Firewall

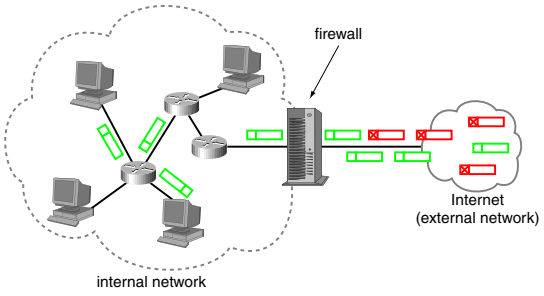
CSC 790

WAKE FOREST
UNIVERSITY
Department of Computer Science

Spring 2014

Network Firewalls

- A firewall is located between the Internet and internal network
  - Remains the forefront defense of most computer systems
  - Inspecting traffic, the firewall drops or accepts packets



  - As a result, firewall provides access control between networks
- The firewall applies a security policy to each arriving packet

Firewall Security Policy

- Security policy is an **ordered** list of rules
  - Rules consist of a 5-tuple: protocol type, IP source address, source port, IP destination address, and destination port
  - Fields can be fully specified or contain wildcards '*'

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action
1	TCP	140.*	*	*	80	accept
2	TCP	150.*	*	120.*	80	accept
3	TCP	140.*	*	130.*	20	accept
4	UDP	150.*	*	*	3030	accept
5	*	*	*	*	*	deny

  - Every rule has an action **accept** or **deny**
- Rules are applied to every packet, (starting with the first rule)
  - If a packet matches a rule, the associated action is performed

A Simple Model of Policies and Rules

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action
1	TCP	140.*	*	130.*	20	accept
2	TCP	140.*	*	*	80	accept
3	TCP	150.*	*	120.*	90	accept
4	UDP	150.*	*	*	3030	accept
5	*	*	*	*	*	deny

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

$$r = (r[1], r[2], ..., r[k])$$

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

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

  - Packets sequentially compared with rules until *first match*

Matching

- Using the models, can formally describe *processing packets*
- Recall a *first-match* policy typically takes place

  Definition Packet d matches r_i if

$$d \Rightarrow r_i \quad \text{iff} \quad d[l] \subseteq r_i[l], \quad l = 1, ..., k$$

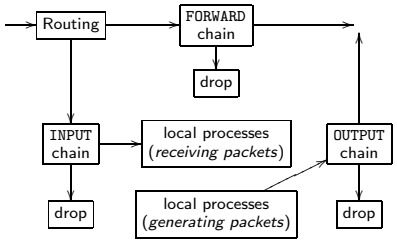
- Assume $d =$ (`TCP, 140.1.1.1, 90, 130.1.1.1, 20`)
  - Using the policy below, $d \Rightarrow r_1$ and $d \Rightarrow r_3$
  - Different actions, but this is **not** considered an anomaly

		Source		Destination		
No.	**Proto.**	**IP**	**Port**	**IP**	**Port**	**Action**
1	TCP	140.*	*	130.*	20	accept
2	TCP	140.*	*	*	80	accept
3	*	*	*	*	*	deny

- *Best match* is another policy, but difficult to manage policies

Linux Firewalls

- The Linux firewall mechanism is Netfilter (`iptables`)
  - Has multiple *rule chains*, where each has a policy
  - Chains include `INPUT` (local machine is destination), `OUTPUT` (local machine generated), and `FORWARD` (just passing through)
- The *general* packet flow using Netfilter is



  - Arriving packets pass through routing, which determines whether to send to `INPUT` or `FORWARD`

  - Locally destined packets are processed by the `INPUT` chain, if accepted it is sent to the receiving process
  - If the machine is allowed to forward and packet is remotely destined, then processed by the `FORWARD` chain
  - Locally generated packets are processed by the `OUTPUT` chain

    So what is a chain?

- It's actually more complicated than described
  1. *Prerouting* called for forwarding and locally destined (NAT)
  2. Appropriate chain is then called
  3. If forwarding, packet managling is called, allows QoS
  4. If generated locally generated, *postrouting* is called (NAT)
  5. Packet sent to the device or process

Netfilter Rules

- `iptable` rules allow more than the 5-tuple
  - Can specify the MAC address and ToS field

    Why? Can you always filter MAC of the source?

