Bypassing Space Explosion in Regular Expression Matching for Network Intrusion Detection and Prevention Systems

Jignesh Patel, Alex Liu and Eric Torng

Dept. of Computer Science and Engineering
Michigan State University

Problem Statement

- Core operation in IDS/IPS is Deep Packet Inspection
  - Past DPI: string matching
  - Current DPI: regular expression (RE) matching
    - Example: SNORT, Bro

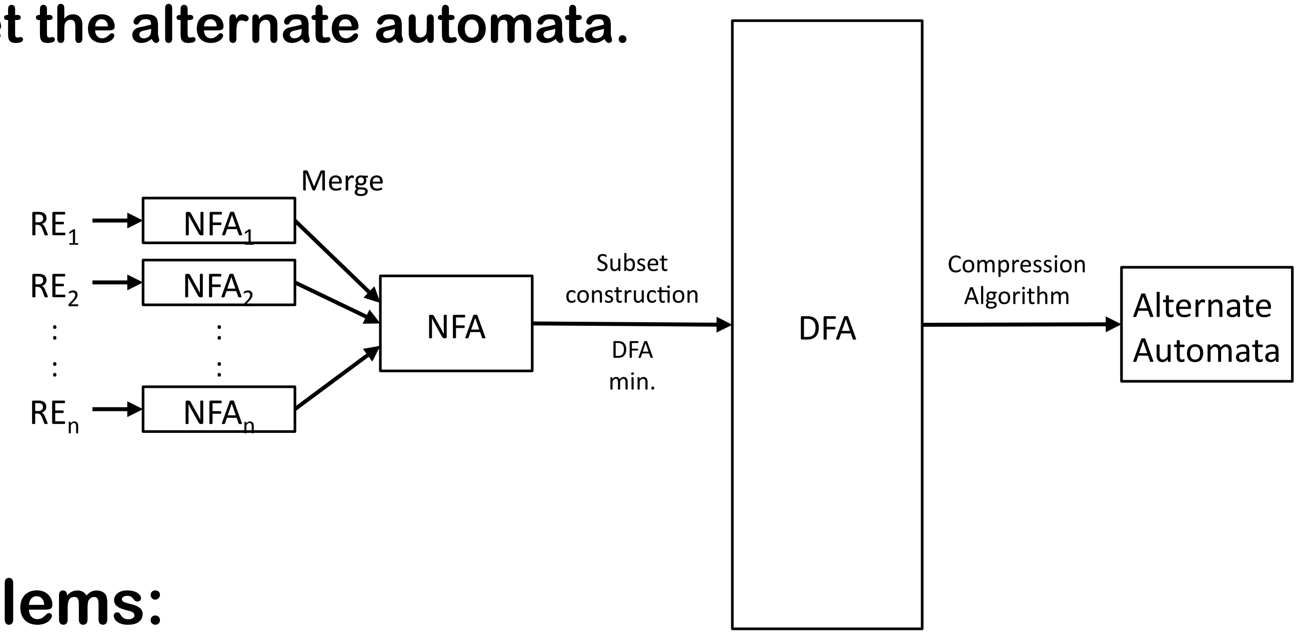
- Problem: given a set of REs, how to quickly scan packet payload to determine which REs are matched?

Solution using Automata

- **Common solution is to build an equivalent Finite State Automata based on DFA.**
- **DFA size grows exponentially with number of REs.**
- **Several alternate automata have been proposed D^2FA, XFA, δFA etc.**

