ARCH

Accelerating Regular Expression Matching Over Compressed HTTP

Presented by:

Omer Kochba

The Interdisciplinary Center Herzliya

Joint work with: Michela Becchi, Anat Bremler-Barr, David Hay and Yaron Koral

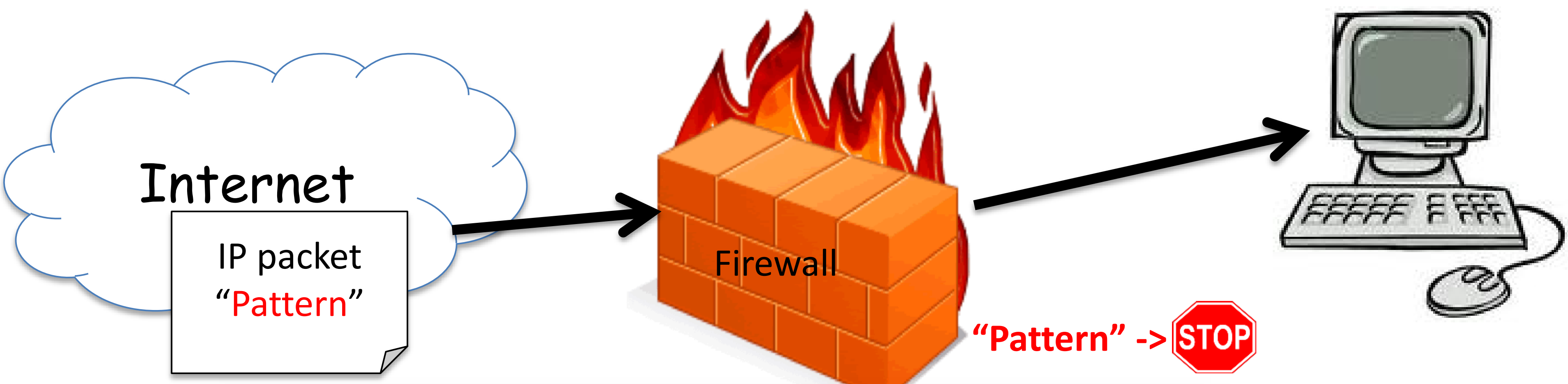
This work was supported by: The European Research Council, The Israeli Centers of Research Excellence, The Neptune Consortium, and National Science Foundation award CNS-1319748

Outline

- Motivation
- Background
  - Regular Expression Matching
  - DPI over Compressed HTTP
- ARCH
  - Input-Depth Calculation
- Experiment
- Additional usages for Input-Depth

Deep Packet Inspection

- Processing of the packet payload
- Identify occurrences from predefined patterns: strings or regular expressions



Snort (cross site inf. disclosure):
```
<\x21DOCTYPE\s+[^>]*SYSTEM[^>]*>.*\x2EparseError
```
ClamAV (Cabir A. worm):
```
886f1f10123a001019040010e5f79547e6ad0100bd006f006400750063007
```
Bro (MS Office 2007 xml docs id):
```
\x50\x4B\x03\x04\x14\x00\x06\x00
```

Motivation

- High volume of compressed HTTP traffic
  - Compressed by the server, decompressed by the browser
  - 84% of top 1000 sites, 60% of all web sites
- DPI is the current bottleneck of middle-boxes
- **ARCH – First algorithm to accelerate regular expression matching of compressed HTTP**

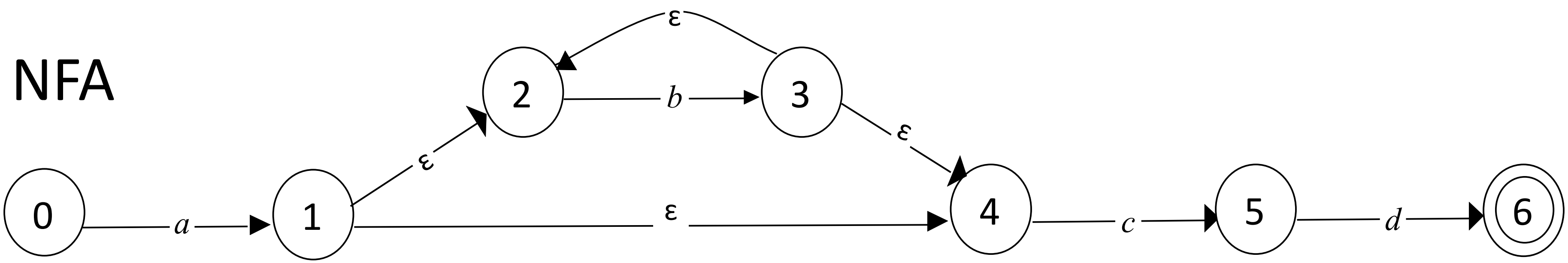
Regular Expression Matching

- Non-Deterministic Finite Automaton (NFA) – space efficient
- Deterministic Finite Automaton (DFA) – time efficient
- Hybrid FA (*CoNext 2007*) – space/time efficiency

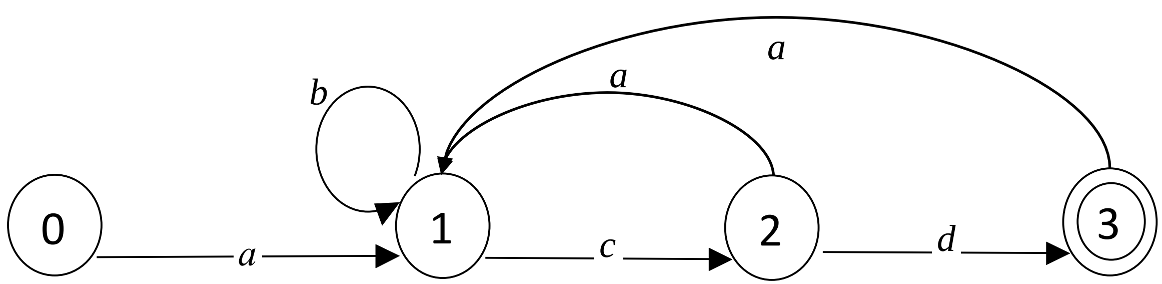
Pattern: **ab*cd**

Zero or more occurrences of the character ‘b’

