

# 第四次实验报告

## 1. EC 数量

根据文件的提示，可以看到在处理验证规则的函数 `verifyRule` 中，每个规则的 `ecCount` 的数目是在函数的开始计算的，在计算结束之后输出到文件流中即可。

```
4 packetClassSearchTime = useCtime;
5
6 ecCount = vFinalPacketClasses.size();
7 if(ecCount == 0)
8 {
9     fprintf(stderr, "[VeriFlow::verifyRule] Error in rule: %s\n", rule.toString().c_str());
10    fprintf(stderr, "[VeriFlow::verifyRule] Error: (ecCount = vFinalPacketClasses.size() = 0). Terminating process.\n");
11    exit(1);
12 }
13 else
14 {
15     // fprintf(stdout, "\n");
16     fprintf(fp, "[VeriFlow::verifyRule] ecCount: %lu\n\n", ecCount);
17     fprintf(stdout, "[VeriFlow::verifyRule] ecCount: %lu\n\n", ecCount);
18 }
19
20 // fprintf(stdout, "[VeriFlow::verifyRule] Generating forwarding graphs...\n");
21 gettimeofday(&start, NULL);
```

## 2. 环路输出

打印出环路信息：首先锁定 `traverseForwardingGraph` 函数，该函数用来遍历一个固定的 EC 集合并计算转发图，如果找到环路之后就打印相关信息。这里看了下 `unordered_set` 的作用，主要是用来判断第一个出现重复元素的地方，说明在这之前的函数的递归调用时没有重复元素的，所以可以用 `vector` 等动态数组来记录插入时的顺序

```
C:\Users\DELL\Desktop\SDN\第四次资料\VeriFlow.cpp - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
VeriFlow.cpp VeriFlow.h EquivalenceClass.cpp
1109 if(currentLocation.compare("") == 0)
1110 {
1111     return true;
1112 }
1113 bool existed = false;
1114 for(int i = 0; i < visited.size(); i++){
1115     if(currentLocation == visited.at(i)){
1116         existed = true;
1117         break;
1118     }
1119 }
1120 if(existed)
1121 {
1122     // Found a loop.
1123     visited.push_back(currentLocation);
1124     fprintf(fp, "\n");
1125     fprintf(fp, "[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node %s.\n", currentLocation.c_str());
1126     fprintf(fp, "[VeriFlow::traverseForwardingGraph] PacketClass: %s\n", packetClass.toString().c_str());
1127     fprintf(fp, "[VeriFlow::traverseForwardingGraph] Loop path is:\n");
1128
1129     fprintf(fp, "\n");
1130     fprintf(stdout, "[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class at node %s.\n", currentLocation.c_str());
1131     fprintf(stdout, "[VeriFlow::traverseForwardingGraph] PacketClass: %s\n", packetClass.toString().c_str());
1132     fprintf(stdout, "[VeriFlow::traverseForwardingGraph] Loop path is:\n");
1133
1134     fprintf(fp, "%s", visited.at(0).c_str());
1135     cout << visited.at(0);
1136     for(int i = 1; i < visited.size(); i++){
1137         fprintf(fp, " -> %s", visited.at(i).c_str());
1138         cout << " <- " << visited.at(i);
1139     }
1140     fprintf(fp, "\n");
1141     cout << endl;
1142
1143     for(unsigned int i = 0; i < faults.size(); i++) {
1144         if (packetClass.subsumes(faults[i])) {
1145             faults.erase(faults.begin() + i);
1146             i--;
1147         }
1148     }
1149     faults.push_back(packetClass);
1150
1151     return false;
1152 }
1153
1154 // visited.insert(currentLocation);
1155 visited.push_back(currentLocation);
1156 // cout << currentLocation << endl;
1157 if(graph->links.find(currentLocation) == graph->links.end())
1158 {
1159     // Found a black hole.
1160     fprintf(fp, "\n");
1161     fprintf(fp, "[VeriFlow::traverseForwardingGraph] Found a BLACK HOLE for the following packet class as current location (it's not found in the graph).\n", currentLocation.c_str());
1162 }
```