Limitations of Prior Work

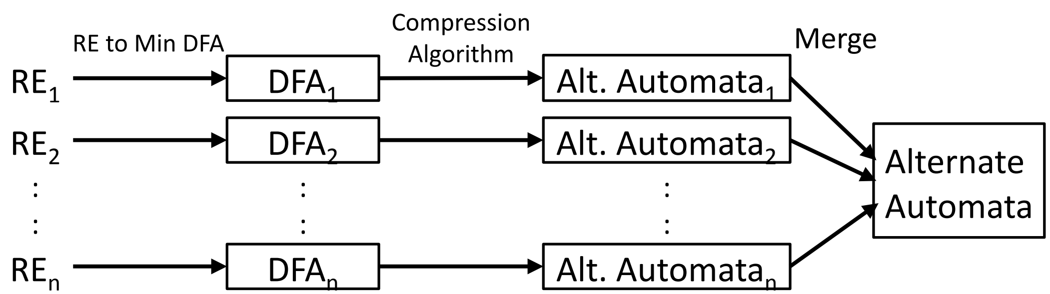
- Prior solution: **Union then Minimize** framework.
  - First combined DFA for the whole RE set is built.
  - Compression technique is applied to the combined DFA to get the alternate automata.



- Problems:
  - The minimization/compression is applied on large combined automata, hence requires too much time and memory.
  - The intermediate DFA might be too large to fit in memory.
  - Whole automata needs to be rebuilt if new RE is added to set.

Our Approach

- Our approach: **Minimize then Union** framework.
  - Build individual DFAs for each RE in the RE set.
  - Compress each DFA to get individual alternate automata.
  - Merge the all compressed alternate automata together.

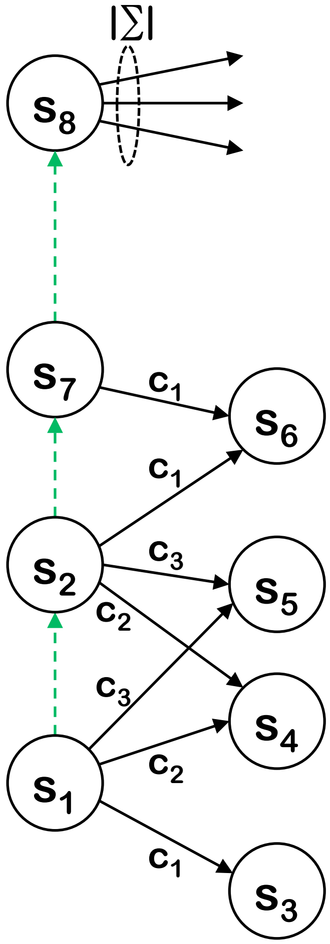


- Advantages
  - The compression algorithm is applied to small DFAs.
  - Large intermediate DFA does not need to be built.
  - Easy to add new RE to the set with one merge.
- In this work we focus on the D^2FA.

D^2FA Overview

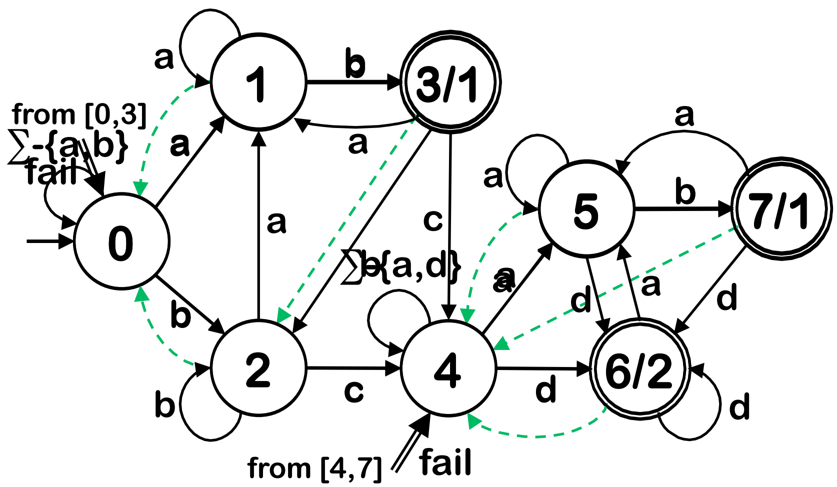
- D^2FA [Kumar et al., 2006] uses common transitions between states to reduce the number of transitions.