NFA



Equivalent DFA



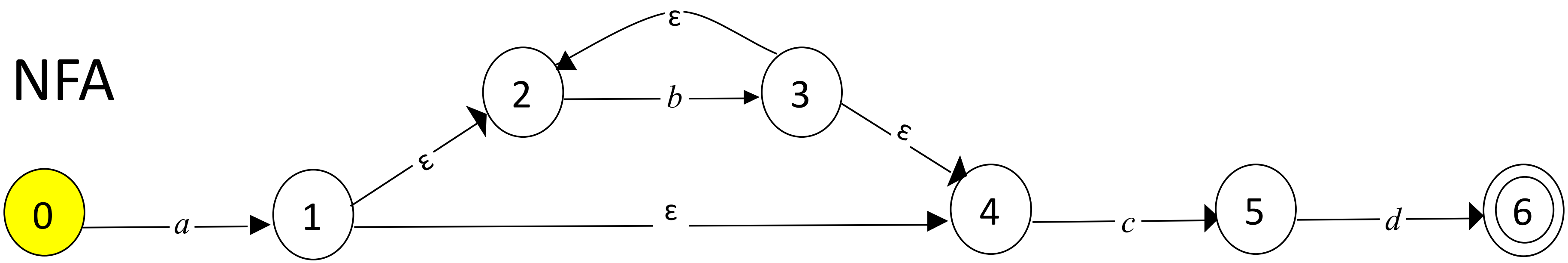
Regular Expression Matching

- An NFA may have multiple active states
- A DFA will have only one current state
- An NFA contains ε transitions

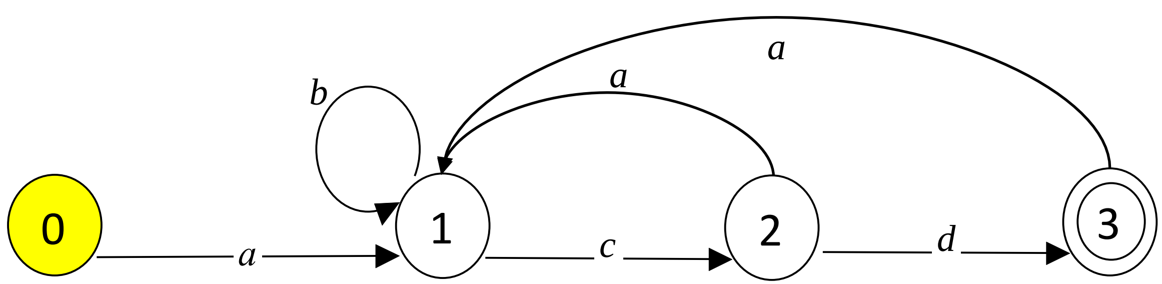
Pattern: **ab*cd**

Input:

NFA



Equivalent DFA



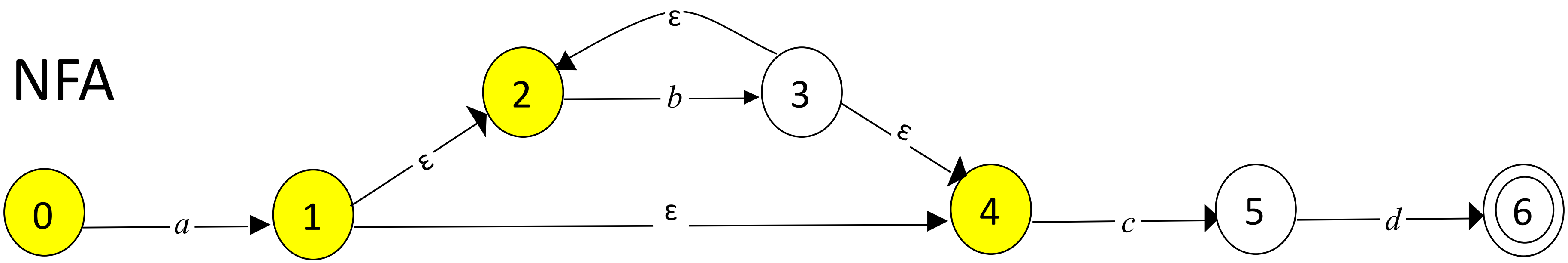
Regular Expression Matching

- An NFA may have multiple active states
- A DFA will have only one current state
- An NFA contains ε transitions

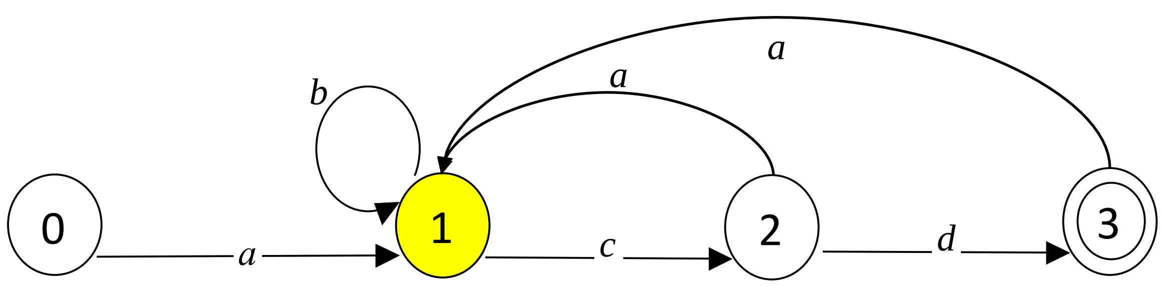
Pattern: **ab*cd**

Input: **a**

NFA



Equivalent DFA

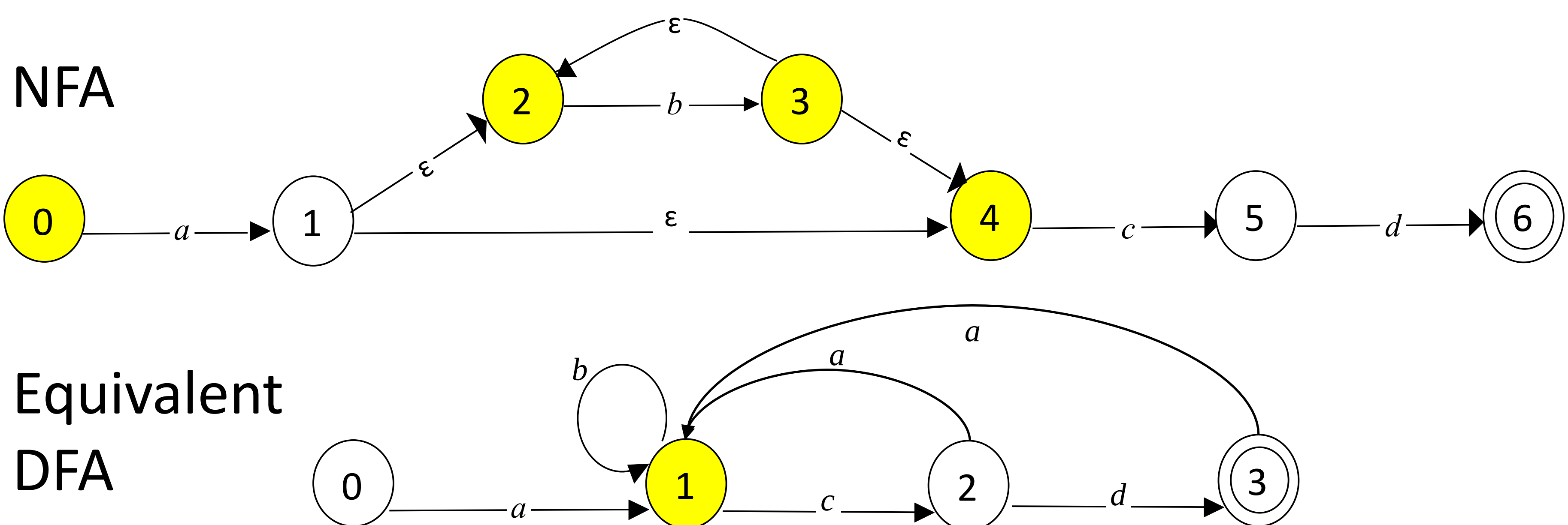


Regular Expression Matching

- An NFA may have multiple active states
- A DFA will have only one current state
- An NFA contains ε transitions

