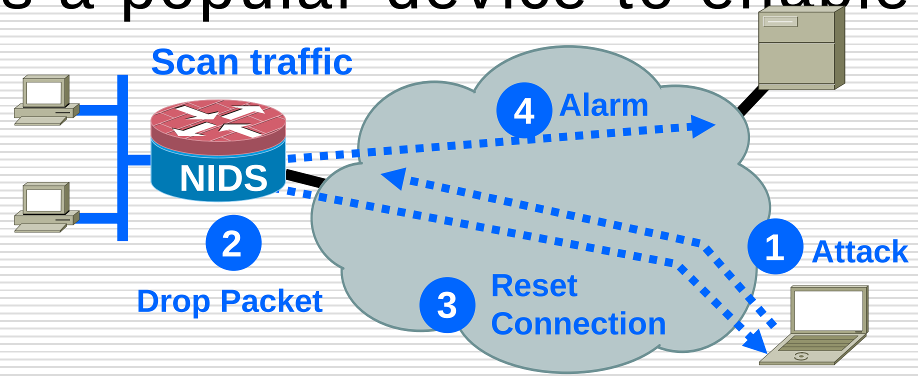
Curing Regular Expressions Matching Algorithms from Insomnia, Amnesia, and Acalculia

Sailesh Kumar
Balakrishnan Chandrasekaran
Jonathan Turner
George Varghese

Regular Expressions in Security

- Signature based NIDS is a popular device to enable network security.



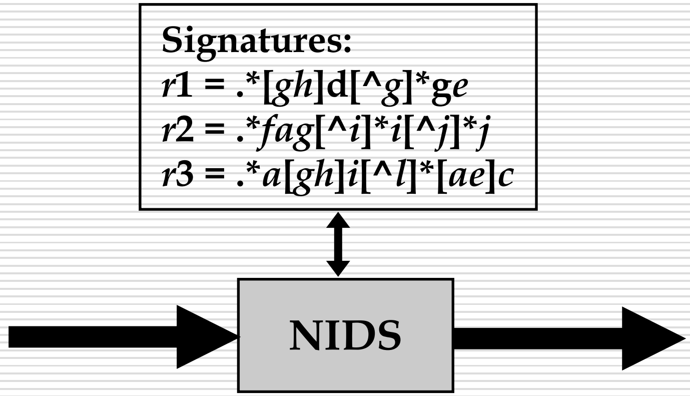
- Attack patterns are specified as regular expressions.
  - » `[ \t]*[Cc][Ww][Dd][ \t]+[~]root`
    - –Represents an attempt to change working directory to root.
- Regular expression matching is expensive.
  - » Thousands of signatures.
  - » High speed implementation requires GB memory. (often impractical.)

Traditional Implementation

- NIDS implementation.

Traditional implementation attempts to match traffic with the entire Virus signature

