

Towards Optimal Firewall Rule Ordering Utilizing Directed Acyclical Graphs

Ashish Tapdiya *and* Errin W. Fulp



Department of Computer Science

supported by

US Department of Energy



GreatWall Systems



Network Firewalls and Performance

- Firewalls remain the forefront defense for computer systems
 - Inspect packets sent between networks
 - Provide access control, auditing, and traffic control
 - Actions performed on packets specified by security policy
- Policy is a list of rules, specifying which packets to accept or deny
 - Policies are increasing in length and complexity
 - For list oriented implementations, complexity and length impacts performance
 - *Optimize the policy*, organize the rules to find a match quickly

Firewall Policies

No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29

- Firewall rule consists of a 5-tuple and an action (IP networks)

$$r = (r[1], r[2], \dots, r[k])$$

- $r[l]$ can be fully specified or contain wildcards ‘*’
- Security policy is a ordered list of rules

$$R = \{r_1, r_2, \dots, r_n\}$$

- Packets sequentially compared with rules until *first match*

Policy Optimization

- Reorder policy rules to reduce the average number of comparisons
 - More *popular* rules should appear first
 - Must maintain policy **integrity**
- Rule list has an implied **precedence relationship**

No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29

- r_1 must appear before r_2 , r_3 , r_5 , and r_5 must be last
- However relative order of r_2 and r_3 can change

Modeling Precedence

- Precedence modeled as a Directed Acyclical Graph (DAG)
 - Nodes are rules, edges are precedence relationships
 - Edge exists between r_i and r_j , if $i < j$ and the rules intersect

Definition

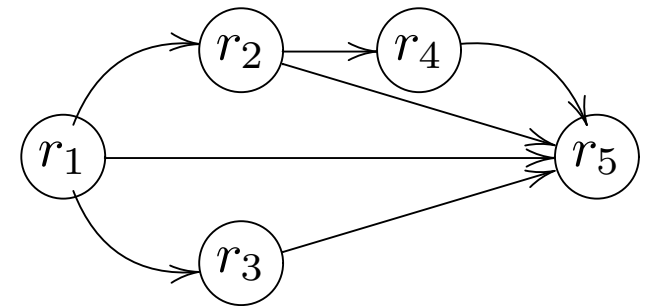
The intersection of rule r_i and r_j , denoted as $r_i \cap r_j$ is

$$r_i \cap r_j = (r_i[l] \cap r_j[l]), \quad l = 1, \dots, k$$

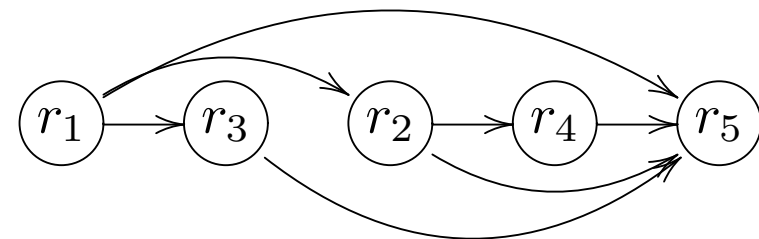
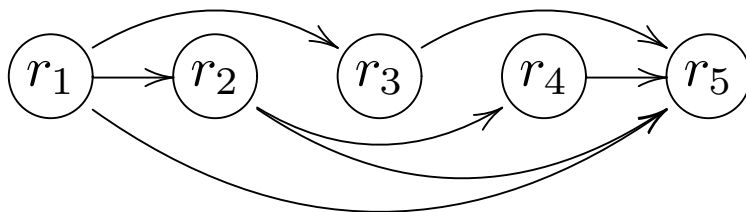
- *Why is the intersection appropriate?*
 - Intersection describes the set of packets that match both rules
 - If rules intersect, then order is significant **for maintaining integrity**