Pattern: **ab*cd**

Input: **ab**

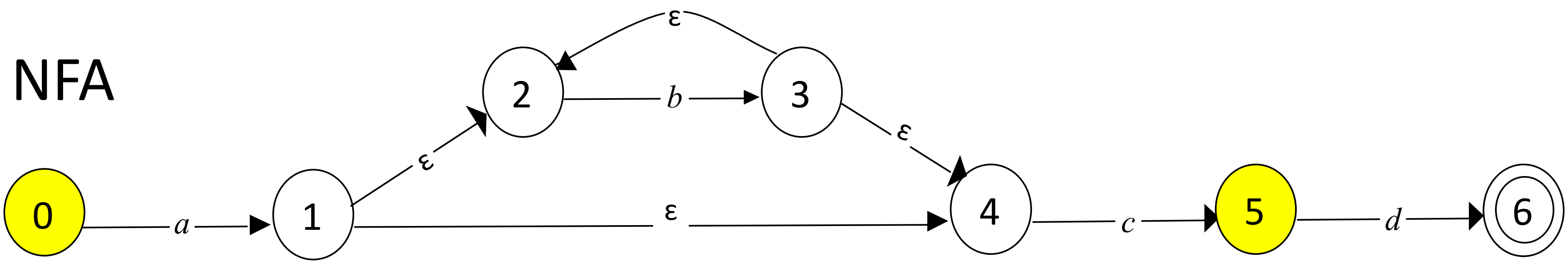


Regular Expression Matching

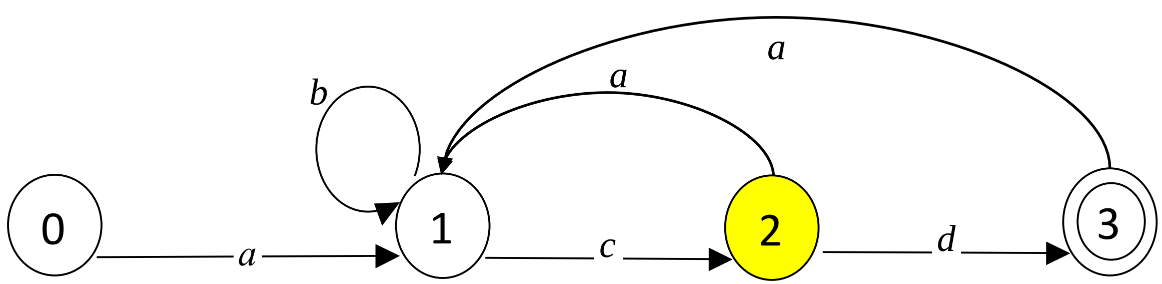
- An NFA may have multiple active states
- A DFA will have only one current state
- An NFA contains ε transitions

Pattern: **ab*cd**	Input: **abc**

NFA

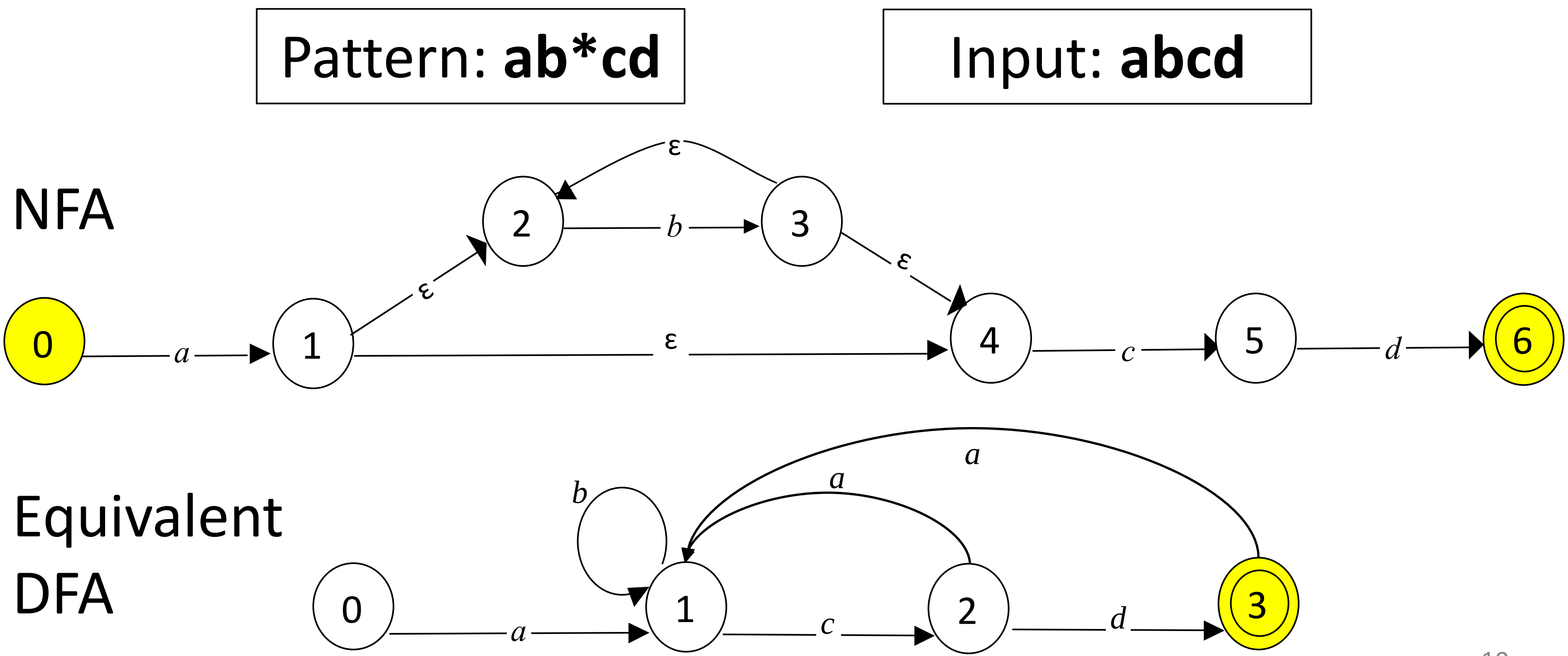


Equivalent DFA



Regular Expression Matching

- The automatons are equivalent
- Both will reach accepting state together



Compressed HTTP

- Compressed HTTP is a standard of HTTP 1.1
- Mainly uses GZIP and DEFLATE
- Based on LZ77 (an adaptive compression)