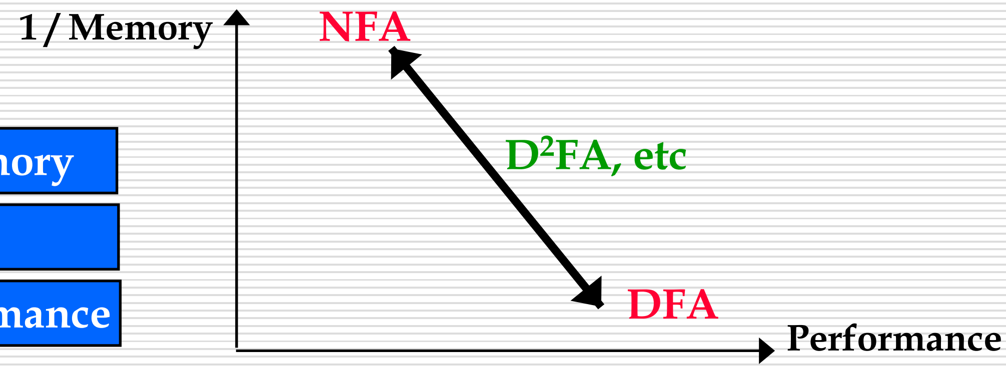
Complex signatures lead to trade-off



DFA: Fast, but requires large memory

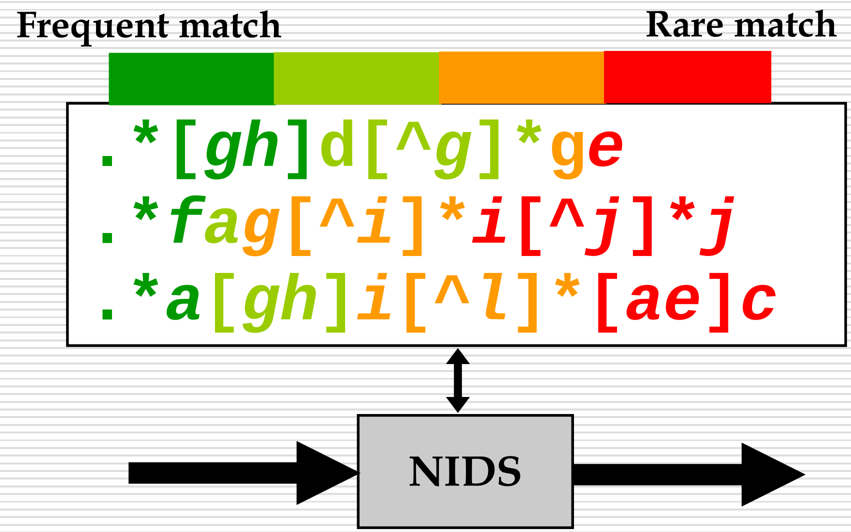
NFA: Compact, but slow

D²FA: Trades-off memory-performance



Insomnia

- NIDS implementation.



OBSERVATION:

Typical traffic rarely match first few symbols within any virus signature.

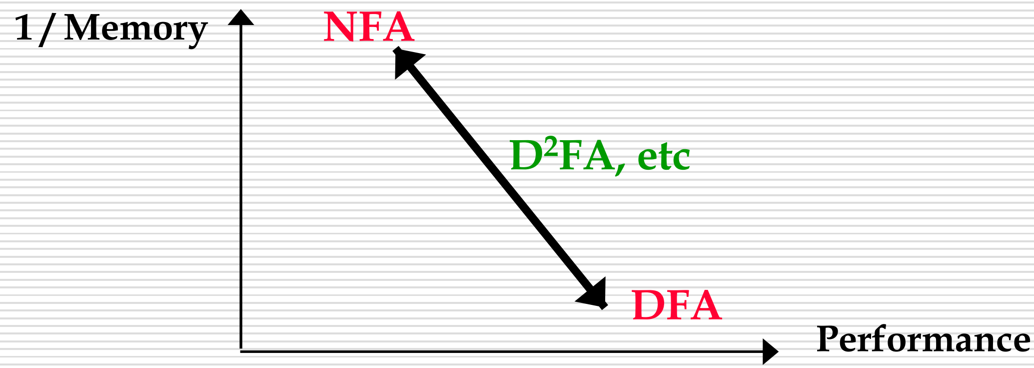
NIDS keeps the entire signature active.

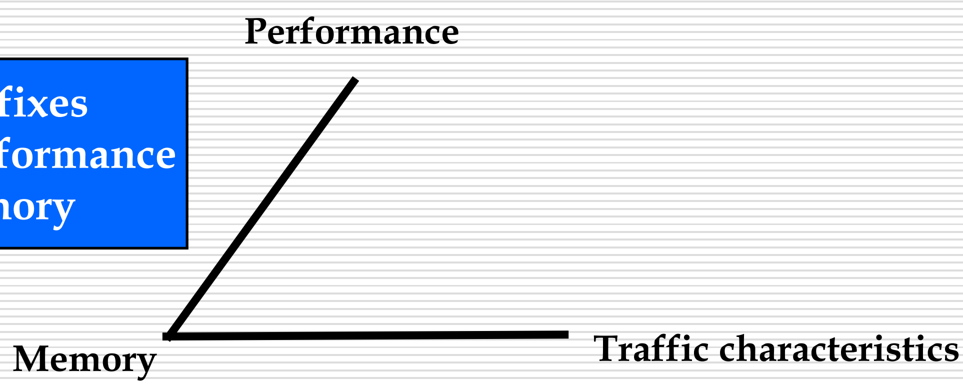
(Unvisited tail portions can be kept to sleep)

We refer to this problem as Insomnia

Cure to Insomnia

- Solve Insomnia with a three-way trade-off.





