Hyperscan: A Fast Multi-pattern Regex Matcher for Modern CPUs

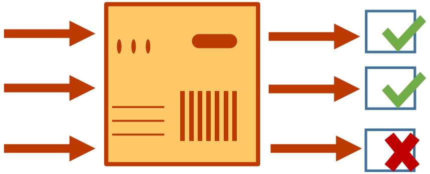
Xiang Wang[1], Yang Hong[1], Harry Chang[1], KyoungSoo Park[2], Geoff Langdale[3], Jiayu Hu[1] and Heqing Zhu[1]

1 Intel Corporation; 2 KAIST; 3 branchfree.org

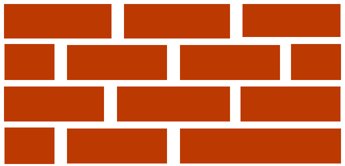
Networking Applications with Regex Matching

- Deep packet inspection (DPI) – key functionality of L7 traffic monitoring
- Regular expression (regex) matching – core element of DPI
- Big problem – regex matching is SLOW

IPS/IDS



WAF



Application Identification

Current Best Practice: Prefilter-based Pattern Matching

Rule 0:

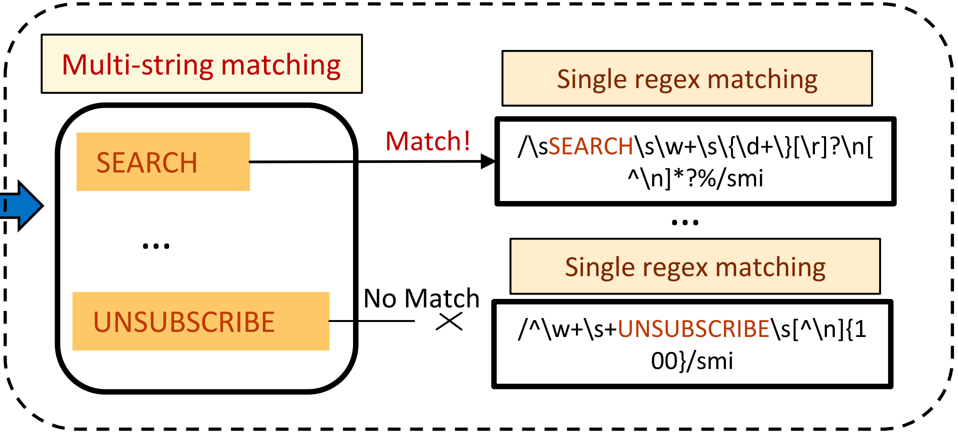
```
content:"SEARCH";
pcre:"/\sSEARCH\s\w+\s\{\d+\}[\r]?\n[^\n]*?%/smi"
```

...

Rule N:

```
content:"UNSUBSCRIBE";
pcre:"/^\w+\s+UNSUBSCRIBE\s[^\n]{100}/smi";
```

Two-stage Pattern Matching



Problems with Prefilter-based Pattern Matching

Manual choice of improper string keywords

```
content:“/';
pcre:"/(?=[defghilmnoqrstwz])(m(ookflolfctm\x2fnmot\.fmu|clvompycem\x2fcen\.vcn)"
```

Duplicate matching of the string keywords

Complex regexes lead to slow NFA

Pattern

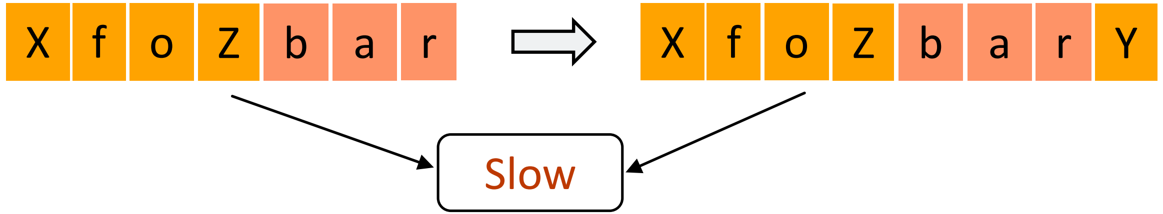
.*foo[^x]barY+

Input

XfoZbarY

String Matching for “bar”

Regex Matching



Contributions

Issues ✗

Problems with current best practices
- Manual choice of improper string keywords
- Duplicate matching of the string keywords
- Complex regexes lead to slow NFA

→ **Novel regex decomposition**

Suboptimal matching performance
- Slow multi-string matching
- Slow NFA matching

→ **SIMD-based pattern matching**
- Efficient multi-string matching
- Fast bit-based NFA

Solutions ✓

Outcome →
- Snort: 8.7x Speedup
- Multi-string matching: 3.2x Speedup over DFC
- Multi-regex matching: 13.5x Speedup over RE2