<u>Plain Text:</u>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="en-US"><head><meta http-equiv=
```

<u>Compressed Text:</u>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD{26,6}4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<{20,4} lang="en-US{20,5}head{7,3}meta {73,4)-equiv=
```

<u>Compression Algorithm:</u>

1. Identify repeated strings
2. Replace each string with the *(distance, length)* syntax
3. Further compress the syntax using Huffman Coding

DPI on Compressed HTTP

- An LZ77 pointer represents a repeated string
- It is possible to skip scanning most of it
- Borders must still be considered
- Existing works discuss matching acceleration but are limited to string matching *(Infocom 2009)*

Traffic = e m c d e f c e a { 7 , 7 } b b c d

Uncompressed= e m c d e f c e a c d e f c e a b b c d

Pattern: **ab*cd**

ARCH

- Upon encountering a repeated string:
    1. Scan the left border until **Input-Depth**(*b*) ≤ j
        - ***b*** is the current byte, ***j*** is its index inside the pointer
        - **Input-Depth** – number of bytes that can be part of a future match
    2. Skip internal pointer area
    3. Scan the right border

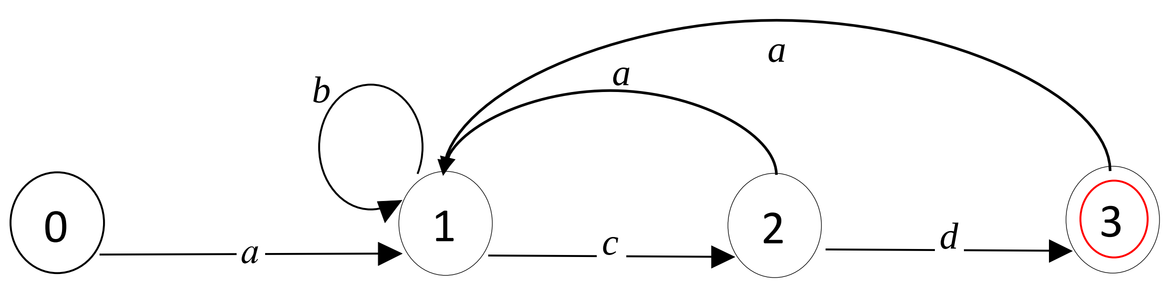
Traffic = e m c d e f c e a { 7 , 7 } b b c d

Uncompressed= e m c d e f c e a c d e f c e a b b c d

Pattern: **ab*cd**

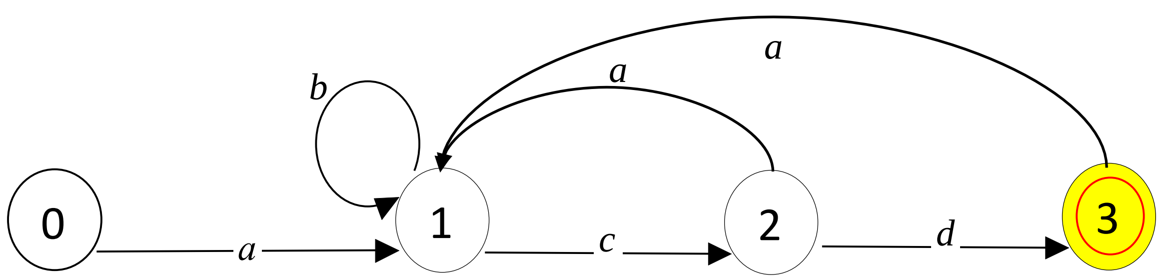
Input-Depth=0

j=3



ARCH

- **ARCH** is mainly based on *Input-Depth*
  - ***Input-Depth(T)*** is the length of the shortest suffix of *T* in which inspection starting at S_0 ends at S
- For string matching, *Input-Depth* = DFA-Depth
- For regular expression matching it varies
  - depends on both the automaton and the input



Pattern: **ab*cd**

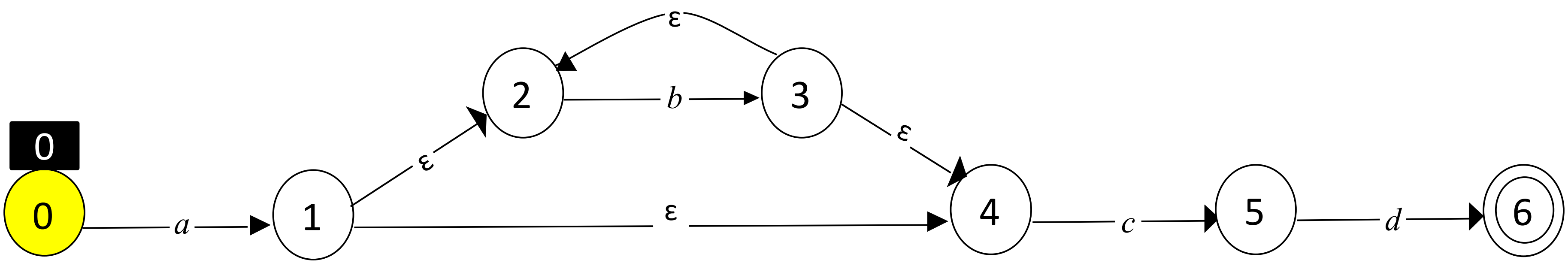
- Input = *eabbcd*
- DFA-Depth = 3
- **Input-Depth = 5**

Input-Depth for NFA

Algorithm for Active States NFA:

- *Input-Depth* parameter for each active state
- When a state is added to the list of active states:
  - *Input-Depth* = predecessor's *Input-Depth* + 1 (labeled transition)
  - *Input-Depth* = predecessor's *Input-Depth* (epsilon transition)
- Total *Input-Depth = max(Input-Depth[ActiveStates])*