Smaller matching signature prefixes
=> high performance
low memory

In practice, frequently matching prefixes are very small in length

Cure to Insomnia

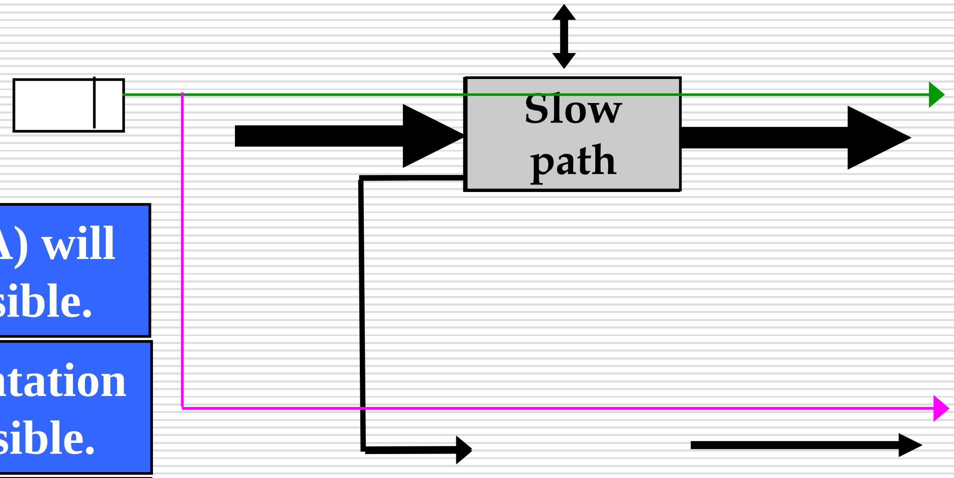
- Insomnia cure.
- If we se
  - » Prefixe
  - » Few packets match them – goto slow path

Only prefixes of signatures are matched in fast path

Frequent match — Rare match

```
.*[gh]d[^g]*ge
.*fag[^i]*i[^j]*j
.*a[gh]i[^l]*[ae]c
```

Packets that don't match prefix



Fast prefix implementation (e.g. DFA) will require less memory, and will be feasible.

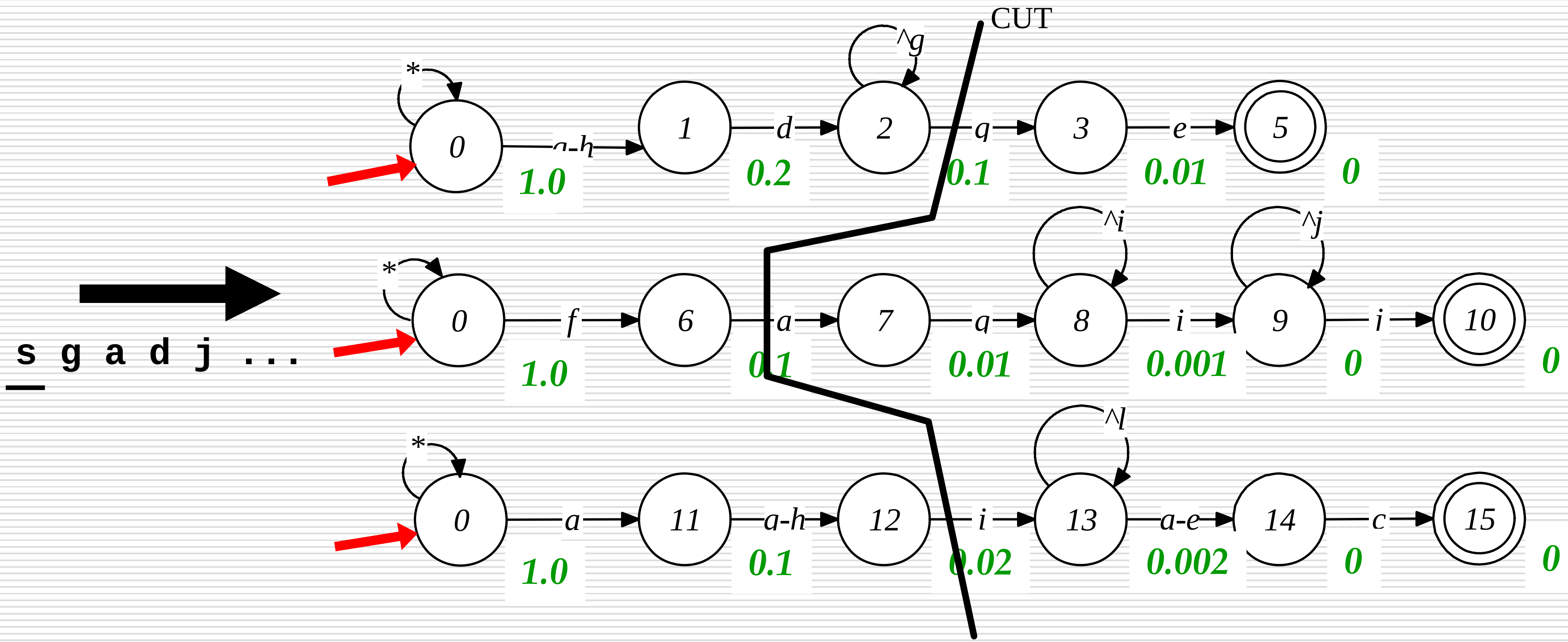
Suffixes won't require fast implementation will use less memory, and will be feasible.