这里的环路的最终的结点没有正常打印，这是因为在 trans 函数中插入节点的位置存在问题，在函数调用的最开始插入数据即可显示出完整的环路信息。

```
bool verifyRule(const Rule& rule, int command, double& updateTime, double& packetClassSearchTime, double& graphBuildTime, double& queryTime, unsigned long& ecCount, FILE* fp){
bool traverseForwardingGraph(const EquivalenceClass& packetClass, ForwardingGraph* graph, const string& currentLocation, const string& lastHop, vector < string > visited, FILE* fp){

int getTotalRuleCount() const;

void setDataPathId(unsigned short socketPort, uint64_t datapathId);
```

### 3. 五元组打印

五元组信息打印：这里主要是通过修改 EC 等价类的 to\_string 函数进行打印，涉及到对应的 EquivalenceClass.cpp 中函数，主要是过滤出来需要的五元组即可。

```

109
110 string EquivalenceClass::toString() const
111 {
112     char buffer[1024];
113     sprintf(buffer, "nw_src (%s - %s), nw_dst (%s - %s)",
114             ::getIpValueAsString(this->lowerBound[NW_SRC]).c_str(),
115             ::getIpValueAsString(this->upperBound[NW_SRC]).c_str(),
116             ::getIpValueAsString(this->lowerBound[NW_DST]).c_str(),
117             ::getIpValueAsString(this->upperBound[NW_DST]).c_str());
118     string retVal = buffer;
119     retVal += ", ";
120
121     sprintf(buffer, "nw_proto (%lu - %lu), tp_src (%lu - %lu), tp_dst (%lu - %lu)",
122             this->lowerBound[10], this->upperBound[10],
123             this->lowerBound[12], this->upperBound[12],
124             this->lowerBound[13], this->upperBound[13]);
125
126     retVal += buffer;
127
128     return retVal;
129     // char buffer[1024];
130     // sprintf(buffer, "[EquivalenceClass] dl_src (%lu-%s, %lu-%s), dl_dst (%lu-%s, %lu-%s)",
131     //         this->lowerBound[DL_SRC], ::getMacValueAsString(this->lowerBound[DL_SRC]).c_str(),
132     //         this->upperBound[DL_SRC], ::getMacValueAsString(this->upperBound[DL_SRC]).c_str(),
133     //         this->lowerBound[DL_DST], ::getMacValueAsString(this->lowerBound[DL_DST]).c_str(),

```

#### 实验四:

这里可以看到，主要的变化是对规则的输入端口的设置以及对应在比

较时增加对端口的比较，对每个规则都增加相应的输入端口的过滤来

进一步处理。通过对比在控制台的输出，可以看到增加补丁之后的

faults size 是变小的，可以认为是新的补丁可以减少对于错误的判断，

只修复关键的节点，减少消耗。

```

@@ -292,10 +292,8 @@ void OpenFlowProtocolMessage::processFlowRemoved(const char* data, ProxyConnecti
rule.type = FORWARDING;
rule.wildcards = ntohl(ofr->match.wildcards);

- rule.fieldValue[IN_PORT] = "0"; // ::convertIntToString(ntohs(ofr->match.in_port));
- rule.fieldMask[IN_PORT] = "0"; // ((rule.wildcards == OFFFW_ALL) || ((rule.wildcards & OFFFW_IN_PORT) != 0)) ? "0" : "65535";
+ rule.in_port = ntohs(ofr->match.in_port);
+ rule.fieldValue[IN_PORT] = ::convertIntToString(ntohs(ofr->match.in_port));
+ rule.fieldMask[IN_PORT] = ((rule.wildcards == OFFFW_ALL) || ((rule.wildcards & OFFFW_IN_PORT) != 0)) ? "0" : "65535";

rule.fieldValue[DL_SRC] = ::getMacValueAsString(ofr->match.dl_src);
rule.fieldMask[DL_SRC] = ((rule.wildcards == OFFFW_ALL) || ((rule.wildcards & OFFFW_DL_SRC) != 0)) ? "0:0:0:0:0:0" : "FF:FF:FF:F:FF:FF";
@@ -348,13 +346,12 @@ void OpenFlowProtocolMessage::processFlowRemoved(const char* data, ProxyConnecti