Pattern: **ab*cd**



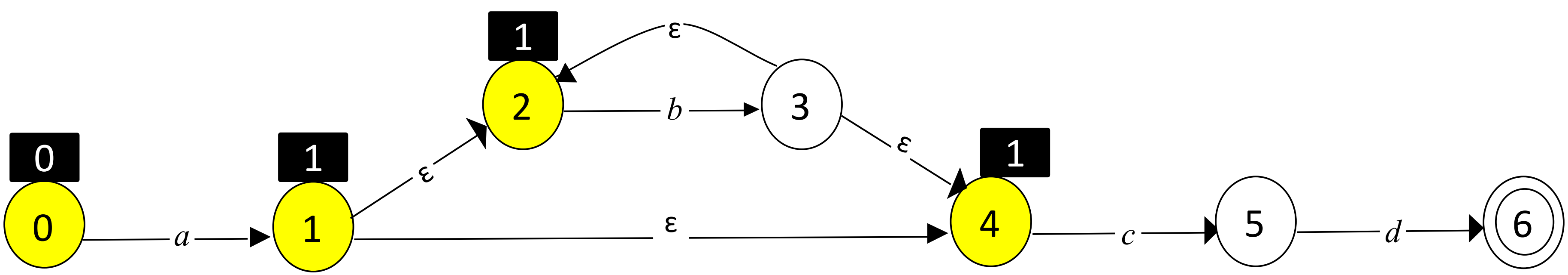
- Input =
- **Input-Depth = 0**

Input-Depth for NFA

<u>Algorithm for Active States NFA:</u>

- *Input-Depth* parameter for each active state
- When a state is added to the list of active states:
  - *Input-Depth* = predecessor's *Input-Depth* + 1 (labeled transition)
  - *Input-Depth* = predecessor's *Input-Depth* (epsilon transition)
- Total *Input-Depth = max(Input-Depth[ActiveStates])*

Pattern: **ab*cd**



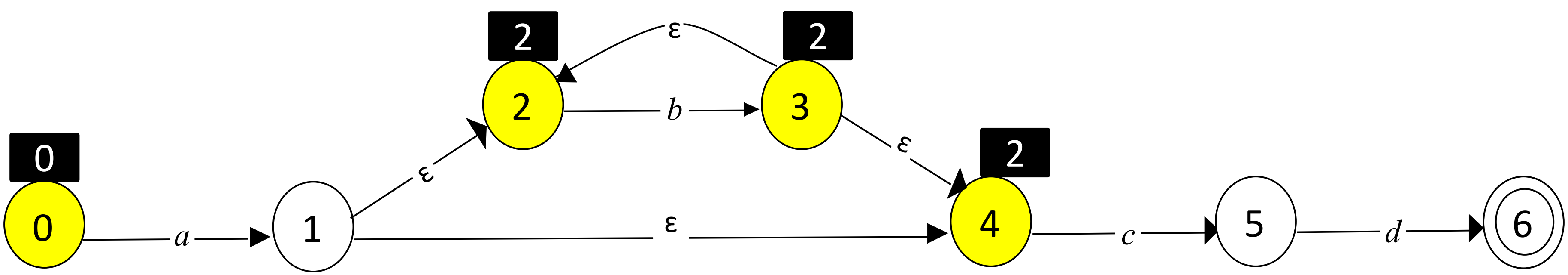
- Input = *a*
- **Input-Depth = 1**

Input-Depth for NFA

Algorithm for Active States NFA:

- *Input-Depth* parameter for each active state
- When a state is added to the list of active states:
  - *Input-Depth* = predecessor's *Input-Depth* + 1 (labeled transition)
  - *Input-Depth* = predecessor's *Input-Depth* (epsilon transition)
- Total *Input-Depth = max(Input-Depth[ActiveStates])*

Pattern: **ab*cd**



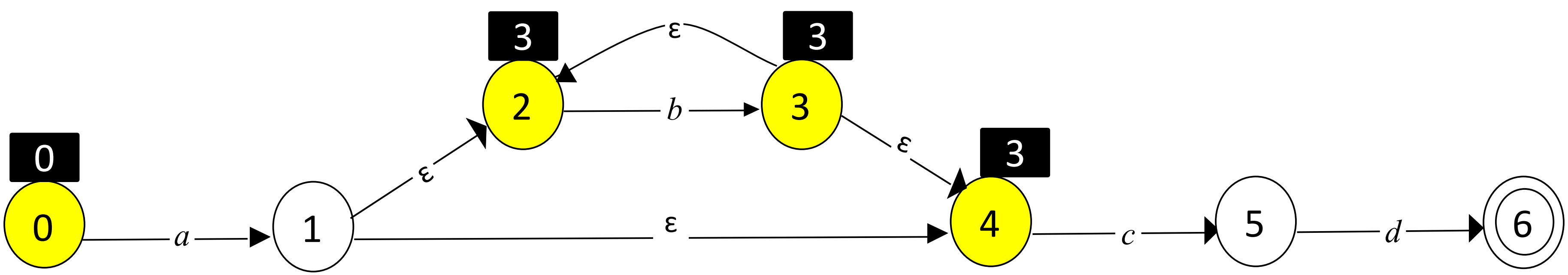
- Input = *ab*
- **Input-Depth = 2**

Input-Depth for NFA

<u>Algorithm for Active States NFA:</u>

- *Input-Depth* parameter for each active state
- When a state is added to the list of active states:
  - *Input-Depth* = predecessor's *Input-Depth* + 1 (labeled transition)
  - *Input-Depth* = predecessor's *Input-Depth* (epsilon transition)
- Total *Input-Depth = max(Input-Depth[ActiveStates])*

Pattern: **ab*cd**



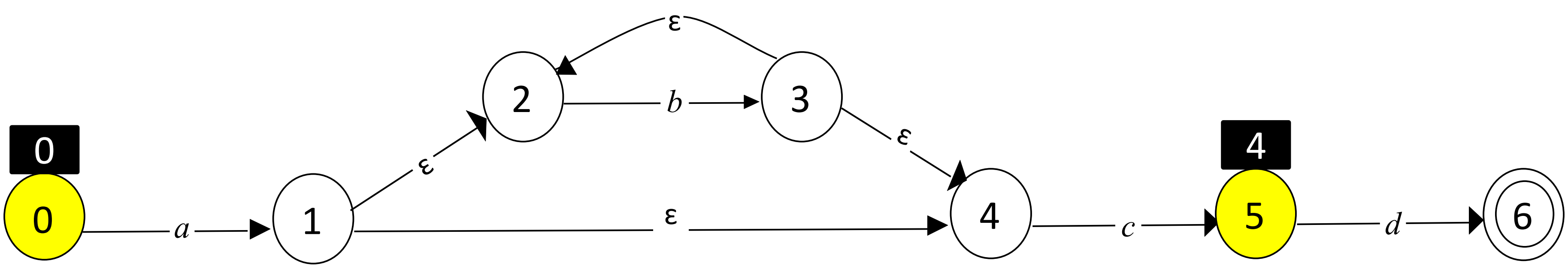
- Input = *abb*
- **Input-Depth = 3**

Input-Depth for NFA

<u>Algorithm for Active States NFA:</u>

- *Input-Depth* parameter for each active state
- When a state is added to the list of active states:
  - *Input-Depth* = predecessor's *Input-Depth* + 1 (labeled transition)
  - *Input-Depth* = predecessor's *Input-Depth* (epsilon transition)
- Total *Input-Depth = max(Input-Depth[ActiveStates])*