High performance, Less memory

Suffixes of the prefi… matching signa… matched in slow p…

How to select the prefixes?

Prefix Generation

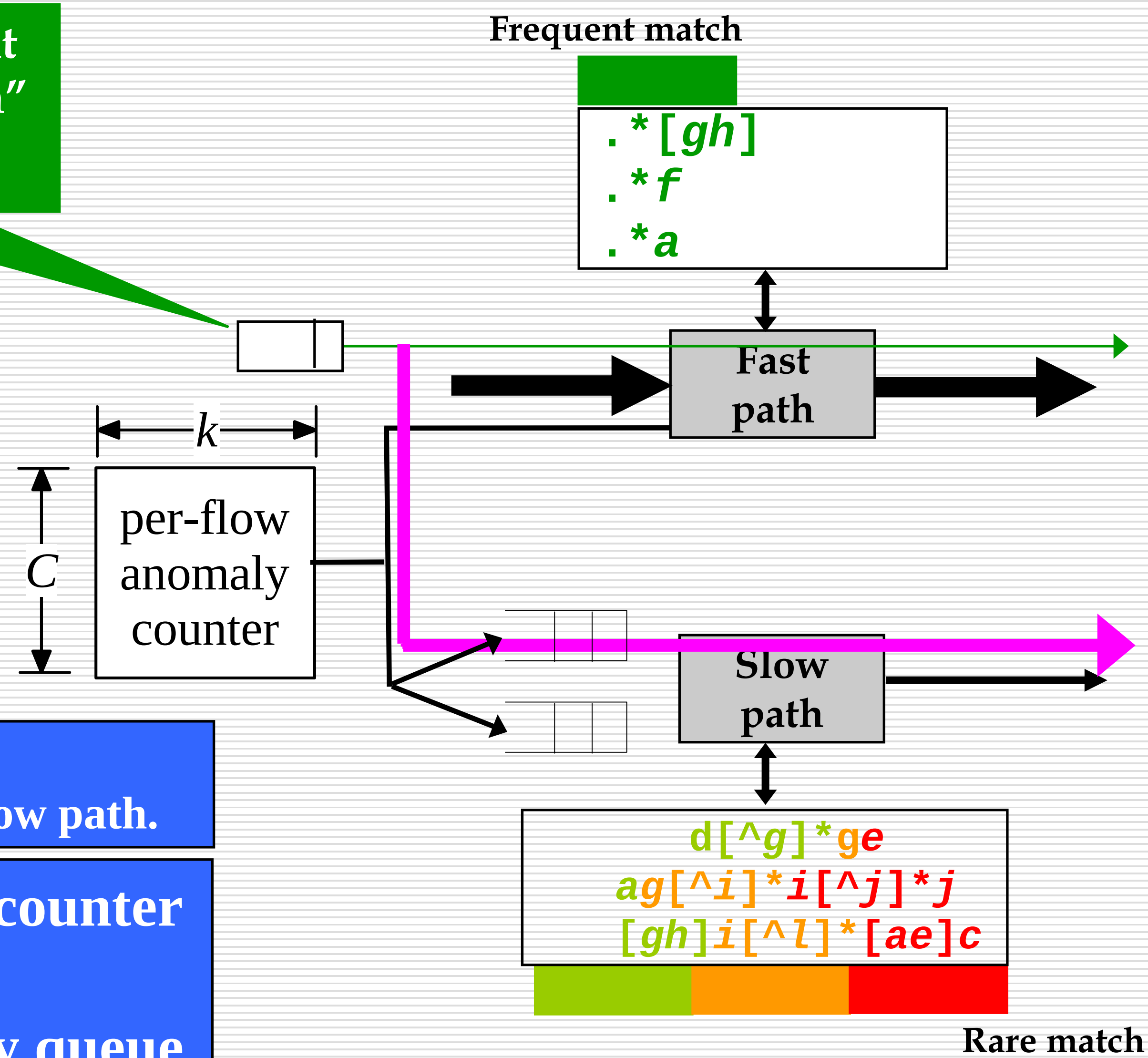


Construct the NFA → **Run NFA for an input trace** → **Count # times state is active** → **Find probability of state activity** → **MAKE A CUT (Limit the total slow path state probability)**

