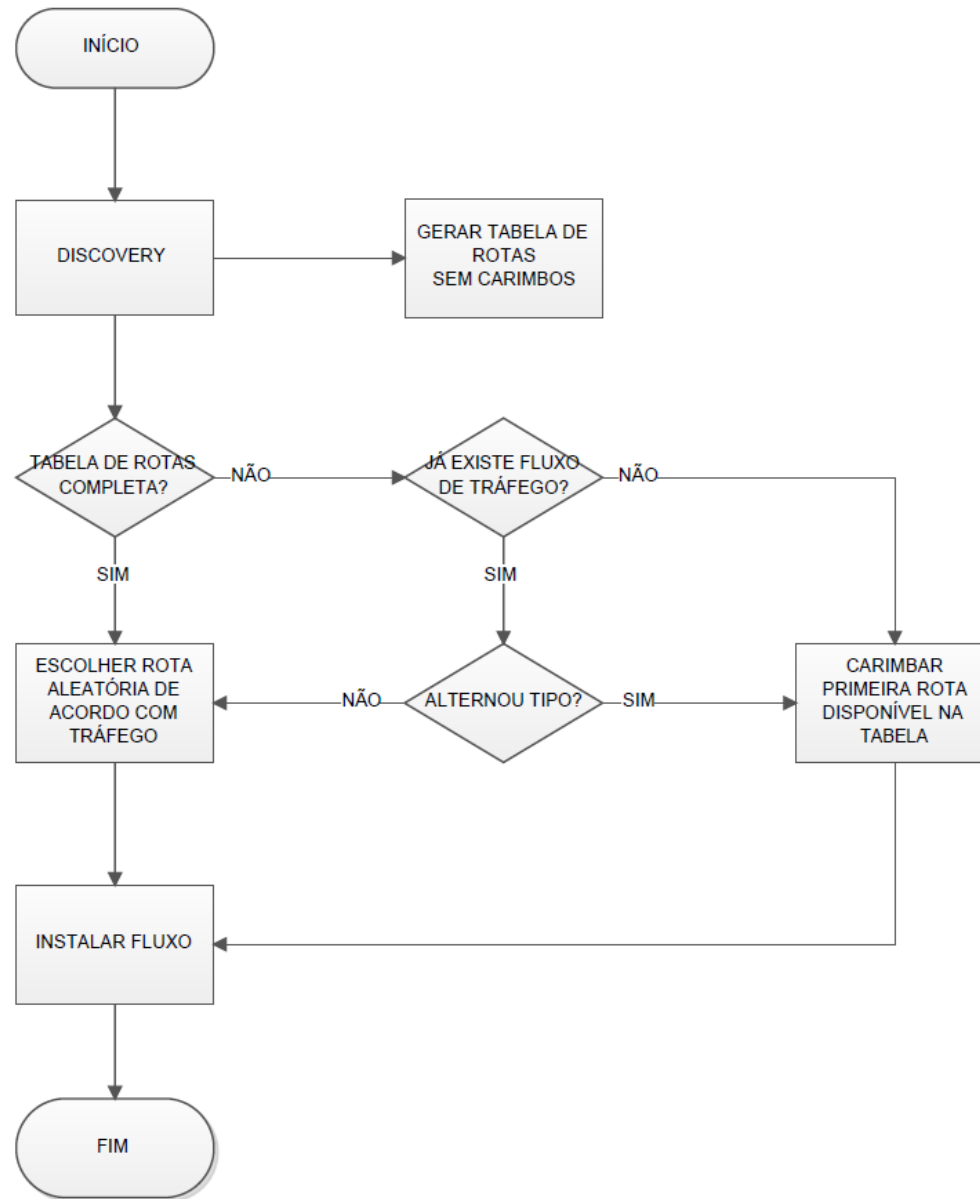
Solução (primeira abordagem: direcionamento por portas)

- Escolha em função do protocolo
 - Dependendo do protocolo o tráfego seria direcionado para uma porta
 - Solução apenas para o primeiro switch
 - Não considera endereços de origem e destino