Wide Adoption of Hyperscan

- Successfully deployed by over 40 commercial projects globally
- In production use by tens of thousands of cloud servers in data centers
- Integrated into 37 open-source projects

Regex Decomposition

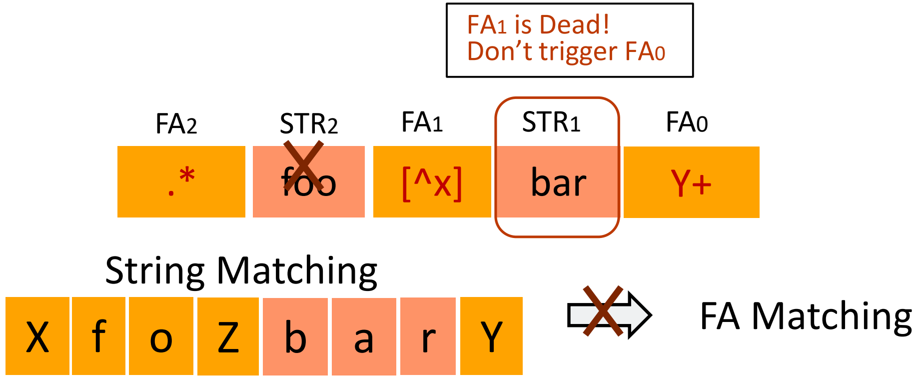
Decomposition-based Matching

Pattern

.*foo[^x]barY+

Input

XfoZbarY



- Decomposes a pattern into string (STR) and subregex (FA) components
- String matching is the entrance
- All components have to be matched in order

- No duplicate string keyword matching
- Smaller FAs with fast DFA matching
- Facilitate multi-regex matching

Key Issues with Regex Decomposition

- How to automatically decompose a regex?
- How many real-world regexes can be decomposed?

Key Issues with Regex Decomposition

- How to automatically decompose a regex?
- How many real-world regexes can be decomposed?

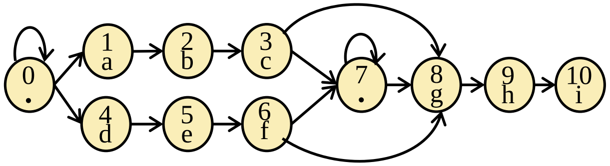
Graph-based Regex Decomposition

- Textual regex decomposition is often tricky, e.g. /b[il1]l\s{0,10}/
- Graph structure delivers more insights

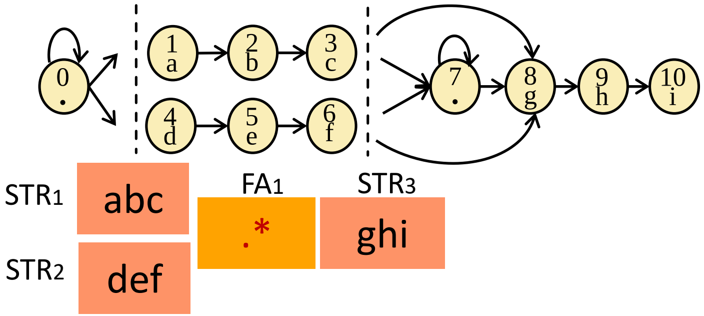
Regex

(abc|def).*ghi

Glushkov NFA

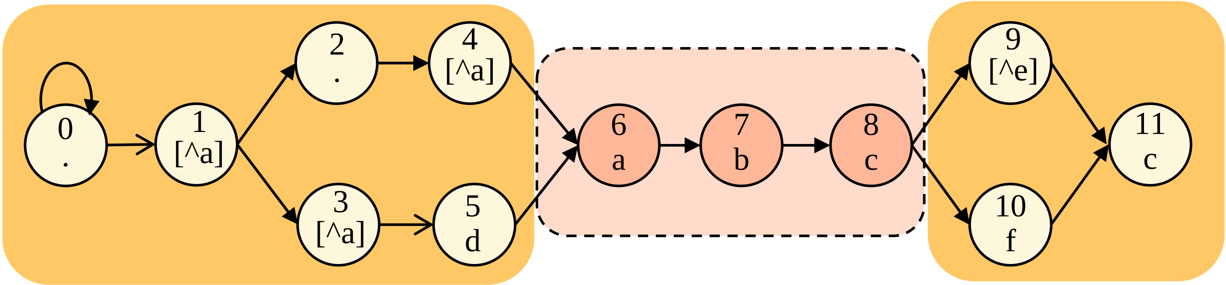


Graph-based Decomposition
1) Dominant Path Analysis
2) Dominant Region Analysis
3) Network Flow Analysis