  - `iptables` can also provide rate limiting
- Some example `iptable` commands to add different rules

```
iptables -F INPUT
iptables -P INPUT DROP
iptables -A INPUT -i eth0 -s 127.0.0.0/8 -j DENY
iptables -A INPUT -p UDP --dport 2045 -j DROP
iptables -A INPUT -p UDP -s 0/0 -d 0/0 --dport 53 -j ACCEPT
iptables -vnL
```

Default accept or deny, which is better?

Firewalls and State

- The previous firewall examples are **packet filters/screens**
  - Process packets based on a *static* rule-set
  - No **state** information is stored
- Stateful firewalls maintain connection information
  - Assume a connection is established from the internal network
  - Firewall keeps track of this connection
  - Packets that arrive from the external network must be part of an **existing** connection
- Example state rule in `iptables`

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

What is the disadvantage of maintaining state?

Firewalls and Payload Inspection

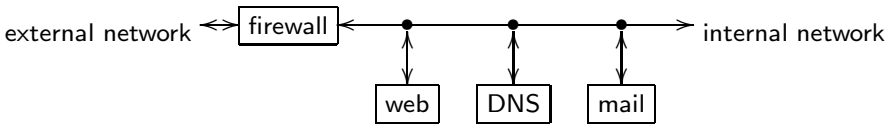
- The previous firewall examples only inspect the header
  - It is possible to some simple payload inspections
  - Can inspect HTTP for URLs
  - More complex payload inspections should be done by IDS
- Example string match rule in `iptables`

```
iptables -m string -help
iptables -A INPUT -p tcp -m string --algo bm --string "exe" -j DROP
iptables -A INPUT -p tcp -m length --length 10:100 -j LOG
```

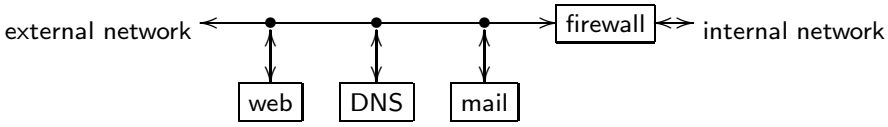
- Replace the string with whatever the new Windows "worm-of-the-moment" is requesting and just add another rule.

Firewall Topologies

- Consider a single firewall between internal and external networks
  - Assume there are web, DNS, and mail servers

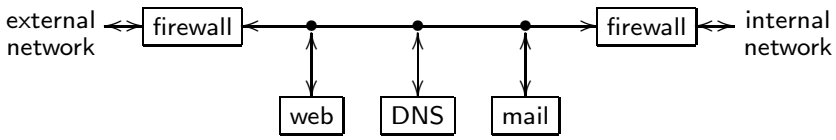


  - Simple implementation but single point of failure
- Place servers outside of the internal network



Advantages and disadvantages?

- Common to use multiple firewalls
  - Create a demilitarize zone (DMZ), containing Internet servers
    "The DMZ is a network that separates an external network from a purely internal network"



  - Firewall separates internal network from DMZ and Internet
  - Protects against compromised first firewall or DMZ
  - Can use multiple DMZ and build *enclaves*
    Advantages and disadvantages?

- *Can use personal firewalls at every machine...*

Proxy

- A firewall mediates access to a network
  - Allowing and disallowing certain type of access
- A proxy is an intermediate agent or server
  - Acts on behalf of endpoints without allowing direct connection
- A proxy firewall uses proxies to perform access control
  - Direct connections are not allowed between endpoints
  - Bases access control on content of the packets
  - Also called a *bastion host*

Other Firewall Issues

- Multiple connections per application
  - Applications may one connection for data, another for control
  - Firewall may know one connection, must be aware of the other

    For example?
- Dynamic connections per application
  - We know that *"IP addresses and TCP ports play an essential role in the binding a client and a remote target object"*
  - Applications (CORBA) may dynamically create connections
  - CORBA needs location transparency and P2P communication

- Large range of port numbers possible *that firewall must allow*

  "A packet filter firewall located between client and server is likely to be configured in a way that remote invocations are blocked. The application will be unable to complete the request. To enable all remote invocations on objects behind the packet filter firewall, the firewall would have to be opened for connections to all hosts running CORBA servers. A broad range of port numbers would have to opened on the firewall: any port a CORBA server could be listening on which is potentially any non-privileged port."

  So what? What aboout NAT?

- *As a result firewalls may need to become application aware*

  So what? Isn't this what IDS does...