Pattern: **ab*cd**



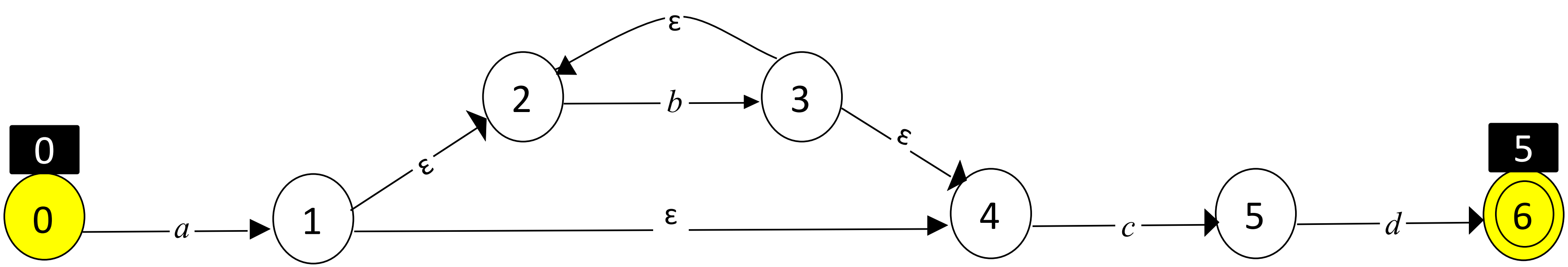
- Input = *abbc*
- **Input-Depth = 4**

Input-Depth for NFA

<u>Algorithm for Active States NFA:</u>

- *Input-Depth* parameter for each active state
- When a state is added to the list of active states:
  - *Input-Depth* = predecessor's *Input-Depth* + 1 (labeled transition)
  - *Input-Depth* = predecessor's *Input-Depth* (epsilon transition)
- Total *Input-Depth = max(Input-Depth[ActiveStates])*

Pattern: **ab*cd**



- Input = *abbcd*
- **Input-Depth = 5**

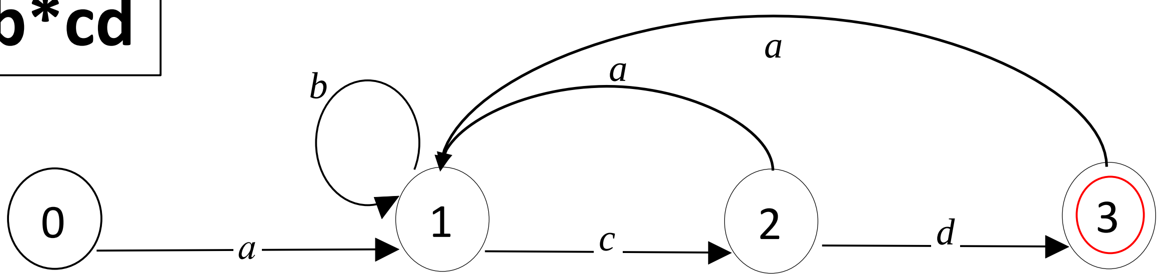
Input-Depth for DFA

- NFA *Input-Depth* is exact
- A DFA transition may result in:
  - Increasing the *Input-Depth* by one
  - Decreasing the *Input-Depth* by any value (unlike NFA)
- For DFA we provide an upper bound:
  - Simple and Complex states
  - Positive and Negative transitions

Simple and Complex States

- *A* ***simple state*** *S is a state for which all possible input strings that upon scan from S_0 terminate at S have the same length*
- All other states are **complex**
- Identified during the construction algorithm

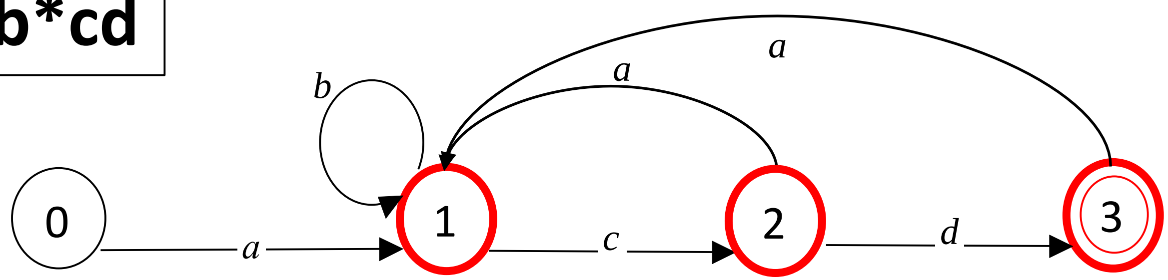
Pattern: **ab*cd**



Simple and Complex States

- *A **simple state** S is a state for which all possible input strings that upon scan from S_0 terminate at S have the same length*
- All other states are **complex**
- Identified during the construction algorithm

Pattern: **ab*cd**

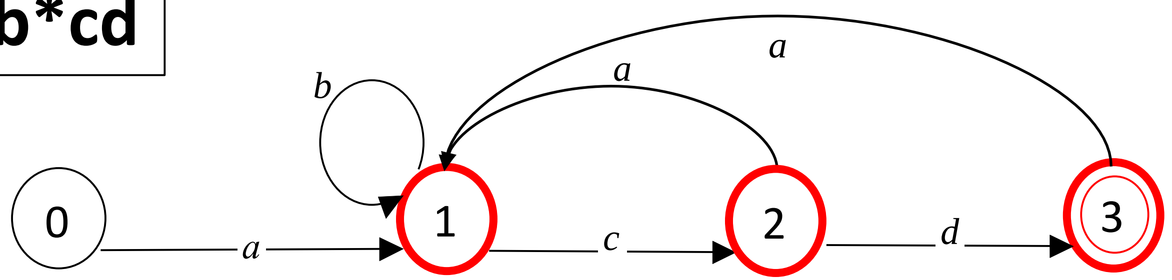


➢ Complex states are marked in red

Simple and Complex States

- Upon traversal:
  - to a simple state – *Input-Depth* = DFA-Depth
  - to a complex state – *Input-Depth* += 1

Pattern: **ab*cd**

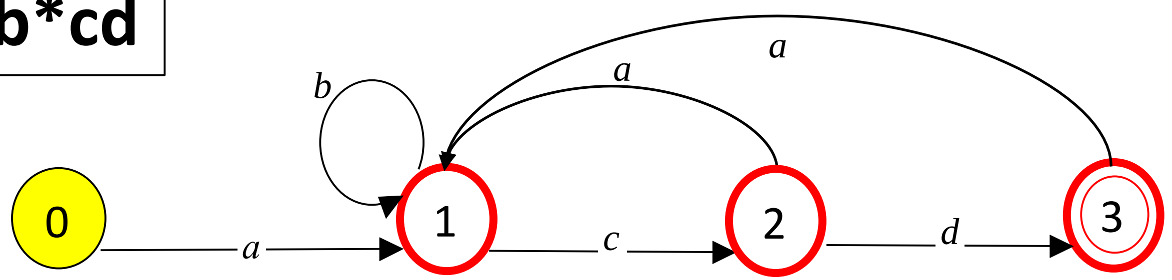


➢ Complex states are marked in red

Simple and Complex States

- Upon traversal:
  - to a simple state – *Input-Depth* = DFA-Depth
  - to a complex state – *Input-Depth* += 1