Modeling Precedence

No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29



- Seek a linear arrangement of the DAG to improve performance



- Several linear arrangements, *which is best?*

Optimality Criterion

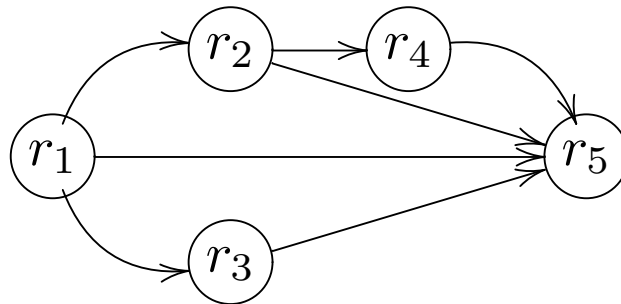
- Every firewall rule has a match probability (*hit ratio*)
 - Develop a *policy profile* over time $\{p_1, \dots, p_n\}$
 - Where p_i is the probability of matching r_i
- Want to minimize the average number of rule comparisons

$$E[n] = \sum_{i=1}^n i \cdot p_i$$

- Determining optimal order is same as job-shop scheduling
 - Job-shop scheduling is \mathcal{NP} -hard so is optimal firewall rule ordering

DAG Properties and Algorithm Variables

- Proposed algorithm will consider sub-graphs from the DAG



- Let $G(r_i)$ be the set of rules dependent on rule r_i
 - For example, $G(r_5) = \{r_1, r_2, r_3, r_4, r_5\}$ and $G(r_4) = \{r_1, r_2, r_4\}$
 - In addition let $G^*(r_i)$ be the set **not** including r_i
- Let $\bar{P}(G(r_i))$ be the average probability of $G(r_i)$
 - This will be the heuristic used to order rules...
- Q is a set of un-sorted rules, S is the list of sorted rules

Sub-Graph Merging Algorithm (SGM)

```
1 while( $Q \neq \emptyset$ )
2   set  $r_b$  to a rule in  $Q$ 
3   for( $\forall r_j \in Q$  and  $r_j \neq r_b$ )
4     if( $\overline{P}(G(r_b)) < \overline{P}(G(r_j))$ )then
5        $r_b \leftarrow r_j$ 
6     end
7   end
8   while( $r_b$  not added to  $S$ )
9     if( $r_b$  has no precedence)then
10      remove  $r_b$  from graph and  $Q$ 
11      add  $r_b$  to  $S$ 
12    else
13      set  $r_t$  to rule in  $G^*(r_b)$ 
14      for( $\forall r_i \in G^*(r_b)$ )
15        if( $\overline{P}(G(r_t)) < \overline{P}(G(r_i))$ )then
16           $r_t \leftarrow r_i$ 
17        end
18      end
19    end
20     $r_b \leftarrow r_t$ 
21  end
22 end
```

Algorithm has two major parts

- Find best *un-sorted* rule (*lines 2-7*)
 - Use average sub-graph probability
- Place best sub-graph rule (*lines 8-21*)
 - If no precedence, place in S
 - Select best rule in sub-sub...-graph

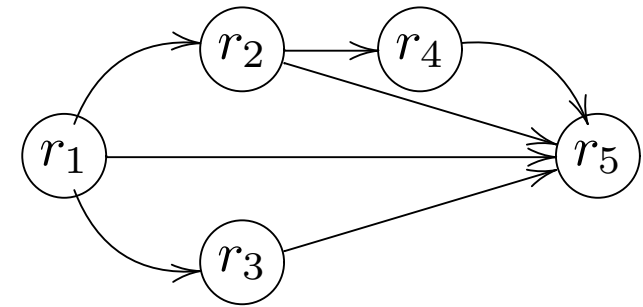
- Does not have to order an entire sub-graph per iteration (*line 8*)
 - This design allows merging of sub-graphs (*but at a cost*)

Integrity and Complexity

- SGM will maintain the integrity of the original policy
 - A rule is only added to S if it has no precedence constraints
 - If a rule has constraints, all preceding rules will be placed first
- Algorithm presented recalculates several values per iteration
 - Data structures used to store intermediate values, fewer updates
 - See algorithm presented in the paper, $O(n) = n^3$