DoS Attacks

Attacker sends traffic that matches prefix “too often”
Overloads the slow path

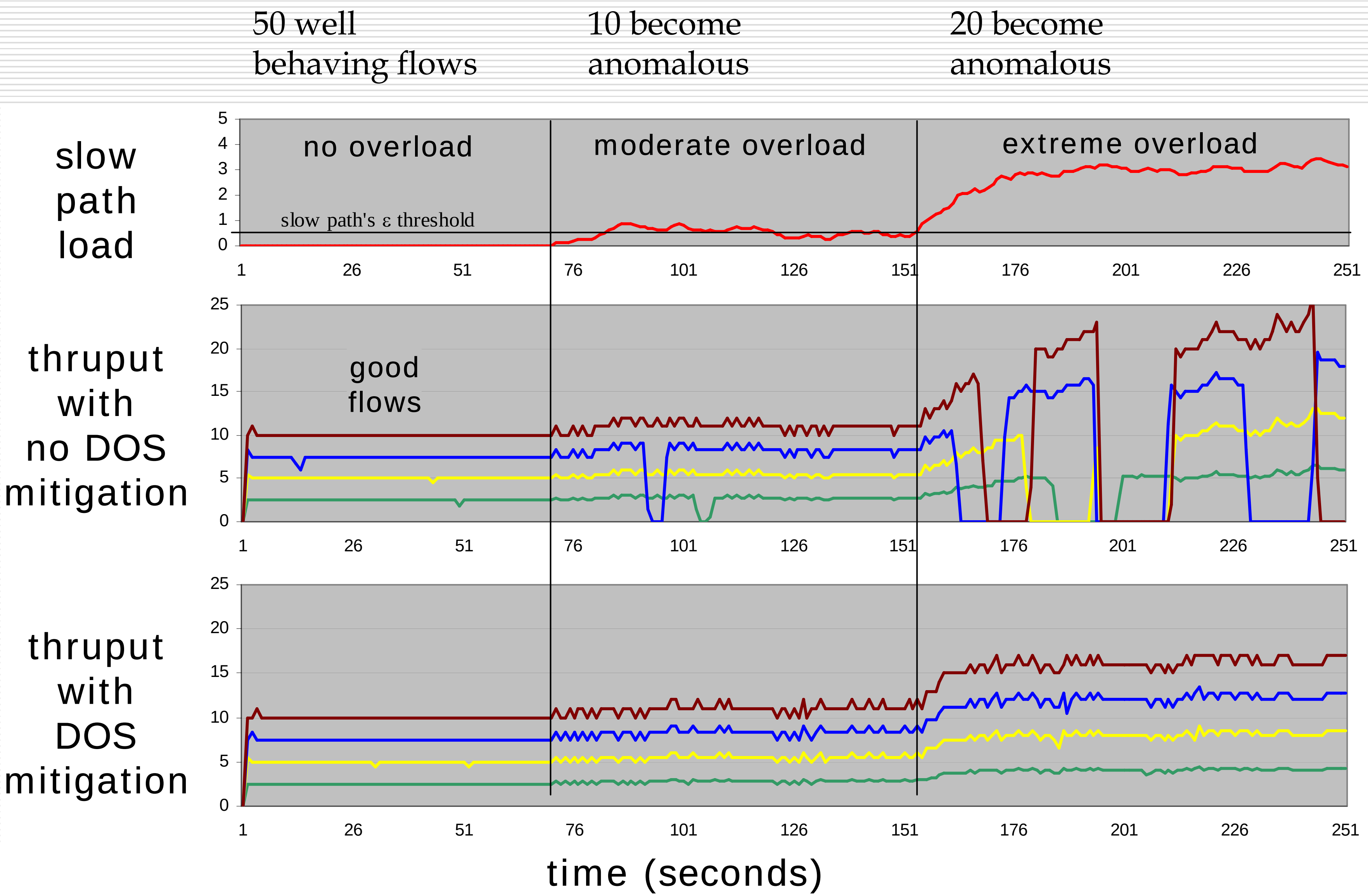
well behaving flows will suffer



Use per-flow anomaly counter
Counts # of packets sent to the slow path.

Flows with high anomaly counter value are attack flows
Send then to a low priority queue

Simulation of DoS Mitigation



Results of Splitting Prefix/Suffix

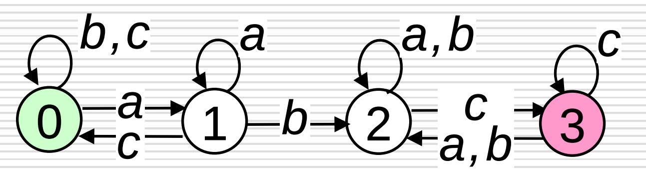
Source	# of Rules	Regular expressions before split			Prefixes after split		
		ASCII length	Number of DFA	Total memory	ASCII length	Number of DFA	Total memory
Cisco	68	44.1	6	973 MB	19.8	1	152 MB
Linux	70	67.2	4	30.7 MB	21.4	2	15.8 MB