Improving Firewall Performance

- Firewalls remain the first defense for most systems
  - Applying policy to all arriving packets
  - Must manage increasing volumes of packets, increasing number of rules, and strict **latency requirements**
- Two general methods for improving performance
  - Reorganize the rule list
  - Improve rule search algorithm
  - Perform search in parallel
- In all the methods must **maintain 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**

		Source		Destination			
No.	Proto.	IP	Port	IP	Port	Action	Prob.
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

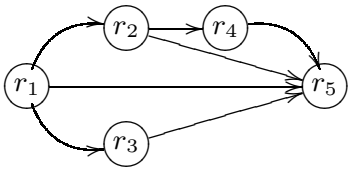
  Definition The intersection of rule r_i and r_j, denoted as $r_i \cap r_j$ is

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

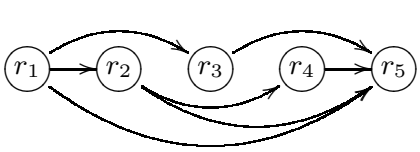
- *Why is the intersection appropriate?*
  - Intersection describes the set of packets that match both rules
  - If rules intersect, then order is significant **for maintaining integrity**

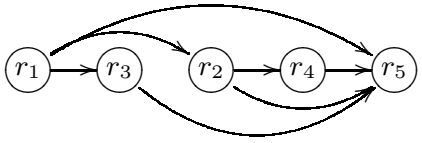
Modeling Precedence

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action	Prob.
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 $\{p_1, ..., p_n\}$
  - 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

Simple Rule Sort

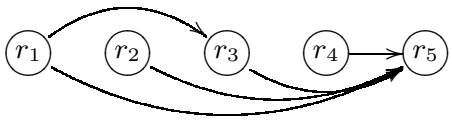
- Swap out-of-order rules if no precedence edge exists between them
  - Repeat until all rules are in order...

```
1 done = false
2 while (!done)
3     done = true
4     for (i = 1; i < n; i++)
5         if (p_i < p_{i+1} AND r_i ⊄ r_{i+1})
6             interchange rules r_i and r_{i+1} and probabilities p_i and p_{i+1}
7             done = false
8             end
9     end
10 end
```

Any problems with the above algorithm? Will it maintain integrity?

Example SRS Ordering

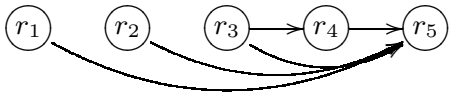
No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action	Prob.
1	UDP	170.1.*	*	*	90	accept	0.05
2	UDP	180.1.1.*	*	*	90-94	deny	0.15
3	UDP	170.1.1.*	*	*	*	deny	0.05
4	UDP	190.1.1.2	*	*	90	accept	0.25
5	*	*	*	*	*	deny	0.5



- First pass of SRS
  1. Compare r_1 and r_2 and swap
  2. Compare r_1 and r_3
  3. Compare r_3 and r_4 and swap
  4. Compare r_4 and r_5
- Final SRS ordering is $\{r_4, r_2, r_1, r_3, r_5\}$, $E[n] = 3.4$ which is optimal

SRS does not always work...

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action	Prob.
1	UDP	170.1.*	*	*	90	accept	0.05
2	UDP	180.1.1.*	*	*	90-94	deny	0.15
3	UDP	190.1.1.*	*	*	80	deny	0.05
4	UDP	190.1.1.2	*	*	*	accept	0.25
5	*	*	*	*	*	deny	0.5

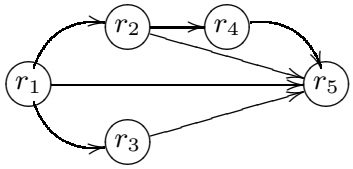


- First pass of SRS
  1. Compare r_1 and r_2 and swap
  2. Compare r_1 and r_3
  3. Compare r_3 and r_4
  4. Compare r_4 and r_5
- Final SRS ordering is $\{r_2, r_1, r_3, r_4, r_5\}$, $E[n] = 3.9$
- Optimal is $\{r_3, r_4, r_2, r_1, r_5\}$, $E[n] = 3.7$

  What went wrong?