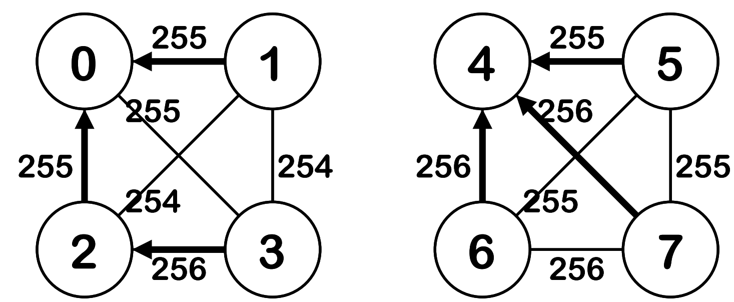
- To build a D^2FA:
  1. We choose a deferred state for each state in the DFA.
  2. For each state, remove transitions that are common with its deferred state.



D²FA Construction

- **Build Space Reduction Graph (SRG)**
- **Find maximum spanning tree (MST) in SRG.**
- **Use the MST to set deferred states.**





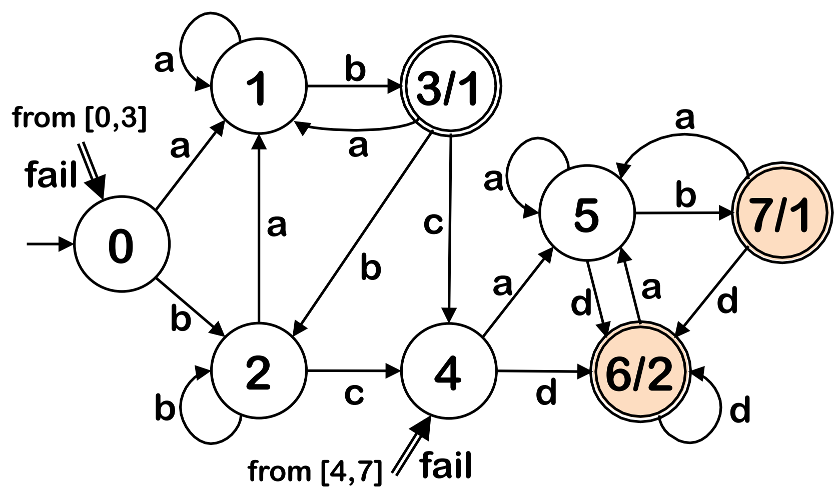
SRG

D²FA for RE set {ab, bc.*d}

2048 Transitions

DFA for RE Matching in DPI

- **Traditional DFA defined as**
$$(Q, \Sigma, \delta, q_0, A),$$
where $A \subseteq Q$ is the set of accepting states.
- **For RE matching in DPI, we redefine DFA as**
$$(Q, \Sigma, \delta, q_0, M),$$
where $M: Q \rightarrow 2^R$ gives, for each state, the subset of REs matched from RE set R.