```

```

@@ -35,7 +34,8 @@ Rule::Rule(const Rule& other)
this->location = other.location;
this->nextHop = other.nextHop;
this->in_port = other.in_port;
this->priority = other.priority;
// this->outPort = other.outPort;
}
@@ -181,7 +179,6 @@ bool Rule::equals(const Rule& other) const
if((this->type == other.type)
    && (this->wildcards == other.wildcards)
    && (this->location.compare(other.location) == 0)
    && (this->in_port == other.in_port)
    // && (this->nextHop.compare(other.nextHop) == 0) // Not present in OFPT_FLOW_REMOVED messages.
    && (this->priority == other.priority)
    // && (this->outPort == other.outPort) // Not used in this version.
@@ -225,7 +222,6 @@ int Rule::operator()() const

```

```

const list< ForwardingLink >& linkList = graph->links[currentLocation];
list< ForwardingLink >::const_iterator itr = linkList.begin();
// input_port as a filter
if(lastHop.compare("NULL") == 0 || itr->rule.in_port == 65536){
    // do nothing
}
else{
    while(itr != linkList.end()){
        string connected_hop = network.getNextHopIpAddress(currentLocation, itr->rule.in_port);
        if(connected_hop.compare(lastHop) == 0) break;
        itr++;
    }

    if(itr == linkList.end()){
        // Found a black hole.
        fprintf(fp, "\n");
        fprintf(fp, "[VeriFlow::traverseForwardingGraph] Found a BLACK HOLE for the following packet class as there is no outgoing link at current location (%s),\n",
            currentLocation.c_str());
        fprintf(fp, "[VeriFlow::traverseForwardingGraph] PacketClass: %s\n", packetClass.toString().c_str());
        for(unsigned int i = 0; i < faults.size(); i++) {
            if (packetClass.subsumes(faults[i])) {
                faults.erase(faults.begin() + i);
                i--;
            }
        }
        faults.push_back(packetClass);
        return false;
    }

    if(itr->isGateway == true)
    {
228,7 +1187,7 @@ bool VeriFlow::traverseForwardingGraph(const EquivalenceClass& packetClass, Form
        fprintf(fp, "[VeriFlow::traverseForwardingGraph] PacketClass: %s\n", packetClass.toString().c_str()); */

        return this->traverseForwardingGraph(packetClass, graph, itr->rule.nextHop, currentLocation, visited, fp);
        return this->traverseForwardingGraph(packetClass, graph, itr->rule.nextHop, visited, fp);
    }
}

```

结果展示:

实验一:

实验二:

```
ca@ubuntu: ~/Desktop/ryu/ryu/app
127.0.0.1 - - [13/Jun/2021 08:03:20] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000777
(2718) accepted ('127.0.0.1', 47094)
127.0.0.1 - - [13/Jun/2021 08:03:20] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.001231
(2718) accepted ('127.0.0.1', 47096)
127.0.0.1 - - [13/Jun/2021 08:03:20] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000569
(2718) accepted ('127.0.0.1', 47098)
127.0.0.1 - - [13/Jun/2021 08:03:20] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000755
(2718) accepted ('127.0.0.1', 47100)
127.0.0.1 - - [13/Jun/2021 08:03:20] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000754
(2718) accepted ('127.0.0.1', 47104)
127.0.0.1 - - [13/Jun/2021 08:03:20] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.001122
(2718) accepted ('127.0.0.1', 47106)

ca@ubuntu: ~/Desktop/BEADS/veriflow/VeriFlow
at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221582-01:80:c2:00:00:0e, 165252221582-01:80:c2:00:00:0e), nw_src (167772160-10.0.0.0, 167772415-10.0.0.255), nw_dst (167772160-10.0.0.0, 167772415-10.0.0.255), Field 0 (0, 65535), Field 1 (0, 281474976710655), Field 2 (165252221582, 165252221582), Field 3 (2048, 2048), Field 4 (0, 4095), Field 5 (0, 7), Field 6 (0, 1048575), Field 7 (0, 7), Field 8 (167772160, 167772415), Field 9 (167772160, 167772415), Field 10 (0, 255), Field 11 (0, 63), Field 12 (0, 65535), Field 13 (0, 65535)
[VeriFlow::traverseForwardingGraph] Loop path is:
20.0.0.25 <- 20.0.0.7 <- 20.0.0.16 <- 20.0.0.9 <- 20.0.0.22 <- 20.0.0.23 <- 20.0.0.1 <- 20.0.0.25
[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class
at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: [EquivalenceClass] dl_src (0-00:00:00:00:00:00, 281474976710655-ff:ff:ff:ff:ff:ff), dl_dst (165252221583-01:80:c2:00:00:0f, 281474976710655-ff:ff:ff:ff:ff:ff), nw_src (167772160-10.0.0.0, 167772415-10.0.0.255), nw_dst (167772160-10.0.0.0, 167772415-10.0.0.255), Field 0 (0, 65535), Field 1 (0, 281474976710655), Field 2 (165252221583, 281474976710655), Field 3 (2048, 2048), Field 4 (0, 4095), Field 5 (0, 7), Field 6 (0, 1048575), Field 7 (0, 7), Field 8 (167772160, 167772415), Field 9 (167772160, 167772415)

ca@ubuntu: ~/Desktop/mininet/mininet/examples
SRI 10.0.0.19
Stanford 10.0.0.20
Tinker 10.0.0.21
UCLA 10.0.0.22
UCSB 10.0.0.23
USC 10.0.0.24
UTAH 10.0.0.25
*** Type 'exit' or control-C to shut down network
*** For testing network connectivity among the hosts, wait a bit for the control
ler to create all the routes, then do 'pingall' on the mininet console.
*** edited for xjtu sdn_exp_2020
*** Starting CLI:
mininet> SDC ping MIT
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data.
64 bytes from 10.0.0.12: icmp_seq=1 ttl=64 time=214 ms
64 bytes from 10.0.0.12: icmp_seq=2 ttl=64 time=234 ms
64 bytes from 10.0.0.12: icmp_seq=3 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=4 ttl=64 time=236 ms
64 bytes from 10.0.0.12: icmp_seq=5 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=6 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=7 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=8 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=9 ttl=64 time=236 ms

ca@ubuntu: ~/Desktop/mininet/mininet/examples
[sudo] password for ca:
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
Install waypoint path: 23 -> 1
23 -> 4:s22:2 -> 2:s9:3 -> 3:s16:2 -> 3:s7:2 -> 3:s25:2 -> 1
ca@ubuntu: ~/Desktop/mininet/mininet/examples
```