More Info, More Better

- Better algorithms 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 $\overline{P}(G(r_i))$ 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 ≠ ∅)
2    set r_b to a rule in Q
3    for(∀r_j ∈ Q and r_j ≠ r_b)
4       if(P̄(G(r_b)) < P̄(G(r_j)))then
5          r_b ←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(∀r_i ∈ G*(r_b))
15            if(P̄(G(r_t)) < P̄(G(r_i)))then
16               r_t ←r_i
17            end
18         end
19      end
20      r_b ←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

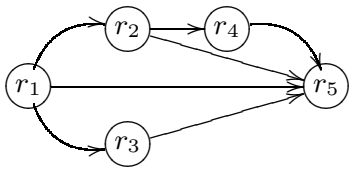
- Does not have to order an entire sub-graph per iteration (*line 8*)
  - This design allows merging of sub-graphs (*but at a cost*)

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 IP	Source Port	Destination IP	Destination Port	Action	Prob.
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

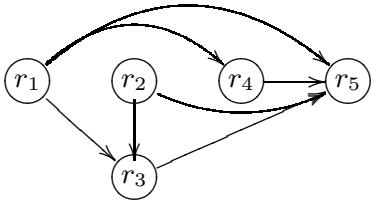


1. r_5 is best, but $G^*(r_5) = \{r_1, r_2, r_3, r_4\}$, select from this sub-graph
2. r_4 is best, but $G^*(r_4) = \{r_1, r_2\}$, select from this sub-sub-graph
3. r_2 is best, but $G^*(r_2) = \{r_1\}$, select from this sub-sub-sub-graph
4. r_1 is best and $G^*(r_1) = \{\emptyset\}$, so placed in S

- 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 SGM does not always work...

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action	Prob.
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*

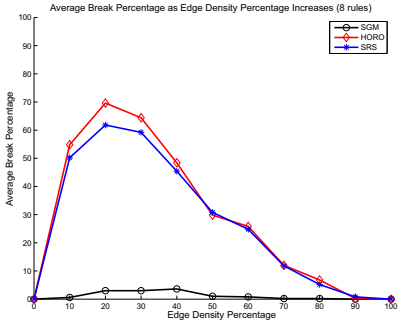
- Final SGM order is $\{r_2, r_1, r_3, r_4, r_5\}$, optimal is $\{r_1, r_4, r_2, r_3, r_5\}$

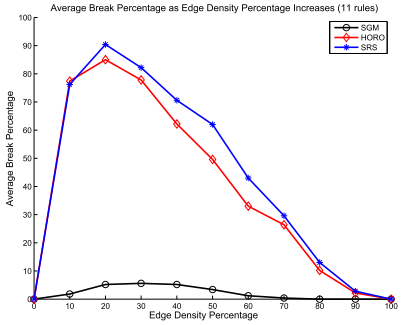
Experiments

- 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

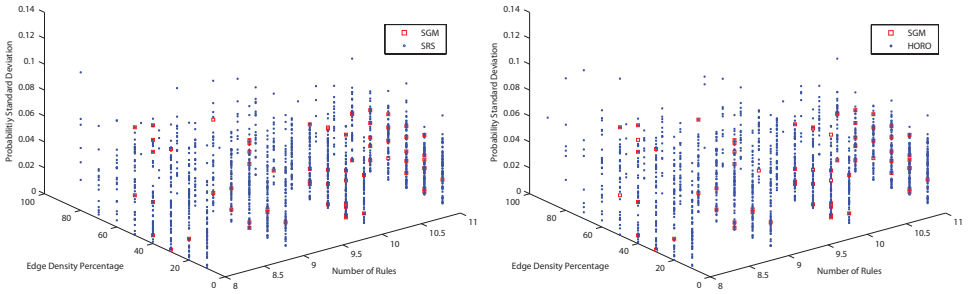




- SGM performs best, all have problems with 10 - 40% edge density

Break Case Comparison

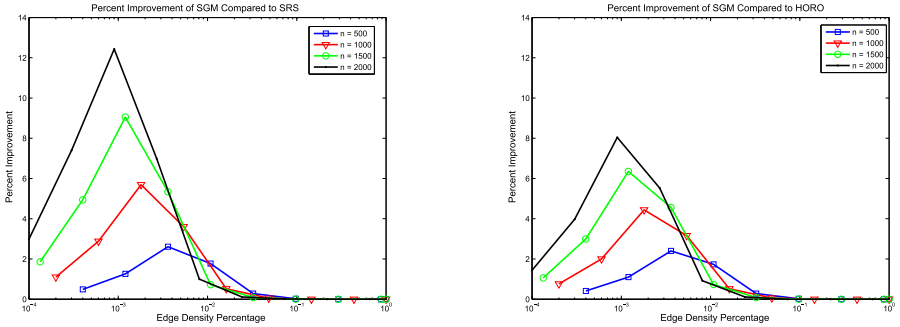
- *When do the algorithms tend to fail? Same policies?*
  - Policy metrics: n, e, and standard deviation of P



- 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

Optimization Summary

- 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

    Why only list based? Are there alternatives?