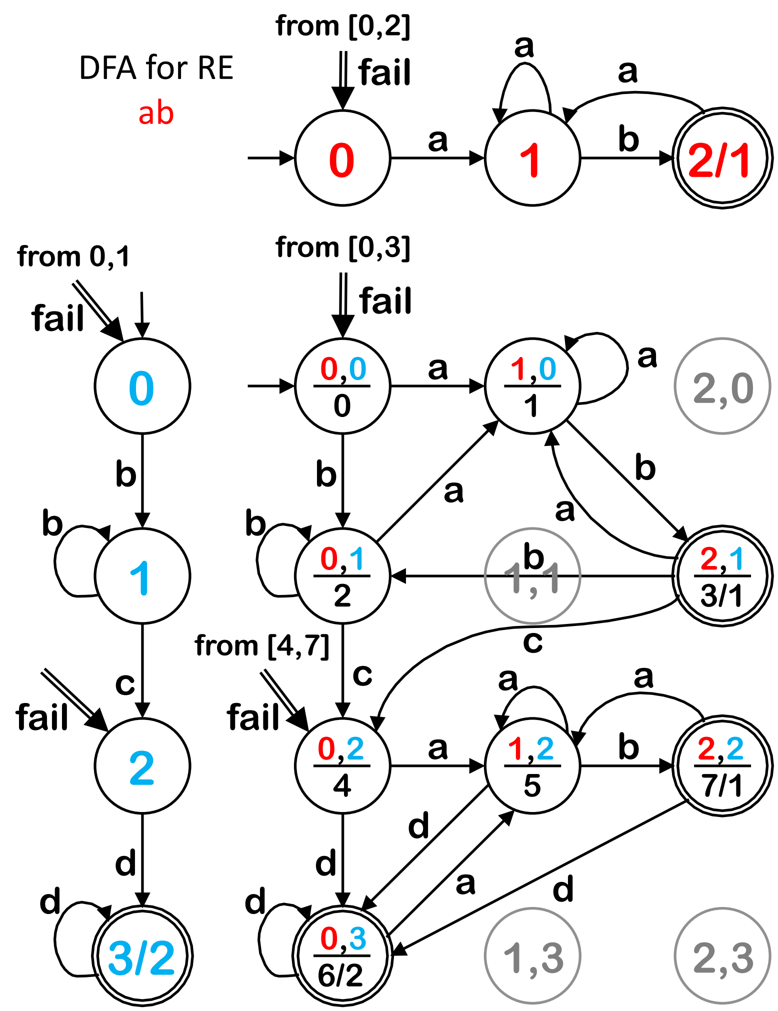


DFA for RE set {ab, bc.*d}

Merging DFAs (1)

- **Input: Min. state DFAs D_1 and D_2 equivalent to RE sets R_1 and R_2.**
- **Output: Min. state DFA D_3 equivalent to RE set $R_1 \cup R_2$.**

- **Solution: Use the standard Union Cross Product (UCP) construction, $D_3 = UCP(D_1, D_2)$**
- **Each state in D_3 corresponds to a pair of states in D_1 and D_2. $Q3 = Q1 \times Q2$.**



DFA for RE bc.*d

Merged DFA for RE set {ab, bc.*d}

Merging DFAs (2)

- For traditional DFA, $D_3 = UCP(D_1, D_2)$ is not guaranteed to be minimum state.
- We prove that for redefined DFA for DPI, D_3 is guaranteed to be minimum state if:
  - Only reachable state pairs are generated, and
  - $R_1 \cap R_2 = \emptyset$.
- To create the DFA for the entire RE set:
  - First create DFA for each RE
  - Merge DFAs together in a binary fashion to get the final DFA.
- Merge method much faster than direct method
  - Time to build largest DFA in our experiments:
    - Direct method: 386 seconds
    - Merge method: 0.66 seconds.

Merging D^2FA

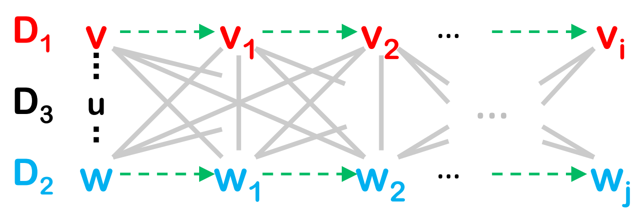
- We extend the UCP construction for merging DFAs to merge D^2FAs.
- To generate D^2FA, we need to set deferred state for each state.

- Set the deferred state as soon as new state is created.
- Since deferred state is set when a state is created, we only need to store the non-deferred transitions for the state.
- The whole DFA is never built since we always store the D^2FA.

Setting Deferred State

- Idea: Use deferment relation from the input D^2FAs to set the deferment in the merged D^2FA.

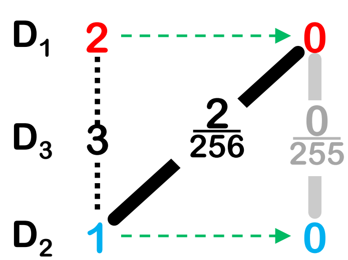
- To choose deferred state for new state, $u = \langle v, w \rangle$, in D_3, we use deferment of v in D_1 and w in D_2.