Solução (base inicial)

- ▶ DISCOVERY e L2_MULTI: FUNCIONAMENTO
 - ▶ Usamos o módulo Discovery do POX associado à execução do L2_multi.py
 - ▶ Lista de switches levantados pelo discovery
 - ▶ L2_multi aplica algoritmo de *shortest path Floyd-Warshall*
 - ▶ Obtém caminho completo (switches e portas usadas) a partir de host de origem e destino
 - ▶ Instala fluxo

Solução (segunda abordagem: rotas)



Solução (segunda abordagem: rotas)

The screenshot displays the MySQL Workbench interface. The left sidebar shows the 'SCHEMAS' tree with 'projeto' expanded to show 'Tables', 'Views', 'Stored Procedures', and 'Functions'. The 'Views' folder is selected. The main window shows a query editor with the following SQL statement:

```
1 • SELECT * FROM discovery
```

The 'Result Set Filter' is empty, and the 'Export' button is visible. The results pane shows a table with the following data:

#	IDTOPOLOGIA	LINK
121	2014061009311402417909111102	00-00-00-00-00-03.1 -> 00-00-00-00-00-02.5
122	2014061009311402417909111102	00-00-00-00-00-05.1 -> 00-00-00-00-00-02.1
123	2014061009311402417909111102	00-00-00-00-00-06.1 -> 00-00-00-00-00-02.4
124	2014061009311402417909111102	00-00-00-00-00-04.1 -> 00-00-00-00-00-02.2
125	2014061009311402417909111102	00-00-00-00-00-02.5 -> 00-00-00-00-00-03.1
126	2014061009311402417909111102	00-00-00-00-00-01.4 -> 00-00-00-00-00-03.2
127	2014061009311402417909111102	00-00-00-00-00-05.2 -> 00-00-00-00-00-01.5
128	2014061009311402417909111102	00-00-00-00-00-02.2 -> 00-00-00-00-00-04.1
129	2014061009311402417909111102	00-00-00-00-00-06.2 -> 00-00-00-00-00-01.1
130	2014061009311402417909111102	00-00-00-00-00-01.2 -> 00-00-00-00-00-04.2
131	2014061009311402417909111102	00-00-00-00-00-03.2 -> 00-00-00-00-00-01.4
132	2014061009311402417909111102	00-00-00-00-00-02.4 -> 00-00-00-00-00-06.1
133	2014061009311402417909111102	00-00-00-00-00-04.2 -> 00-00-00-00-00-01.2
134	2014061009311402417909111102	00-00-00-00-00-01.1 -> 00-00-00-00-00-06.2

The 'Action Output' pane at the bottom shows the following log entries:

Time	Action	Message	Duration / Fetch
09:30:35	SELECT * FROM discovery LIMIT 0, 1000	102 row(s) returned	0.000 sec / 0.000 sec
09:32:05	SELECT * FROM discovery LIMIT 0, 1000	134 row(s) returned	0.000 sec / 0.000 sec