Pattern: **ab*cd**

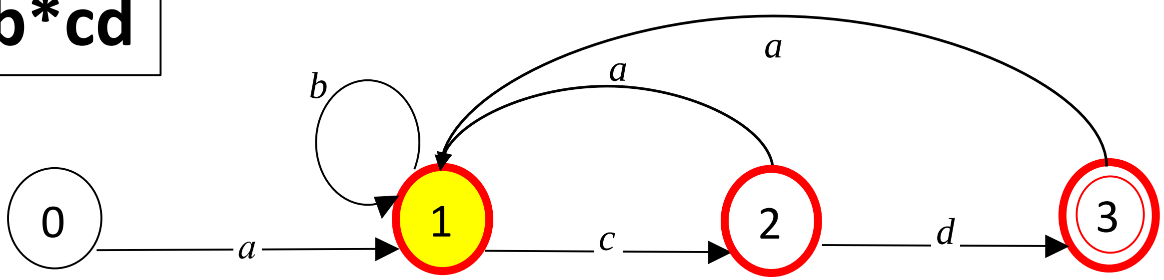


- Input =
- **Input-Depth = 0**

Simple and Complex States

- Upon traversal:
  - to a simple state – *Input-Depth* = DFA-Depth
  - to a complex state – *Input-Depth* += 1

Pattern: **ab*cd**

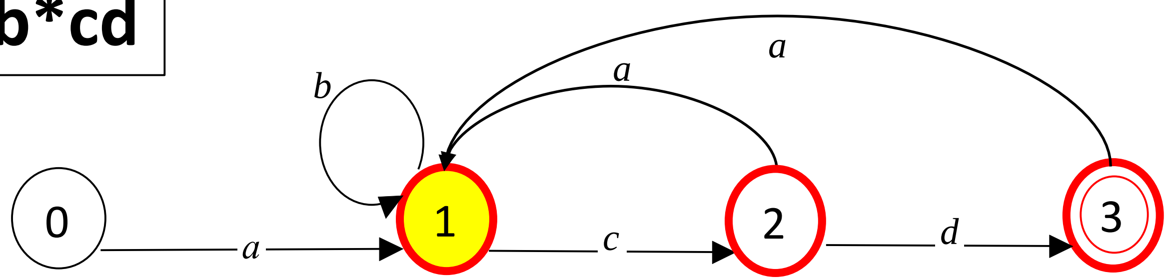


- Input = *a*
- **Input-Depth = 1**

Simple and Complex States

- Upon traversal:
  - to a simple state – *Input-Depth* = DFA-Depth
  - to a complex state – *Input-Depth* += 1

Pattern: **ab*cd**

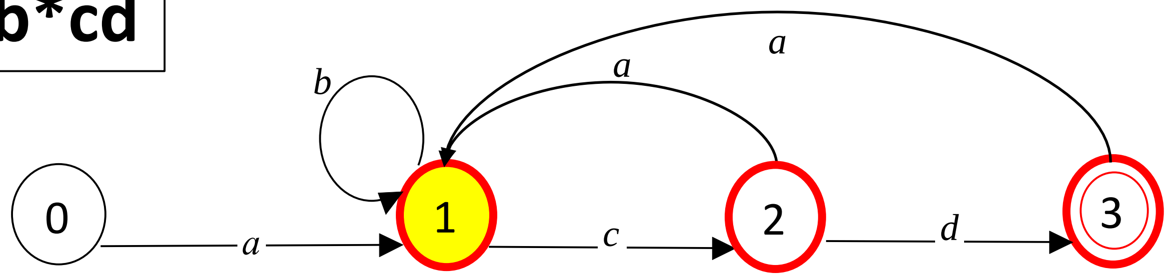


- Input = *ab*
- **Input-Depth = 2**

Simple and Complex States

- Upon traversal:
  - to a simple state – *Input-Depth* = DFA-Depth
  - to a complex state – *Input-Depth* += 1

Pattern: **ab*cd**

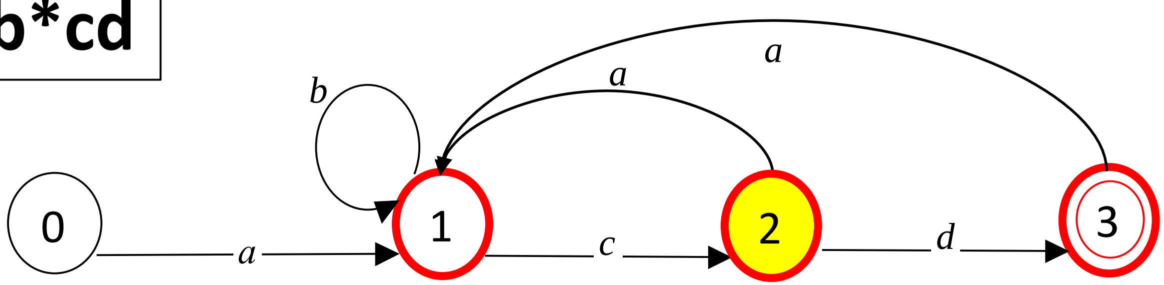


- Input = *abb*
- **Input-Depth = 3**

Simple and Complex States

- Upon traversal:
  - to a simple state – *Input-Depth* = DFA-Depth
  - to a complex state – *Input-Depth* += 1

Pattern: **ab*cd**

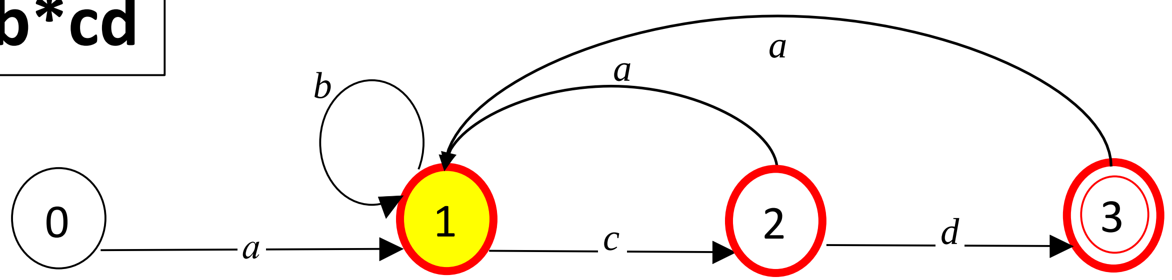


- Input = *abbc*
- **Input-Depth = 4**

Simple and Complex States

- Upon traversal:
  - to a simple state – *Input-Depth* = DFA-Depth
  - to a complex state – *Input-Depth* += 1