Bro	648	23.64	1	3.77 MB	16.1	1	1.23 MB
Snort rule 1	22	59.4	5	114.6 MB	36.9	3	32.1 MB
Snort rule 2	10	43.72	2	64.2 MB	16	1	6.5 MB
Snort rule 3	19	30.72	N/A	N/A	13.8	2	2.42 MB

Slow path probability set to less than 0.01%

Second Contribution - HFA

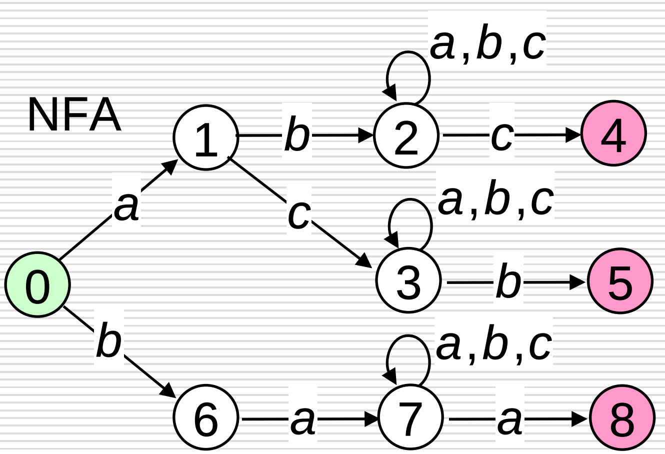
- NFAs are compact but slow
  - » Multiple active state
- DFAs are fast representation
  - » State explosion is serious problem
  - » State explosion mainly occurs due to the presence of closures
- Three patterns
  - » 3 separate DFAs create 12 states
    - – 3 active states
  - » NFA has only 9 states
    - – Up to 6 active state
  - » A single DFA creates 20 states
    - – 1 active state