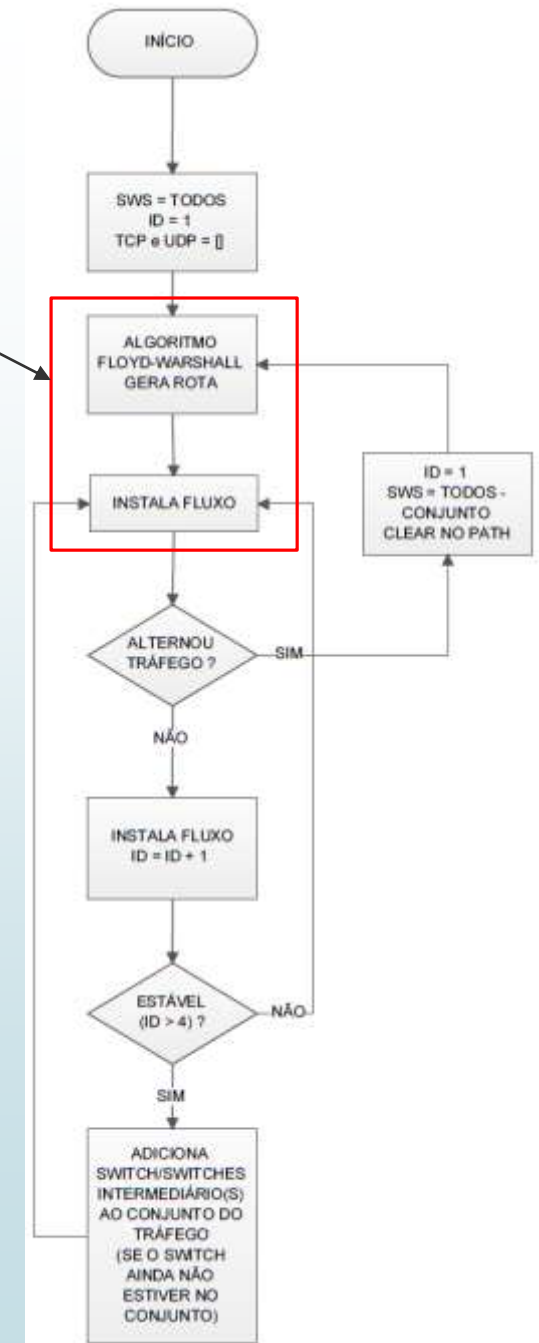
The status bar at the bottom indicates 'Query Completed'.

Solução (terceira abordagem: switches intermediários)

► Implementação

- O fluxograma ao lado indica o algoritmo desenvolvido para escolha de rotas em redes OpenFlow com separação de tráfego TCP e UDP

L2_multi
"puro"



Implementação

- ▶ No *script* `I2_multi.py` **acrescentamos:**
 - ▶ Identificação de fluxos TCP e UDP por meio da *flag* no campo *Ethernet Type* do pacote OpenFlow
 - ▶ Variáveis globais para armazenar caminhos, fluxos, switches e estados
 - ▶ Rotinas para tratar os tráfegos
 - ▶ Rotinas para imprimir os resultados em arquivo texto

Implementação

- Implementamos variáveis globais necessárias ao controle dos fluxos
 - global abc (ID do fluxo)
 - global trafftcp (flag de tráfego TCP)
 - global trafudp (flag de tráfego UDP)
 - global swstcp (lista de switches do tráfego TCP)
 - global swsudp (lista de switches do tráfego UDP)
 - global anterior (flag usado para indicar alteração de tráfego)
 - global protocolo (indicador do protocolo em andamento)
 - global alterou (flag que indica alteração de tráfego)

Implementação

- ▶ Rotina de identificação de tráfego
 - ▶ Usado payload e campo de tipo de tráfego

```
if packet.type == ethernet.IP_TYPE:
```

```
    ip_packet = packet.payload
```

```
    if ip_packet.protocol == 6:
```

```
        protocolo = "TCP"
```

```
        traftcp = 1
```

```
        trafudp = 0
```

```
    else:
```

```
        if ip_packet.protocol == 17:
```

```
            protocolo = "UDP"
```

```
            trafudp = 1
```

```
            traftcp = 0
```

```
else:
```

```
    protocolo = "Outros"
```

```
    abc = 1
```

```
    trafudp = 0
```

```
    traftcp = 0
```

Implementação

- Rotina de identificação de tráfego alterado e alteração da lista de switches (SWS)
 - inclusão de "alterou == 1", para forçar execução do algoritmo de *short path* quando o tráfego se alterna

```

if anterior == 2:
    print "INICIOU O TRAFEGO!!!"
    print "Switch intermediario = ", str(intermediate)
else:
    # a partir do segundo pacote
    if anterior != traftcp:
        print "ALTEROU O TRAFEGO!!!"
        alterou = 1
        abc = 1
        #para trafego TCP remove de sws
        a lista de switches UDP
        if traftcp ==1 and trafudp ==0:
            sws = switches.values()