Pattern: **ab*cd**



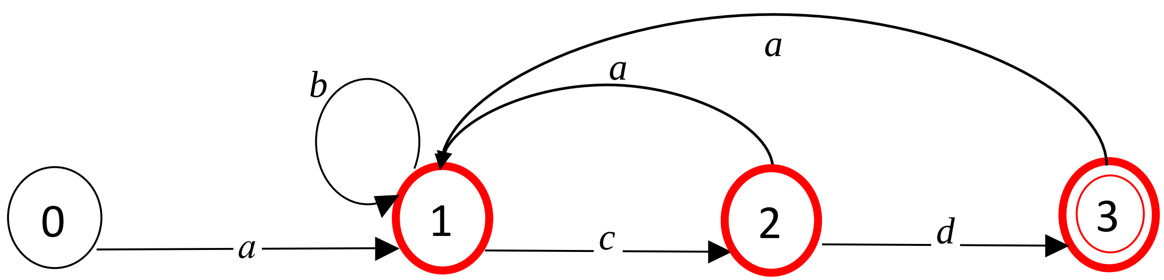
- Input = *abbca*
- **App. Input-Depth = 5**
- **Actual Input-Depth = 1**

Simple and Complex States

- Approximation maintains correctness but may impact performance
- It works well in practice:
  - *Input-Depth* is normally low (avg. = 1.1)
  - Most complex states are at high depths (avg. > 5)
- In theory we can approximate better

Positive and Negative Transitions

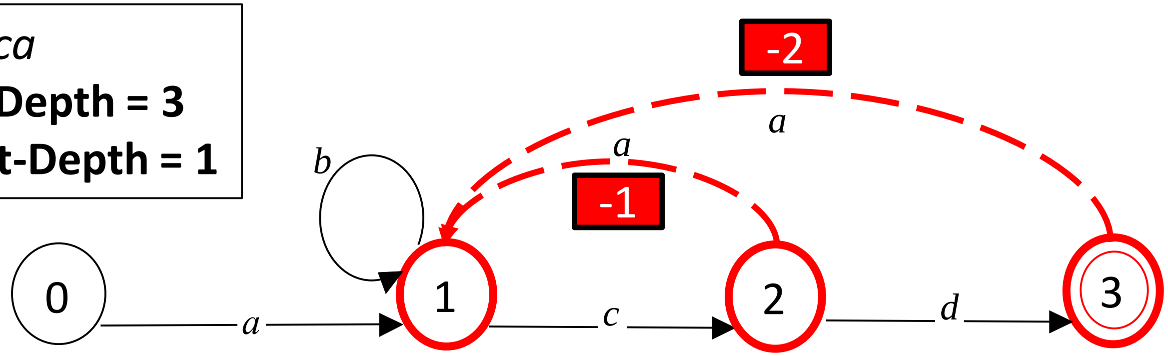
- *Input-Depth* depends on both the states and the transition between them
- We define two types of transitions:
  - A **positive** transition – increases the *Input-Depth* by one
  - A **negative** transition – decreases the *Input-Depth* by $x \geq 0$



Positive and Negative Transitions

- During the DFA construction algorithm determine:
  - Transition Type (positive or negative)
  - Transition *Input-Depth* delta (for negative transitions)

- Input = *abbca*
- **App. Input-Depth = 3**
- **Actual Input-Depth = 1**



➢ Negative transitions are dashed and red

Experiment

- Rulesets from the Snort IPS
- 2301 compressed HTML pages from Alexa top 500 global sites
- 358MB in uncompressed form and 61.2MB in compressed form
- Compared with a simple baseline algorithm, which does not perform any byte skipping

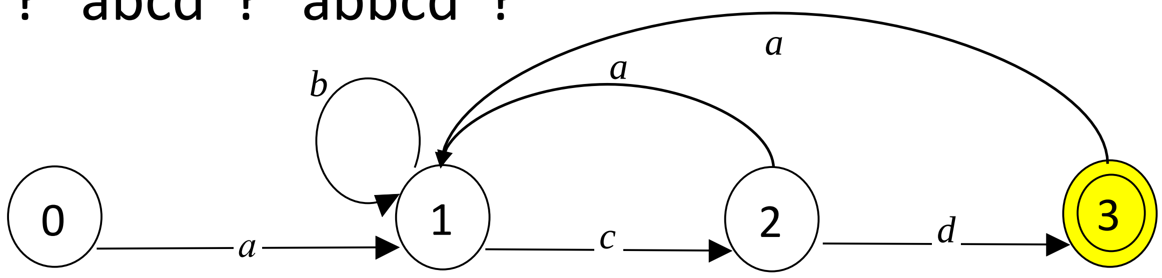
Experimental Results

Automaton Type	Average Skip Rate	Average Processing Time Improvement	Overhead
ARCH-NFA	77.99%	77.21%	1%
ARCH-DFA	77.69%	69.19%	11%
Hybrid-FA	77.88%	69.41%	11%

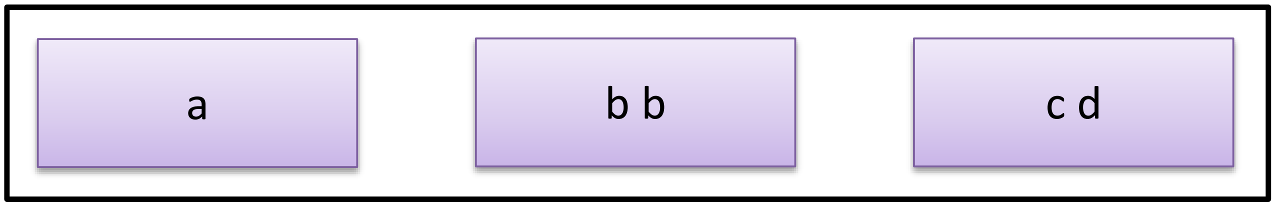
- The overall processing time of ARCH-NFA is **40 times longer** than ARCH-DFA
- The space requirements of ARCH-NFA are **18 times smaller** than those of ARCH-DFA

Additional usages for Input-Depth

- Extract the string that relates to a matched pattern without rescanning the packet
  - “acd”? “abcd”? “abbcd”?



- Determine the number of bytes that should be stored to handle cross-packet DPI



Conclusion

- First generic framework to accelerate any regular expression matching over compressed traffic
- Significant performance improvement compared to a plain scan: **70% faster**
- Suitable for line rate DPI
- *Input-Depth* important to solve other problem domains