$(ab.*c) \mid (ac.*b) \mid (ba.*a)$



1 of 3 DFAs – total 12 states

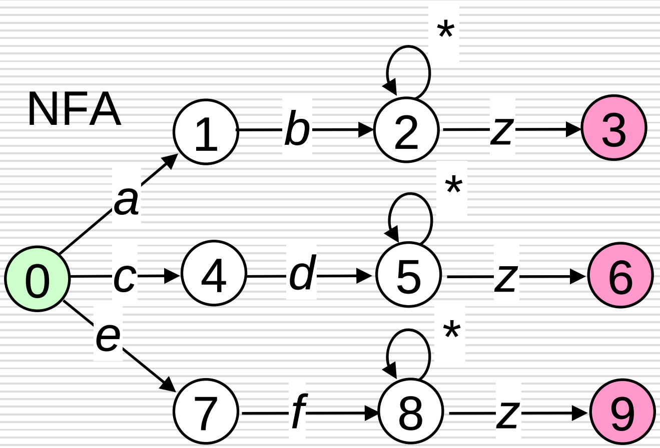
NFA



State Explosion in DFA

- State explosion occurs primarily because
  - » DFA has single active state
  - » Don’t remember anything but the current active state (amnesia)
- Requires a separate DFA state for every situation that may occur during NFA parse

$(ab.*z) \mid (cd.*z) \mid (ef.*z)$



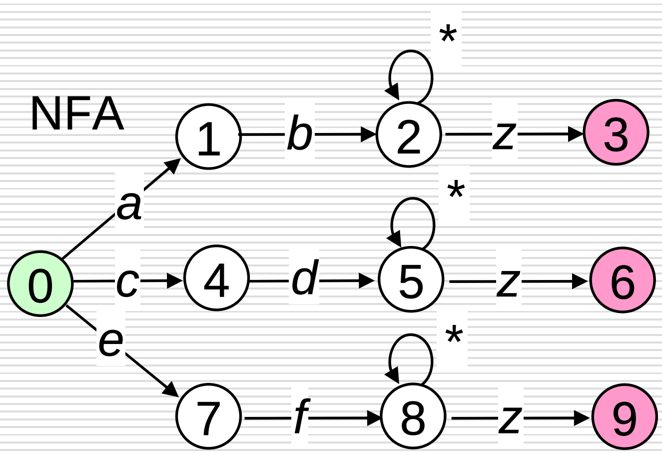
	Active states
Input: abcd	**0, 2, 5**
Input: abef	**0, 2, 8**
Input: cdef	**0, 5, 8**
Input: abcdef	**0, 2, 5, 8**

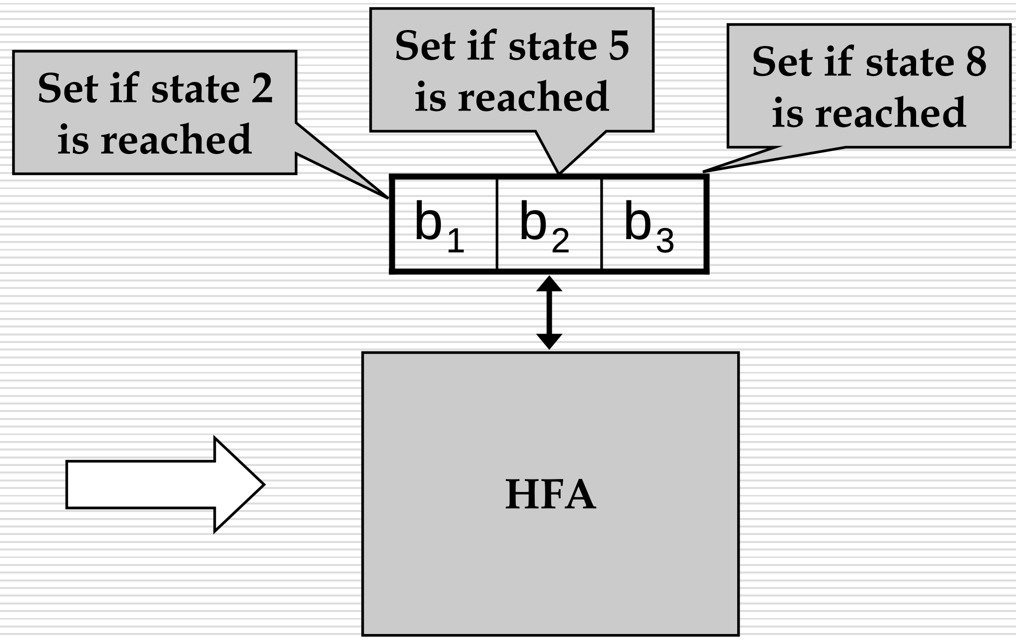
k* closures => Number of DFA states is exponential in *k