```

```

for switchremover in swsudp:
    if switchremover in sws:
        sws.remove(switchremover)
#para trafego UDP remove de sws a lista de switches TCP
if trafudp ==1 and traftcp ==0:
    sws = switches.values()
    for switchremover in swstcp:
        if switchremover in sws:
            sws.remove(switchremover)
    #no final da um clear no path_map para forçar
    path_map.clear()
else:
    print "Switch intermediario = ", str(intermediate)
anterior = traftcp

```

Implementação

Rotina de inclusão dos switches TCP ou UDP nas listas

```
if abc > 4:
```

```
    #print "ESTAVEL APOS 4 FLUXOS INSTALADOS - CARIMBANDO  
    SWITCHES INTERMEDIARIOS!"
```

```
    #trafego TCP, appenda na lista os switches intermediarios  
    (intermediate) obtidos da rota via algoritmo, UDP faz o mesmo
```

```
    #depois comando set para eliminar elementos repetidos da  
    lista
```

```
    if traftcp == 1:
```

```
        if len(swstcp) == 0:
```

```
            swstcp.append(intermediate)
```

```
            print "ADICIONANDO PRIMEIRO SWITCH AO CONJUNTO TCP"
```

```
        else:
```

```
            swstcp.append(intermediate)
```

```
            #adiciona ao conjunto novo switch se for novo
```

```
            swstcp = list(set(swstcp))
```

```
    if trafudp == 1 and traftcp == 0:
```

```
if len(swsudp) == 0:
```

```
    swsudp.append(intermediate)
```

```
    print "ADICIONANDO PRIMEIRO
```

```
    SWITCH AO CONJUNTO UDP"
```

```
    else:
```

```
        swsudp.append(intermediate)
```

```
        #adiciona ao conjunto novo switch
```

```
se for novo
```

```
        swsudp = list(set(swsudp))
```

Implementação

► Instalação dos fluxos

```

log.debug("Installing path for %s -> %s %04x (%i hops)",
        match.dl_src, match.dl_dst, match.dl_type, len(p))

#aqui so tem prints da instalacao do fluxo e do conteudo das
listas TCP e UDP

print " Instalando fluxo de " + str(match.dl_src) + " para " +
str(match.dl_dst) + " -> " + str(match.dl_type) + " (" + str(len(p)) + "
hops" + ")"

print "Switches TCP:", str(swstcp)
print "Switches UDP:", str(swsudp)

```

Testes e resultados obtidos

- As implementações descritas permitiram:
 - Separar o tráfego TCP e UDP em dois caminhos diferentes
 - Caso a topologia mude (eventos de links), o algoritmo escolherá um novo caminho
- Testes:
 - Funcionamento estável e dentro do esperado nas duas topologias adotadas como referência (linear e fat tree)

Próximos passos

- A solução proposta alcançou os objetivos!
 - Separa tráfegos TCP e UDP
 - Operacional para topologias genéricas
- Apresenta algumas limitações
 - Não opera em mais de dois caminhos se a topologia for estática
- Soluções futuras
 - Encontrar solução para as limitação citada acima
 - Separar tráfego para controle de banda em função dos protocolos
 - Separar tráfego para aplicações em função dos protocolos que utilizam

Referências

- Referências
- [1] Site da Open Networking Foundation. Disponível em <https://www.opennetworking.org>. Último acesso em 15 de julho de 2014.
- [2] Mininet walkthrough. Disponível em <http://mininet.org/walkthrough/#part-1-everyday-mininet-usage>. Último acesso em 15 de julho de 2014.
- [3] OpenFlow Tutorial. Disponível em http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial#Install_Required_Software. Último acesso em 15 de julho de 2014.
- [4] Wiki de POX. Disponível em <https://openflow.stanford.edu/display/ONL/POX+Wiki>. Último acesso em 15 de julho de 2014.
- [5] Tutorial e VM de POX. Disponível em <http://sdnhub.org/tutorials/pox>. Último acesso em 15 de julho de 2014.
- [6] Comunidade SDN Central. Disponível em <http://www.sdncentral.com>. Último acesso em 15 de julho de 2014.

A solid black arrow pointing to the right, positioned above the word "Perguntas".

Perguntas