Example SGM Ordering

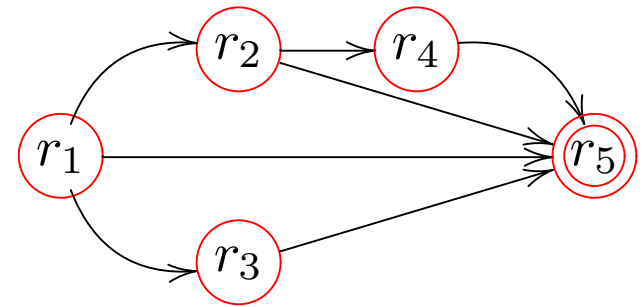
No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29



- ① r_5 is best, but $G^*(r_5) = \{r_1, r_2, r_3, r_4\}$, select from this sub-graph
 - ② r_4 is best, but $G^*(r_4) = \{r_1, r_2\}$, select from this sub-sub-graph
 - ③ r_2 is best, but $G^*(r_2) = \{r_1\}$, select from this sub-sub-sub-graph
 - ④ r_1 is best and $G^*(r_1) = \{\emptyset\}$, so placed in S
- Once the rule is placed in S , repeat the process
 - Final SGM ordering is $\{r_1, r_3, r_2, r_4, r_5\}$

Example SGM Ordering

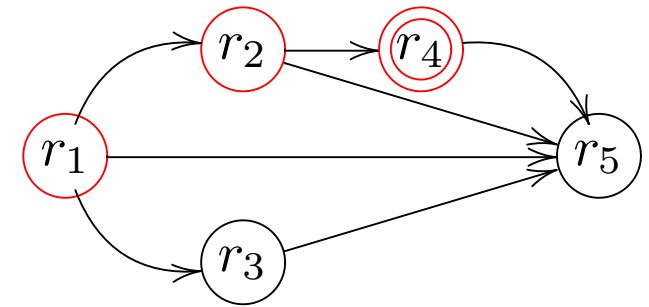
No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29



- ① r_5 is best, but $G^*(r_5) = \{r_1, r_2, r_3, r_4\}$, select from this sub-graph
 - ② r_4 is best, but $G^*(r_4) = \{r_1, r_2\}$, select from this sub-sub-graph
 - ③ r_2 is best, but $G^*(r_2) = \{r_1\}$, select from this sub-sub-sub-graph
 - ④ r_1 is best and $G^*(r_1) = \{\emptyset\}$, so placed in S
- Once the rule is placed in S , repeat the process
 - Final SGM ordering is $\{r_1, r_3, r_2, r_4, r_5\}$

Example SGM Ordering

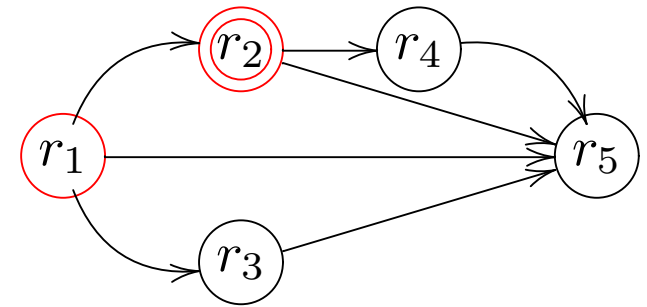
No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29



- ① r_5 is best, but $G^*(r_5) = \{r_1, r_2, r_3, r_4\}$, select from this sub-graph
 - ② r_4 is best, but $G^*(r_4) = \{r_1, r_2\}$, select from this sub-sub-graph
 - ③ r_2 is best, but $G^*(r_2) = \{r_1\}$, select from this sub-sub-sub-graph
 - ④ r_1 is best and $G^*(r_1) = \{\emptyset\}$, so placed in S
- Once the rule is placed in S , repeat the process
 - Final SGM ordering is $\{r_1, r_3, r_2, r_4, r_5\}$