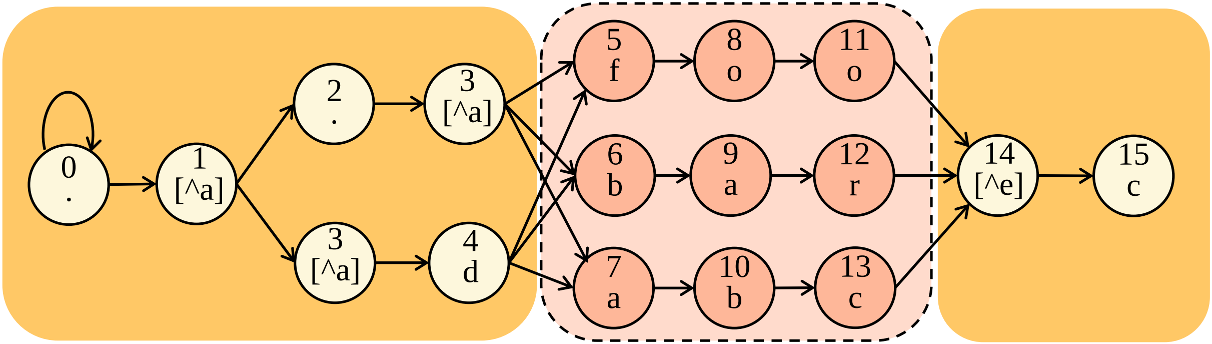


Graph-based String Extraction

Dominant Path Analysis



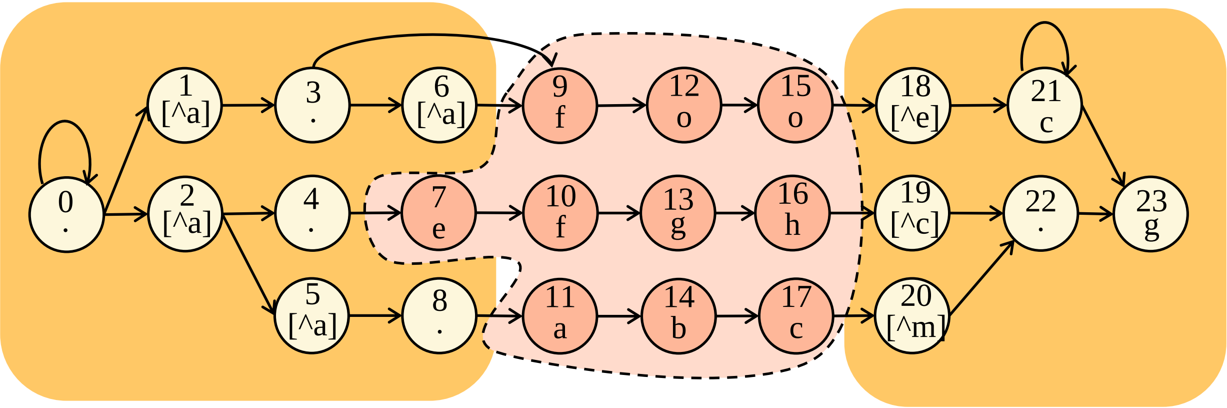
Dominant Region Analysis



Graph-based String Extraction

Network Flow Analysis

- Finds a string (or multiple strings) that ends at the edge
- Assigns a score inversely proportional to the length of the string(s) ending at the edge
- Runs "max-flow min-cut" algorithm [1] to find a minimum cut-set



[1]Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM, 19(2):248–264, 1972.

Key Issues with Regex Decomposition

- How to automatically decompose a regex?
- How many real-world regexes can be decomposed?

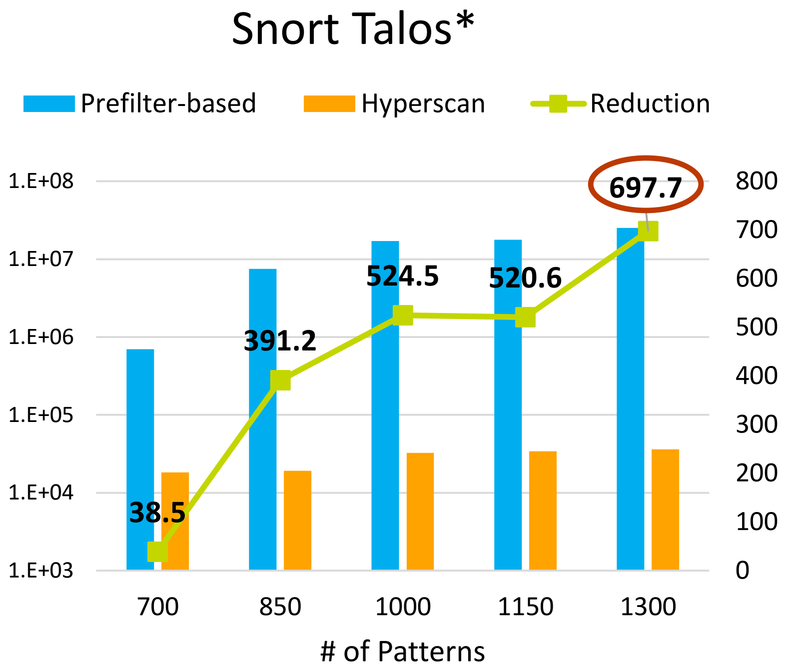
Effectiveness of Graph Analysis on Real-world Rules