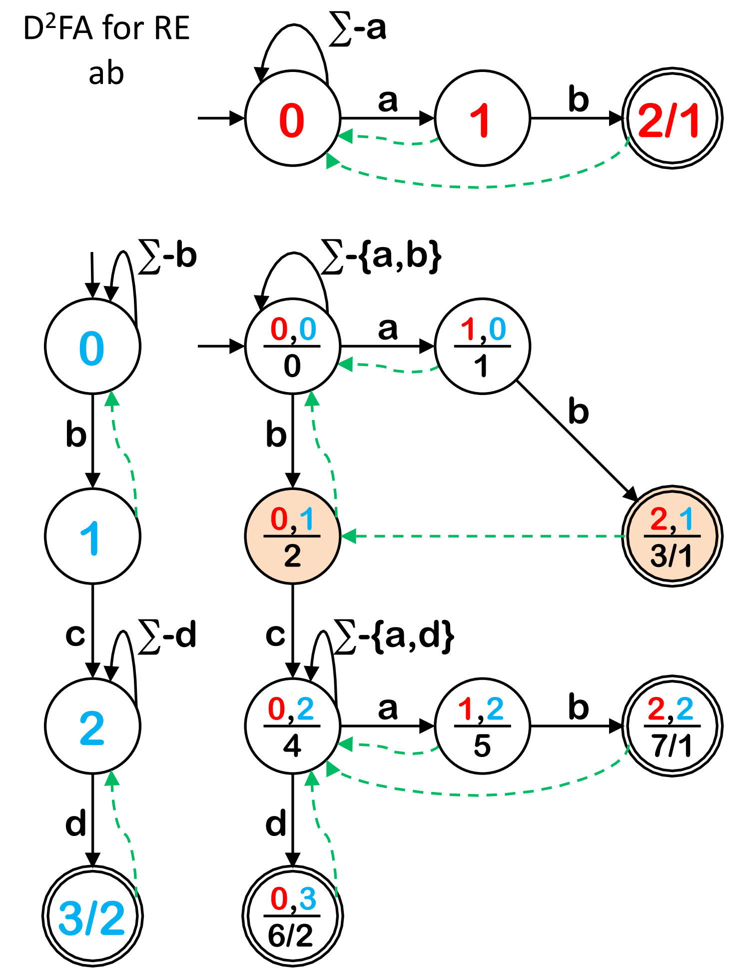


- Among all the (i+1)x(j+1)-1 possible state pairs, choose the one which has most common transitions with $\langle v, w \rangle$.

Merging D^2FA Example

- For most states, one of the first pair is the best pair.
- In our experiments, average number of comparisons needed < 1.5





D^2FA for RE bc.*d

Merged D^2FA for RE set {ab, bc.*d}

Experimental Results: Main

- We used real world 8 RE sets that were used in prior work for our experiments.
- We group the 8 RE sets into three groups according to type of REs in the sets: STRING, WILDCARD, SNORT
- We compare D^2FA Merge algorithm with the Original D^2FA algorithm.

RE set group	# States / ASCII len.	Trans increase	Def. depth ratio		Space ratio	Speedup factor
			Avg.	Max.		
All	17.7	20.10%	7.3	4.8	1390	301.6
STRING	0.7	44.00%	1.8	1.6	2672.8	99.5
WILDCARD	36	3.00%	12	8.2	42.7	338.2
SNORT	10.7	21.30%	6.3	3.6	1882.1	399.7

Experimental Results: Scale

- **To test scalability we use a synthetic RE set with REs of the form** `/`$c_1c_2c_3c_4$`.*`$c_5c_6c_7c_8$`/`
- **We add one RE at a time until memory estimate goes over 1GB.**

- **Original D²FA algorithm:**
  - **# REs added: 12**
  - **# states in final D²FA: 397,312**
  - **Time to build D²FA: 71 hours**

- **D²FA Merge algorithm:**
  - **# REs added: 19**
  - **# states in final D²FA: 80,216,064**
  - **Time to build D²FA: 1.2 hours**

- **For 12 REs, D²FA Merge only needs 10 seconds to build.**
- **D²FA Merge results in same D²FA size as the original algorithm.**

Questions?

- **Thank you for listening!**