Example SGM Ordering

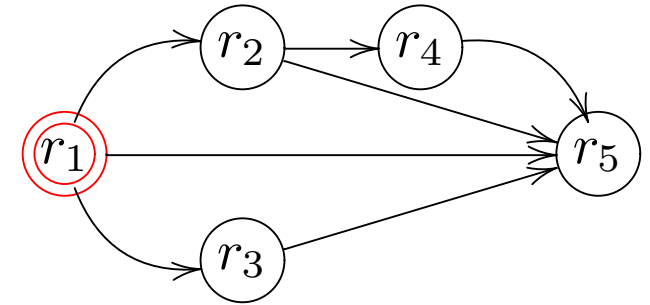
No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29



- 1 r_5 is best, but $G^*(r_5) = \{r_1, r_2, r_3, r_4\}$, select from this sub-graph
 - 2 r_4 is best, but $G^*(r_4) = \{r_1, r_2\}$, select from this sub-sub-graph
 - 3 r_2 is best, but $G^*(r_2) = \{r_1\}$, select from this sub-sub-sub-graph
 - 4 r_1 is best and $G^*(r_1) = \{\emptyset\}$, so placed in S
- Once the rule is placed in S , repeat the process
 - Final SGM ordering is $\{r_1, r_3, r_2, r_4, r_5\}$

Example SGM Ordering

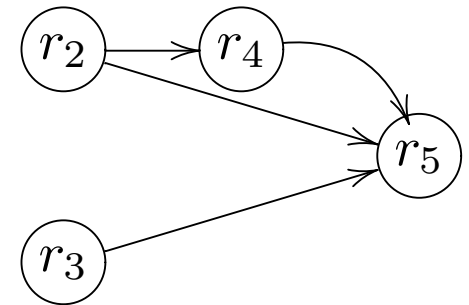
No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29



- ① r_5 is best, but $G^*(r_5) = \{r_1, r_2, r_3, r_4\}$, select from this sub-graph
 - ② r_4 is best, but $G^*(r_4) = \{r_1, r_2\}$, select from this sub-sub-graph
 - ③ r_2 is best, but $G^*(r_2) = \{r_1\}$, select from this sub-sub-sub-graph
 - ④ r_1 is best and $G^*(r_1) = \{\emptyset\}$, so placed in S
- Once the rule is placed in S , repeat the process
 - Final SGM ordering is $\{r_1, r_3, r_2, r_4, r_5\}$

Example SGM Ordering

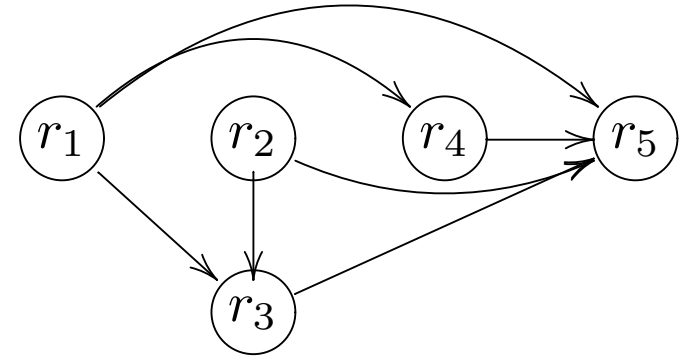
No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0645
2	UDP	190.1.1.*	*	*	90-94	deny	0.161
3	UDP	190.1.2.*	*	*	*	deny	0.2258
4	UDP	190.1.1.2	*	*	94	accept	0.2587
5	*	*	*	*	*	deny	0.29



- ① r_5 is best, but $G^*(r_5) = \{r_1, r_2, r_3, r_4\}$, select from this sub-graph
 - ② r_4 is best, but $G^*(r_4) = \{r_1, r_2\}$, select from this sub-sub-graph
 - ③ r_2 is best, but $G^*(r_2) = \{r_1\}$, select from this sub-sub-sub-graph
 - ④ r_1 is best and $G^*(r_1) = \{\emptyset\}$, so placed in S
- Once the rule is placed in S , repeat the process
 - Final SGM ordering is $\{r_1, r_3, r_2, r_4, r_5\}$

But it does not always work...