- 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
- Optimization focuses on the average case, *what about worst case?*

Alternatives to First Match

- Reordering rules can improve best and average case performance
  - Applicable to any list base firewall
  - However, worst case performance is not better

    How likely is the worst case?

- Best approach for searching is binary search
  - Consider a policy with no intersecting rules

    Is such a policy possible?

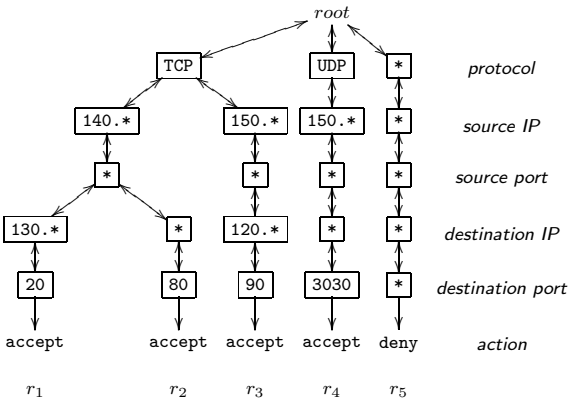
  - Sort the rules, then apply binary search to find the correct match

    How do you sort the rules? What is the key given n-tuples?

Trie-Based Policy Representation

- Use an *n-ary retrieval tree* to represent the policy
  - **Policy trie** groups rules based on common tuples
  - Reduces the number of compares and storage
- Policy trie consists of k levels that stores the policy
  - Each level stores a rule tuple
  - Nodes at a level store tuple values (from the rules)
  - Nodes always ordered from specific to general (left to right)
  - Search occurs from left to right
  - A non-cyclical path from root to a leaf is a rule

Policy Trie

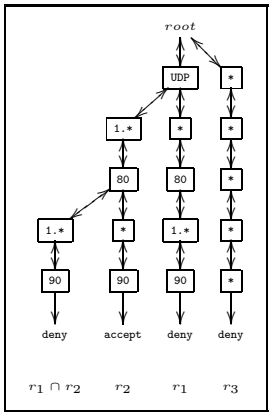


No.	Proto.	SIP	SPort	DIP	DPort	Action
1	TCP	140.*	*	130.*	20	accept
2	TCP	140.*	*	*	80	accept
3	TCP	150.*	*	120.*	90	accept
4	UDP	150.*	*	*	3030	accept
5	*	*	*	*	*	deny

Maintaining Integrity

- *Order of the list and trie may not match*

No.	Proto.	Source IP	Source Port	Destination IP	Destination Port	Action
1	UDP	*	80	1.*	90	deny
2	UDP	1.*	80	*	90	accept
3	*	*	*	*	*	deny



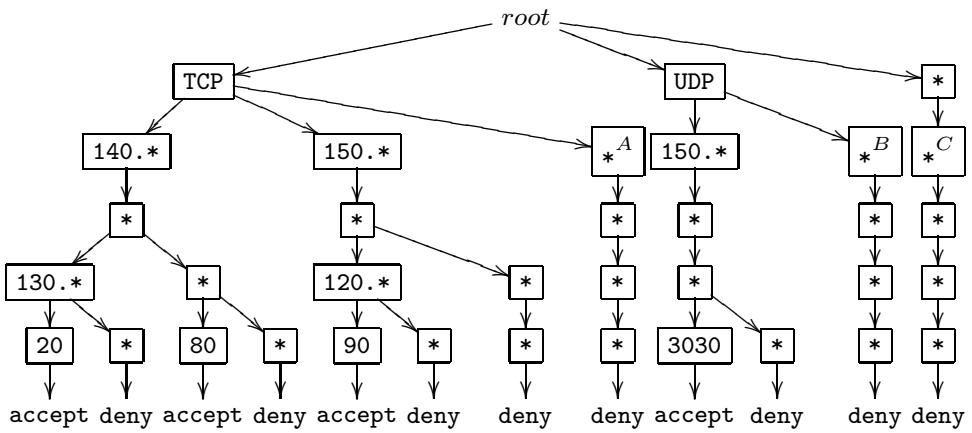
- One important constraint is integrity
  - Original list and policy trie must produce same result
- If a rule is located to the left of existing rules (due to tuple)
  - Take intersection of the new rule and rightward existing rules