实验三:

```
ca@ubuntu: ~/Desktop/ryu/ryu/app
127.0.0.1 - - [13/Jun/2021 08:05:40] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000617
(6058) accepted ('127.0.0.1', 51510)
127.0.0.1 - - [13/Jun/2021 08:05:40] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000913
(6058) accepted ('127.0.0.1', 51512)
127.0.0.1 - - [13/Jun/2021 08:05:40] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000902
(6058) accepted ('127.0.0.1', 51514)
127.0.0.1 - - [13/Jun/2021 08:05:40] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.002448
(6058) accepted ('127.0.0.1', 51516)
127.0.0.1 - - [13/Jun/2021 08:05:40] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000883
(6058) accepted ('127.0.0.1', 51518)

ca@ubuntu: ~/Desktop/BEADS/veriflow/VeriFlow
dst (10.0.0.0 - 10.0.0.255), nw_proto (0 - 255), tp_src (0 - 65535), tp_dst (65535)
[VeriFlow::traverseForwardingGraph] Loop path is:
20.0.0.25 <- 20.0.0.7 <- 20.0.0.16 <- 20.0.0.9 <- 20.0.0.22 <- 20.0.0.23 <- 20.0.0.1 <- 20.0.0.25
[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class
at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0 - 10.0.0.255), nw_dst (10.0.0.0 - 10.0.0.255), nw_proto (0 - 255), tp_src (0 - 65535), tp_dst (0 - 65535)
[VeriFlow::traverseForwardingGraph] Loop path is:
20.0.0.25 <- 20.0.0.7 <- 20.0.0.16 <- 20.0.0.9 <- 20.0.0.22 <- 20.0.0.23 <- 20.0.0.1 <- 20.0.0.25
[VeriFlow::traverseForwardingGraph] Found a LOOP for the following packet class
at node 20.0.0.25.
[VeriFlow::traverseForwardingGraph] PacketClass: nw_src (10.0.0.0 - 10.0.0.255), nw_dst (10.0.0.0 - 10.0.0.255), nw_proto (0 - 255), tp_src (0 - 65535), tp_dst (0 - 65535)
[VeriFlow::traverseForwardingGraph] Loop path is:
20.0.0.25 <- 20.0.0.7 <- 20.0.0.16 <- 20.0.0.9 <- 20.0.0.22 <- 20.0.0.23 <- 20.0.0.1 <- 20.0.0.25

ca@ubuntu: ~/Desktop/mininet/mininet/examples
McLellan 10.0.0.14
NBS 10.0.0.15
RADC 10.0.0.16
RANO 10.0.0.17
SDC 10.0.0.18
SRI 10.0.0.19
Stanford 10.0.0.20
Tinker 10.0.0.21
UCLA 10.0.0.22
UCSB 10.0.0.23
USC 10.0.0.24
UTAH 10.0.0.25
*** Type 'exit' or control-C to shut down network
*** For testing network connectivity among the hosts, wait a bit for the control
ler to create all the routes, then do 'pingall' on the mininet console.
*** edited for xjtu sdn_exp_2020
*** Starting CLI:
mininet> SDC ping MIT
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data.
64 bytes from 10.0.0.12: icmp_seq=2 ttl=64 time=250 ms
64 bytes from 10.0.0.12: icmp_seq=3 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=4 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=5 ttl=64 time=235 ms

ca@ubuntu: ~/Desktop/mininet/mininet/examples
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
Install waypoint path: 23 -> 1
23 -> 4:s22:2 -> 2:s9:3 -> 3:s16:2 -> 3:s7:2 -> 3:s25:2 -> 1
ca@ubuntu: ~/Desktop/mininet/mininet/examples$ sudo python waypoint_path.py
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
[Response [200]]
```