No.	Proto.	Source		Destination		Action	Prob.
		IP	Port	IP	Port		
1	UDP	190.1.*	*	*	90	accept	0.0585333
2	UDP	190.1.1.*	*	*	88	deny	0.0728863
3	UDP	190.1.1.2	*	*	88-94	deny	0.156762
4	UDP	190.1.2.*	*	*	*	accept	0.123346
5	*	*	*	*	*	deny	0.5884724



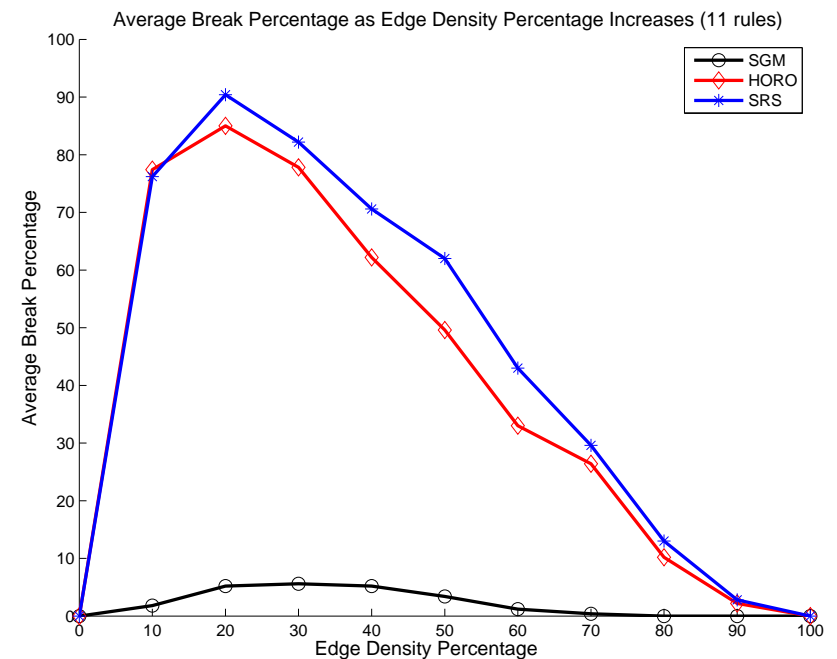
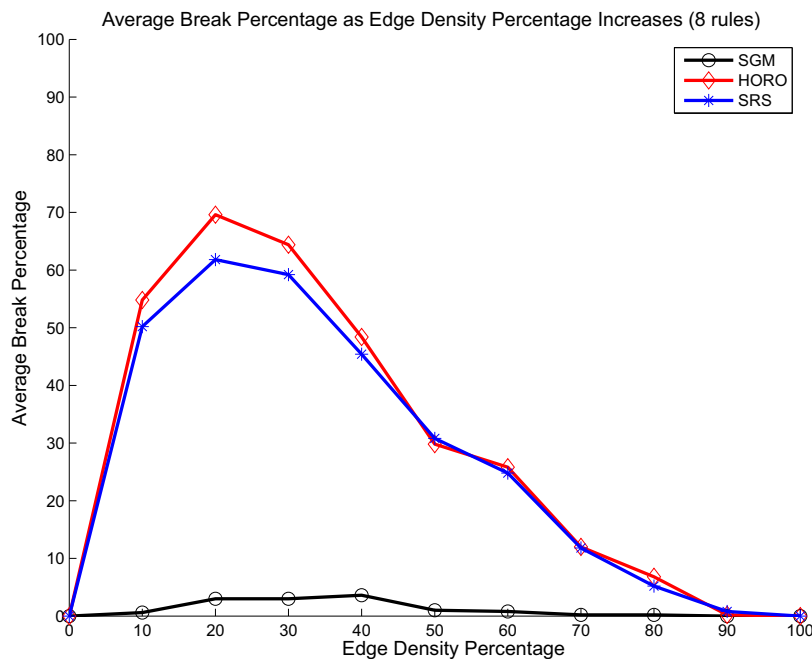
- 1 r_5 is best, but $G^*(r_5) = \{r_1, r_2, r_3, r_4\}$, select from this sub-graph
 - 2 r_3 is best, but $G^*(r_3) = \{r_1, r_2\}$, select from this sub-sub-graph
 - 3 r_2 is best, but this is a mistake
 - It will place r_2 before r_1 , causing r_4 to be fourth...
 - SGM only *considers one graph per iteration*
- Final SGM order is $\{r_2, r_1, r_3, r_4, r_5\}$, optimal is $\{r_1, r_4, r_2, r_3, r_5\}$

