Final Project Presentation

Simulation of Packet Classification on Network Simulator (NS2)

- Yang Li
- Fatin G. Sabbagha
- Jinzhu Chen
Outline

- Introduction
- What is NS2?
- Packet Classification
- Related Algorithms
- Simulation Design
- Experimental Result and Evaluation
- Conclusion
- Demo
Introduction

- What is packet classification?
  - Categorizing the packet into different flows
- Why do we need packet classification?
  - Firewall packet filter
  - Provide different services VPNs
  - QoS guarantees
  - Intrusion detection, DoS, etc.
- How does the classification work?
  - Identify the flows by multiple header fields of packet
  - Classifier compare header of each incoming packet with a set of predefined rules
Packet Classification Example[1]

From ‘Algorithms for Packet Classification’, Pankaj Gupta and Nick McKeown, Stanford University

<table>
<thead>
<tr>
<th>Flow</th>
<th>Relevant packet fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email and from ISP2</td>
<td>Source link-layer address, source transport port number</td>
</tr>
<tr>
<td>From ISP2</td>
<td>Source link-layer address</td>
</tr>
<tr>
<td>From ISP3 and going to E2</td>
<td>Source link-layer address, destination network-layer address</td>
</tr>
</tbody>
</table>
Packet Classification Example cont.

Rules with four header fields

<table>
<thead>
<tr>
<th>Rule</th>
<th>Network-layer destination (address/mask)</th>
<th>Network-layer source (address/mask)</th>
<th>Transport-layer destination</th>
<th>Transport-layer protocol</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>152.163.190.69/255.255.255.255</td>
<td>152.163.80.11/255.255.255.255</td>
<td>*</td>
<td>*</td>
<td>Deny</td>
</tr>
<tr>
<td>$R_2$</td>
<td>152.168.3.0/255.255.255.0</td>
<td>152.163.200.157/255.255.255.255</td>
<td>eq www</td>
<td>udp</td>
<td>Deny</td>
</tr>
<tr>
<td>$R_5$</td>
<td>152.163.198.4/255.255.255.255</td>
<td>152.163.160.0/255.255.252.0</td>
<td>gt 1023</td>
<td>tcp</td>
<td>Permit</td>
</tr>
<tr>
<td>$R_6$</td>
<td>0.0.0.0/0.0.0.0</td>
<td>0.0.0.0/0.0.0.0</td>
<td>*</td>
<td>*</td>
<td>Permit</td>
</tr>
</tbody>
</table>

Classification results

<table>
<thead>
<tr>
<th>Packet header</th>
<th>Network-layer destination</th>
<th>Network-layer source</th>
<th>Transport-layer destination</th>
<th>Transport-layer protocol</th>
<th>Best matching rule, action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>152.163.190.69</td>
<td>152.163.80.11</td>
<td>www</td>
<td>tcp</td>
<td>$R_1$, deny</td>
</tr>
<tr>
<td>$P_2$</td>
<td>152.168.3.21</td>
<td>152.163.200.157</td>
<td>www</td>
<td>udp</td>
<td>$R_2$, deny</td>
</tr>
<tr>
<td>$P_3$</td>
<td>152.168.198.4</td>
<td>152.163.160.10</td>
<td>1024</td>
<td>tcp</td>
<td>$R_5$, permit</td>
</tr>
</tbody>
</table>
Our Approach and Contributions

• Our Approach
  • Develop the algorithm module in C++
  • Design the simulating network topology
  • Design and generate the packet header and rules
  • Build the simulating environment, record the performance result.

• Our Contributions
  • Simulating linear search and BDD classify algorithms on NS2
  • Patching the algorithm module in original NS2 system
  • Evaluating the performance according to the searching time
Network Simulator (NS2)

- What is NS2?
  - An opensource packet-level simulator
  - For standard experiment environment
  - Simulate wired and wireless network
  - Integrated with graphical monitoring tools (nam, xgraph)
  - Using C++ and OTcl as the developing language
    - OTCL: Setup the topology and network configuration
    - C++: Extend the transmission protocols or routing algorithms
Related Algorithms

Algorithms of Packet Classification:
- Linear Search
- Binary Decision Diagram (BDD)
- Firewalls Decision Diagram (FDD)
- Hierarchical Intelligent Cutting (Hicut)
- Multidimensional Cutting (Hypercut)
- Ternary Content Addressable Memory (TCAM)
Linear Search

- Match packet against rules in order of priority
  - First match is highest matching rule
- Space Complexity: $O(N)$
  - Simple list of rules
- Time Complexity: $O(Nd)$
  - Linear growth with number of rules
- Characteristic:
  - Sequential search, simple
  - Poor scalability
### Example for Linear Search

<table>
<thead>
<tr>
<th>Rule#</th>
<th>Src ip</th>
<th>Src port</th>
<th>Dst ip</th>
<th>Dst port</th>
<th>Protocol</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule1</td>
<td>192.168.0.0/16</td>
<td>0-65536</td>
<td>10.0.0.0/24</td>
<td>0-65536</td>
<td>TCP</td>
<td>accept</td>
</tr>
<tr>
<td>Rule2</td>
<td>100.10.10.0/24</td>
<td>0-65536</td>
<td>159.0.0.0/8</td>
<td>0-65536</td>
<td>UDP</td>
<td>discard</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Incoming packet

<table>
<thead>
<tr>
<th>Packet#</th>
<th>Src ip</th>
<th>Src port</th>
<th>Dst ip</th>
<th>Dst port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pkt1</td>
<td>192.168.20.10</td>
<td>25</td>
<td>10.0.0.25</td>
<td>14</td>
<td>TCP</td>
</tr>
</tbody>
</table>