HFA

- Our solution is History based Finite Automata (HFA)
  - » Enable a single state of execution
  - » Use a bit to represent the condition that a closure is reached
  - » Certain transitions depends upon the bit values
  - » Bits are also updated as HFA makes its transitions

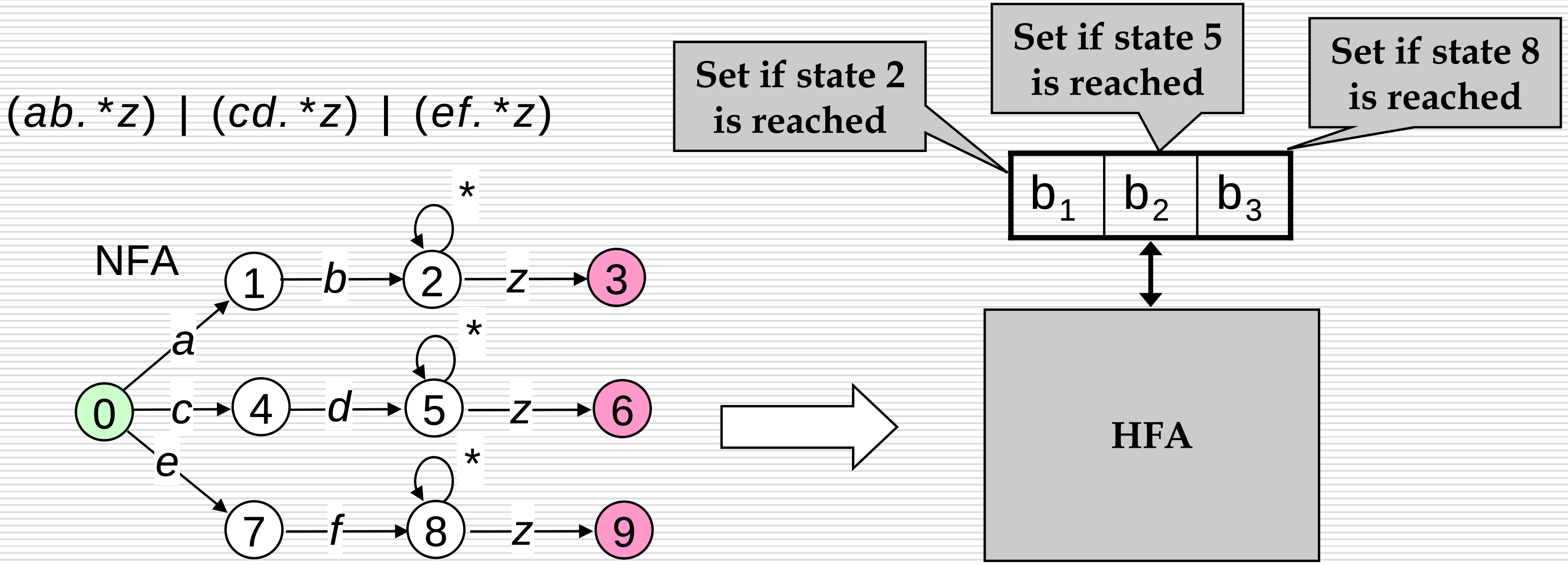
$(ab.{*}z) \mid (cd.{*}z) \mid (ef.{*}z)$





Benefits of HFA

- Single State of Execution – high performance
- Few bits are required (16, 32) – stored in registers
- Avoids state explosion – memory efficient



Results

Source	# of closures	DFA		H-FA			% space reduction with H-FA	H-FA parsing rate speedup
		# of automata	total # of states	# of automata	# of flags	Total # of states		
Cisco64	14	1	132784	1	6	3597	94.69	-
Cisco64	14	1	132784	1	13	1861	96.77	-
Cisco68	19	1	328664	1	17	2956	97.03	-
Snort 1	6	3	62589	1	5	583	97.40	3x
Snort 2	1	1	12703	1	1	71	98.58	-
Snort 3	5	2	4737	1	5	116	93.48	2x
Linux70	11	2	20662	1	9	1304	81.63	2x

- Thank you and Questions???