Experiments

- Interested in performance given various policies, characterized by
 - Policy size (number of rules), n
 - Number of intersections (number of edges in the DAG), e
 - Policy profile (hit ratio per rule), P
- Performance measured using one of two metrics
 - For a given policy, average number of comparisons
 - For a set of policies, number of break cases
- SGM compared with
 - Optimal ordering (*small policies only*)
 - Simple Rule Sort (SRS)
 - Heuristic Optimal Rule Ordering (HORO)

Edge Density and Break Cases

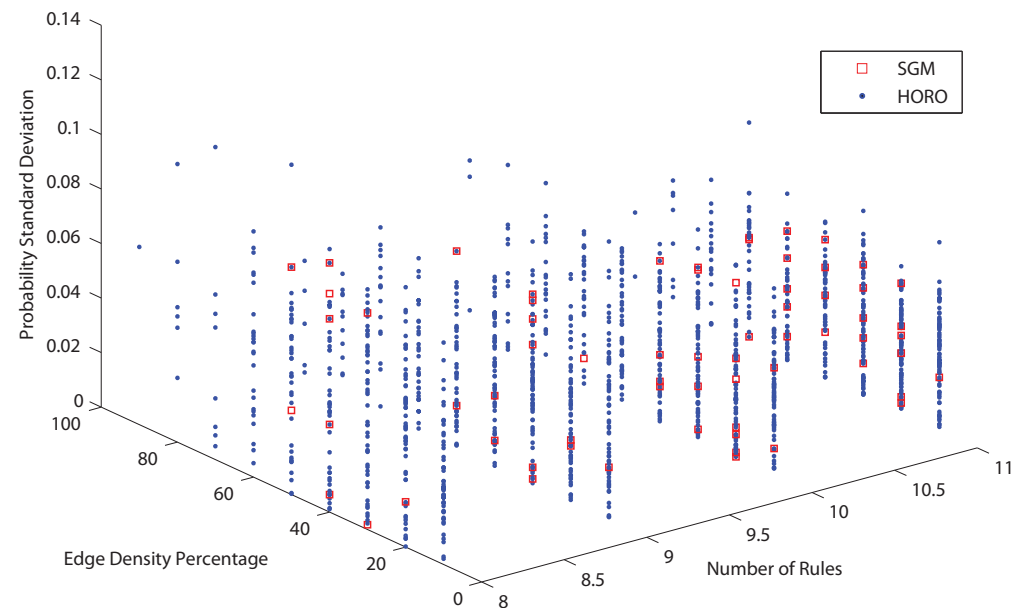
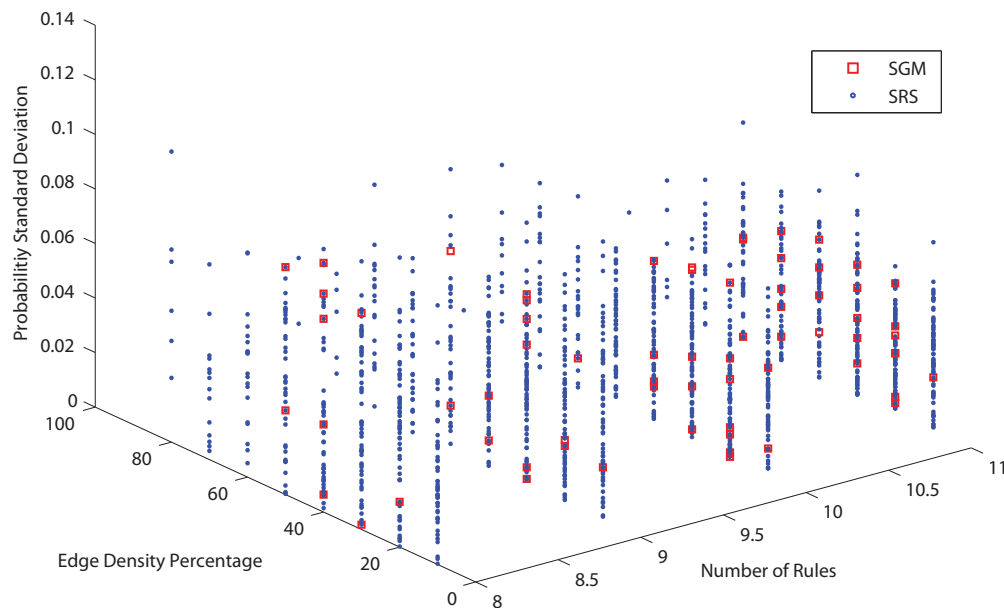
- Edge density, 0% no precedence and 100% completely dependent
 - Compared algorithms to optimal ordering, thus policies are small