192.168.20.10 & 255.255.0.0 → 192.168.0.0 & 25 ∈ (0, 65536)

10.0.0.25 & 255.255.255.0 → 10.0.0.0 & 14 ∈ (0, 65536)

**Decision? Accept**
Binary Decision Diagram (BDD)

- **BDD**: A directed binary tree for classification.
  - Convert the rules into binary string
  - Build a binary tree according to the rules
  - Search in directed paths for matching
  - If no match found, take default decision

- **Performance**
  - Time complexity: $O(\log n)$
  - No matter how many rules we have!
  - No redundant rules in the tree.
### BDD Example

#### Table

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Where

1 means accept
0 means discard

--- means 1
- - - - means 0

Packet = 1011
Implementation

- Packet Generation
- Network Construction
- Tcl Script
- “command”
- Classifier Integration
- Get Object Reference
Packet Header Structure

- Contiguous address space for headers
- Protocol specific header’s offset varied
- Three approaches
  - Add flow attributes in payload
  - Add new type of header
  - Modify the existing header
Network Construction

- CBR Agent
  - Input generated packets
  - Attach agent

- Pkt_Generator
  - Generate the packet and attach on CBR agent to send

- UDP Agent (Sending)
  - Connect
  - Attach agent

- NULL Agent (Receiving)
  - Protocol Agent to construct the pkt
  - Packet forwarding through links

- Node1

- Node2
  - port_classifier
  - addr_classifier

- Node3

- Link 1
- Link 3
Node Structure
Sample Tcl script

```
set ns [new Simulator] #Create a simulator object
$ns tpot2 DV #Tell the simulator to use dynamic routing
#Open the nam trace file
...
#Define a 'finish' procedure
proc finish {} {
    ...
}
#Create two nodes
set n(0) [ns node]
set n(1) [ns node]

#Create links between the nodes
$ns duplex-link $n(0) $n(1) 1Mb 10ms DropTail

#Create a UDP agent and attach it to node n(0)
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

#Create a Null agent (a traffic sink) and attach it to node n(1)
set null0 [new Agent/UDP]
$ns attach-agent $n(1) $null0

$ns connect $udp0 $null0 #Connect the traffic source with the traffic sink

#Schedule events for the CBR agent and the network dynamics
$ns at 0.5 "$cbr0 start"
$ns at 9.5 "$cbr0 stop"
$ns at 10.0 "finish"

$ns run #Run the simulation
```
“command”

- Interact with “command” function
  - In NS2, for each TclObject, like node, agent and classifier, there is an interface function “command” which will be understood by script interpreter.
  - Example: `%n(1) address?`
    - This will implicitly call the “command” of the n(1) object and pass the “address?” as an argument.
  - We use “command” to configure our objects.
Classifier Integration

- Classifier is TclObject - configure through "command"
- Original hash classifier is modified
  - Linear and BDD classifier object are added to hash classifier.

```cpp
class Classifier : public NsObject {
    ... public:
    LinearClsfr in_lncslfr_;
    LinearClsfr out_lncslfr_
    BinaryClsfr in_bddclsfr_
    BinaryClsfr out_bddclsfr_
    int type_classifier;
    int classifier_name;
    char in_acl_file[80];
    char out_acl_file[80];
};

int DestHashClassifier::command(int argc, const char* argv) {
    ... } else if (argc == 5) {
        if (strcmp(argv[1], "in-acl-file") == 0) {
            strcpy(in_acl_file, argv[2]);
            if (strcmp(argv[3], "linear") == 0) {
                type_classifier = 0;
                in_lncslfr_.initializeRules(in_acl_file);
            } else if (strcmp(argv[3], "bdd") == 0) {
                type_classifier = 1;
                in_bddclsfr_.initializeRules(in_acl_file);
            }
        classifier_name = atoi(argv[4]);
        fprintf(stderr, "classifier_name: %d\n", classifier_name);
        return TCL_OK;
    } return TCL_OK;
} // command
```
Get Object Reference

- The object reference is the interface where we can access the object through Tcl interpreter.
- Classifier is attached on the Entry point
- \%set en [\$n(1) entry]
  - This returns the entry point reference of n(1) to “en”
- Example of classifier
  ```tcl
  set entry_clsfr1 [\$n(1) entry]
  \$entry_clsfr1 in-acl-file node_ACL_1.txt linear 1
  ```
- Example of Packet generation
  ```tcl
  set udp0 [new Agent/UDP]
  \$udp0 msg-file node_packet_1.txt
  ```
Simulation Design

Network Topology

• Two subnets
• Routers with three ACLs
• Packet flows from S1 to S2
Experimental Result and Evaluation

- Initialization Time:
  - **Linear Search:** 0.001053 s
  - **BDD:** 0.580812 s

- Execution time – time to run the search function
- Rule index – the rule index that matches the packet
  - **Linear Search:** index of the rule in the ACL table
  - **BDD:** depth in the constructed binary tree
Experimental Result and Evaluation

- Linear search shows more variance for different packet

Figure 13. Linear Classification - Average execution time for each packet

Figure 14. Binary Decision Classification - Average execution time for each packet
Experimental Result and Evaluation

- Linear search shows trend of increasing execution time for the rule index increasing.
Conclusion

- We implemented Linear search and BDD classifier algorithms and integrated into NS2 system
- Designed the experimental topology and rules to test the classifier
- With Nam, we visualized the packet transmission and classification process
- Finally, we evaluate our experimental result.

Future work
- Large rule set support
- BDD tree initializing time optimization
- More classification algorithms support
Figure 8. Linear Classification - Stage 1

Figure 9. Linear Classification - Stage 2

Figure 10. Linear Classification - Stage 3
Questions?