Integrity and Performance

- Policy trie maintains integrity of R

  Theorem 0.1 *Policy trie is equivalent to original security policy.*
- Performance (number of tuple comparisons) is better than R
  - Multiple rules can be eliminated in one comparison
  - However processing a packet may require backtracking (searching for a more *general* rule)
- Non-backtracking trie or **push down trie** (PDT)
  - Created by *pushing-down* all general rules
  - Replicating general rule (contains wild card) towards the left

Push-Down Policy Trie



r_1 $r_8 = r_7 \downarrow r_1$ r_2 $r_7 = r_6 \downarrow r_1$ r_3 $r_{10} = r_9 \downarrow r_3$ $r_9 = r_6 \downarrow r_1$ $r_6 = r_5 \downarrow r_1$ r_4 $r_{12} = r_{11} \downarrow r_4$ $r_{11} = r_5 \downarrow r_4$ r_5

Integrity and Performance

- PDT maintains integrity

  Theorem 0.2 *A push-down policy trie T_p is equivalent to the original policy trie T, which is equivalent to the original security policy R.*

- Worst case number of tuple-comparisons for PDT is $O(n + k)$

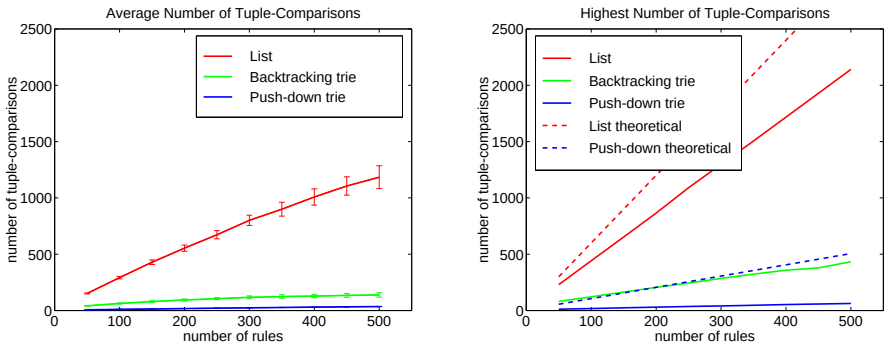
  Theorem 0.3 *A comprehensive push-down trie consisting of k levels and constructed from n rules requires $O(n + k)$ number of tuple-comparisons to match a packet in the worst case.*

  – Only $1/k$ of a list-based representation

- PDT does require more storage, but worst case is $O(n^2)$

Policy Trie Experimental Results

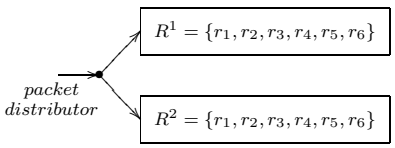
- Policy trie variations compared with list-based representation
  - Randomly generated policies, 50 to 500 rules
  - Processed 10,000 packets per simulation



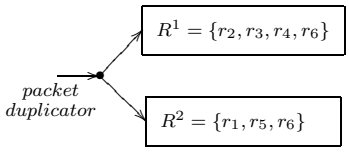
- Policy tries were significantly better, within theoretical bounds

Parallel Firewall Designs

- Parallel firewalls can be used to reduce packet processing time
- Given an array of m machines, consider two parallel designs
  - **Data parallel**, distribute packets
  - **Function parallel**, distribute rules *with restrictions*



Data parallel

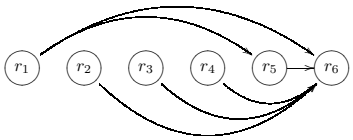


Function parallel

- Make certain **integrity** is maintained for each design

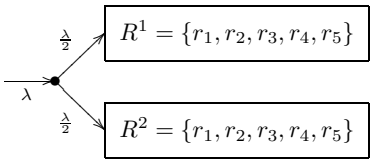
Policy Attributes

- Policy rules may intersect (*a packet can match more than one rule*)



- Every policy R has the following sets
  - Accept set A, deny set D, and no-match set N
- A policy is **comprehensive** if $A \cup D = P$ or $N = \emptyset$
- Consider two policies R and R'
  - Policies **equivalent** if comprehensive and $A = A'$
  - **Integrity** is maintained if R is replaced with R'
  - *Integrity applies to policies and implementation*