- SGM performs best, all have problems with 10 - 40% edge density

Break Case Comparison

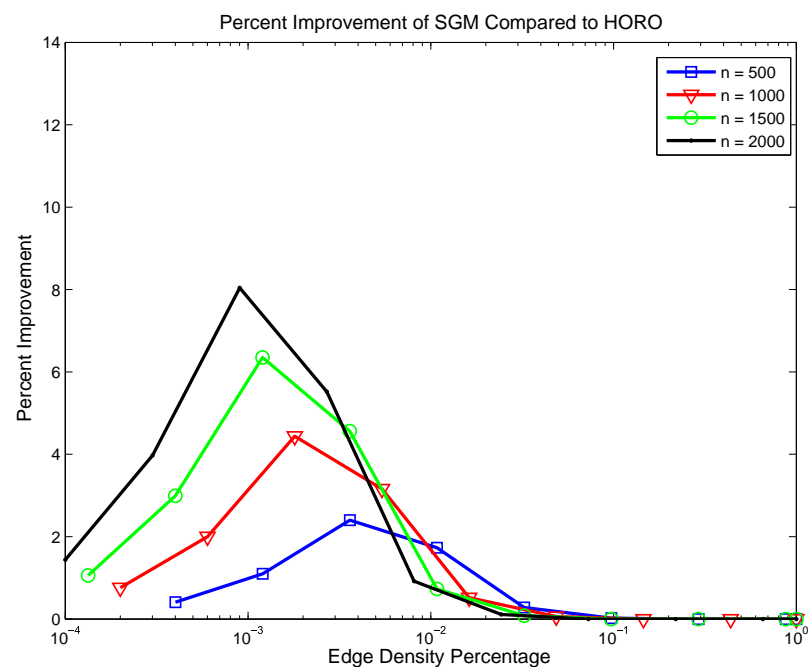
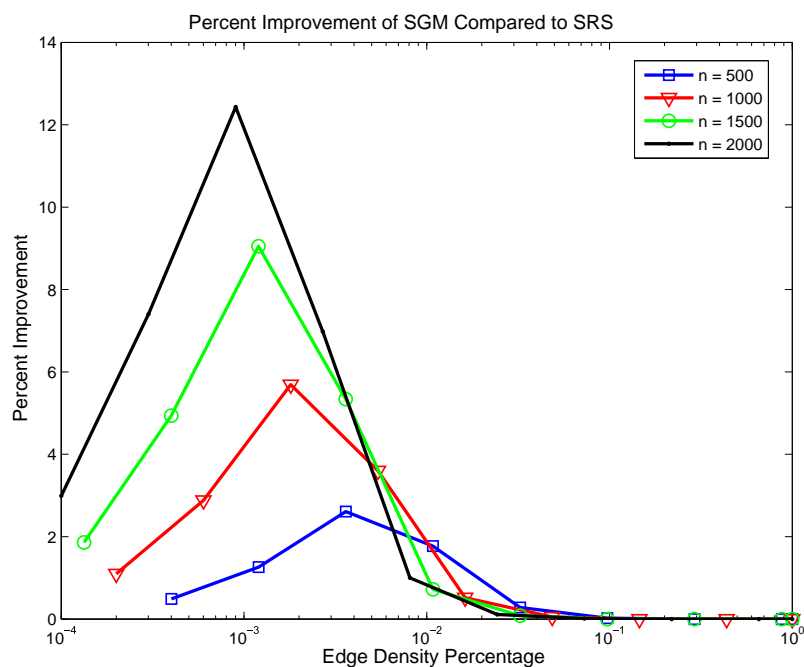
- *When do the algorithms tend to fail? Same policies?*
 - Policy metrics: n , e , and standard deviation of P