实验四:



```
ca@ubuntu: ~/Desktop/sdn/BEADS/veriflow/VeriFlow
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 5
faults size: 8
faults size: 11
faults size: 14
faults size: 14
faults size: 17
faults size: 17
faults size: 17
Removing fault!
Removing fault!
Removing fault!
Removing fault!
faults size: 14
faults size: 14

ca@ubuntu:~/Desktop/mininet/mininet/examples$ sudo python waypoint_path.py
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
<Response [404]>
Install waypoint path: 23 -> 1
23 -> 4:52:22 -> 2:59:13 -> 3:51:6:2 -> 3:57:22 -> 3:52:5:2 -> 1

ca@ubuntu:~/Desktop/mininet/mininet/examples$ make clean all
make: *** No rule to make target 'clean'. Stop.
ca@ubuntu:~/Desktop/mininet/mininet/examples$ make clean all
make: *** No rule to make target 'clean'. Stop.
ca@ubuntu:~/Desktop/mininet/mininet/examples$ sudo python waypoint_path.py

ca@ubuntu: ~/Desktop/ryu/ryu/app
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000437
<Response [200]>
(36763) accepted ('127.0.0.1', 34706)
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000453
<Response [200]>
(36763) accepted ('127.0.0.1', 34708)
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000432
<Response [200]>
(36763) accepted ('127.0.0.1', 34710)
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000611
<Response [200]>
(36763) accepted ('127.0.0.1', 34712)
127.0.0.1 - - [12/Jun/2021 09:16:04] "POST /stats/flowentry/add HTTP/1.1" 200 13
9 0.000505

ca@ubuntu: ~/Desktop/mininet/mininet/examples
UCSB 10.0.0.23
USC 10.0.0.24
UTAH 10.0.0.25
*** Type 'exit' or control-D to shut down network
*** For testing network connectivity among the hosts, wait a bit for the control
ler to create all the routes, then do 'pingall' on the mininet console.
*** edited for kxju sdn_exp_2020
*** Starting CLI:
mininet> ^C

Interrupt
mininet> SDC ping MIT
PING 10.0.0.12 (10.0.0.12) 56(84) bytes of data:
64 bytes from 10.0.0.12: icmp_seq=2 ttl=64 time=170 ms
64 bytes from 10.0.0.12: icmp_seq=3 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=4 ttl=64 time=236 ms
64 bytes from 10.0.0.12: icmp_seq=5 ttl=64 time=234 ms
64 bytes from 10.0.0.12: icmp_seq=6 ttl=64 time=235 ms
64 bytes from 10.0.0.12: icmp_seq=7 ttl=64 time=236 ms
64 bytes from 10.0.0.12: icmp_seq=8 ttl=64 time=236 ms
64 bytes from 10.0.0.12: icmp_seq=9 ttl=64 time=237 ms
64 bytes from 10.0.0.12: icmp_seq=10 ttl=64 time=233 ms
64 bytes from 10.0.0.12: icmp_seq=11 ttl=64 time=234 ms
```