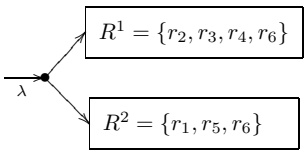
Data Parallel



- Multiple firewalls connected in parallel
  - Packets distributed across the firewalls ($\frac{\lambda}{m}$), where λ is the arrival rate and m is the number of firewalls
  - Each firewall implements $R^i = R, \quad i = 1, ..., m$
- Maintains integrity since $A^i = A, \quad i = 1, ..., m$
- Scales (does not depend on rule set) and robust
- Difficult to provide service differentiation and state information

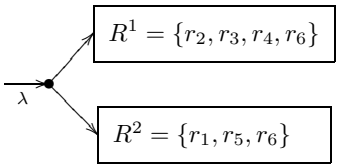
Function Parallel

- An array of m *independent* firewalls
  - Rules are distributed and packets are duplicated



  - A packet is processed by each firewall in parallel
- Has several advantages over data parallel design
  - Scalable and maintains state information
  - Can be faster under low traffic load, *given proper rule distribution*

Function Parallel Firewall Policy Integrity



- Integrity maintained if rules distributed using accept sets

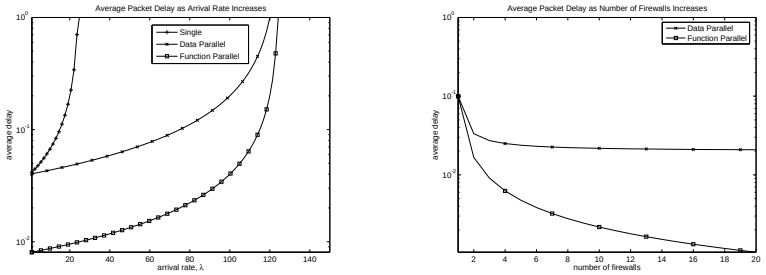
Theorem 0.4 *A function parallel array of m firewalls enforcing a comprehensive policy R maintains integrity if policy rules are distributed such that: each local policy is comprehensive, $\bigcup_{j=1}^{m} A^j = A$, and $\bigcap_{j=1}^{m} A^j = \emptyset$.*

  - Only one firewall will accept a packet, the remaining will deny

Several possible distributions may exist, which is best?

Parallel Firewall Performance

- A simple theoretical analysis is possible using queueing theory
  - System with n firewalls model as n M/M/1 queues
  - Data parallel divides arrivals while function parallel divides work
- Theoretical performance of the different systems

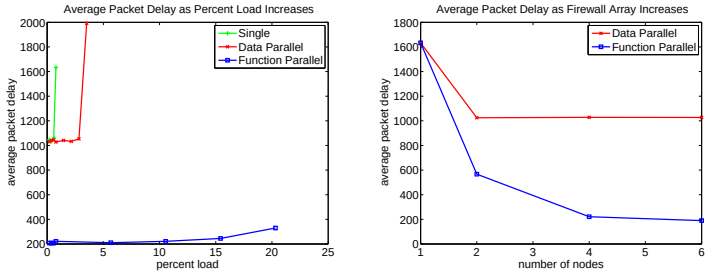


  - Function parallel is n times faster...

What are the limitations of the models?

Results Using BSD Firewalls

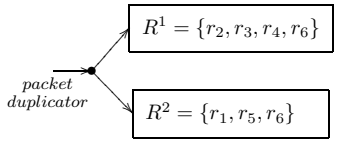
- BSD firewalls, policy size 4096 rules, packet size 512 bytes
  - Traffic generated by SmartBits 200B



- Experimental results are very similar to simulation

Policy Distribution Performance

- Consider function parallel array of m firewalls and a single packet
  - At time t_0 packet compared against m rules simultaneously
  - Probability of first-match at t_0 is the sum of the probabilities of the first rule (p_1^j) in each firewall



- Number of comparisons to find the *original* first-match

$$1 \cdot (p_1^1 + p_1^2) + 2 \cdot (p_2^1 + p_2^2) + 3 \cdot (p_3^1 + p_3^2) =$$
$$1 \cdot (p_2 + p_1) + 2 \cdot (p_3 + p_5) + 3 \cdot (p_4 + p_6) = 2.35$$

- Want to minimize the above summation, *how can it be done?*