- A few policies where HORO successful, while others not
- A few policies where SRS successful, while others not

Percent Improvement

- Compare percent improvement of SGM with other algorithms
 - Considered large policies, each with Zipfs distribution



- SGM performs better with larger policies

Conclusions and Future Work

- Policy optimization is helpful for list-based firewalls
 - Reorganize rules to find first match with fewer compares
 - Model policy as a DAG, finding optimal order is \mathcal{NP} -hard
- Presented Sub-Graph Merging Algorithm (SGM)
 - Used heuristic $\overline{P}(G(r_i))$ to determine *best* rule
 - Will merge subgraphs, allows for more possibilities
 - Experimental results very good, but algorithm is not perfect
- Several areas of future work
 - Heuristic should consider other sub-graph structures
 - Better evaluate if the break cases are realistic
 - Merging and breaking rules

Sub-Graph Merging Algorithm

```
1 while( $Q \neq \phi$ )
2   set  $r_b$  to rule in  $Q$ , selected = false
3   for( $\forall r_j \in Q$  and  $r_j \neq r_b$ )
4     if( $(X[r_b]/C[r_b]) < (X[r_j]/C[r_j])$ )then
5       set  $r_b$  to  $r_j$ 
6     end
7   end
8   while(! selected)
9     if( $C[r_b] == 1$ )then
10      add  $r_b$  to  $S$  and remove  $r_b$  from  $Q$ 
11      set selected to true and  $r_{selected}$  to  $r_b$ 
12    else
13      bool temp = false
14      for( $i = 1; i < b; i++$ )
15        if( $DEP[r_i][r_b] == 1$ ) then
16          if(temp == false) then
17            set  $r_t$  to  $r_i$ 
18            set temp to true
19          else if( $(X[r_t]/C[r_t]) < (X[r_i]/C[r_i])$ )then
20            set  $r_t$  to  $r_i$ 
21          end
22        end
23      end
24      set  $r_b$  to  $r_t$ 
25    end
26  end
27  for ( $i = selected+1; i \leq n; i++$ )
28    if ( $DEP[r_{selected}][r_i] == 1$ )
29      set  $DEP[r_{selected}][r_i]$  to 0
30      decrement  $C[r_i]$  by 1
31       $X[r_i] = X[r_i] - PROB[r_{selected}]$ 
32    end
33  end
34 end
```