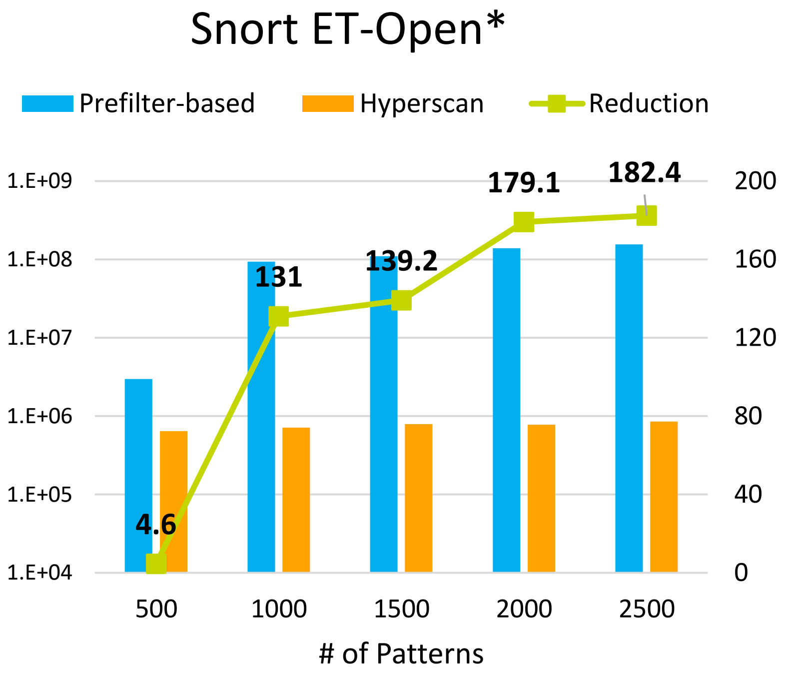
Majority of Regex Rules are Decomposable

Dominant Path Analysis is Effective

Ruleset	Total	All Graph Analyses	Dominant Path	Dominant Region	Network Flow
Snort Talos (May 2015)	1663	94.0%	93.3%	1.9%	1.0%
Snort ET-open 2.9.0	7564	89.3%	86.9%	1.3%	2.7%
Suricata 4.0.4	7430	87.5%	85.0%	1.3%	2.7%

Quality of Automatically Extracted Keywords



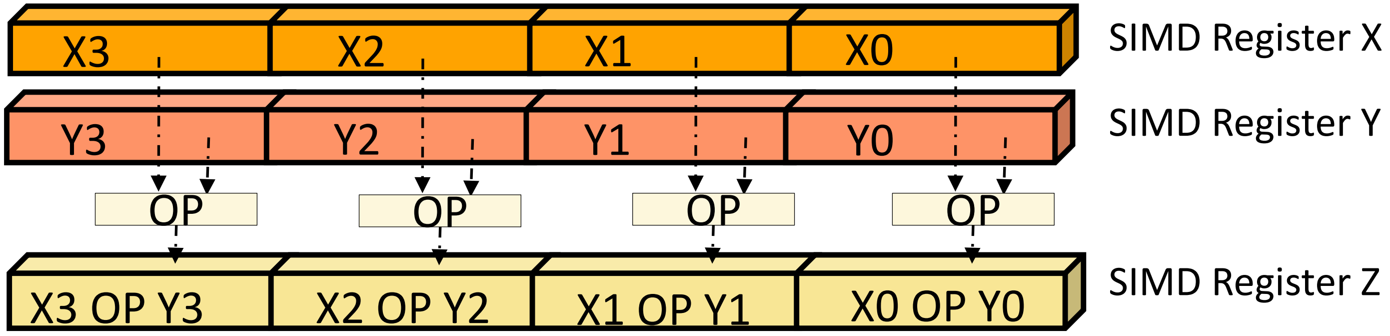


* Left vertical axis: # of regex matching process invocations (In logarithmic scale based on 10)
* Right vertical axis: reduction of Hyperscan

SIMD-based Pattern Matching

How to Accelerate Pattern Matching Algorithms?

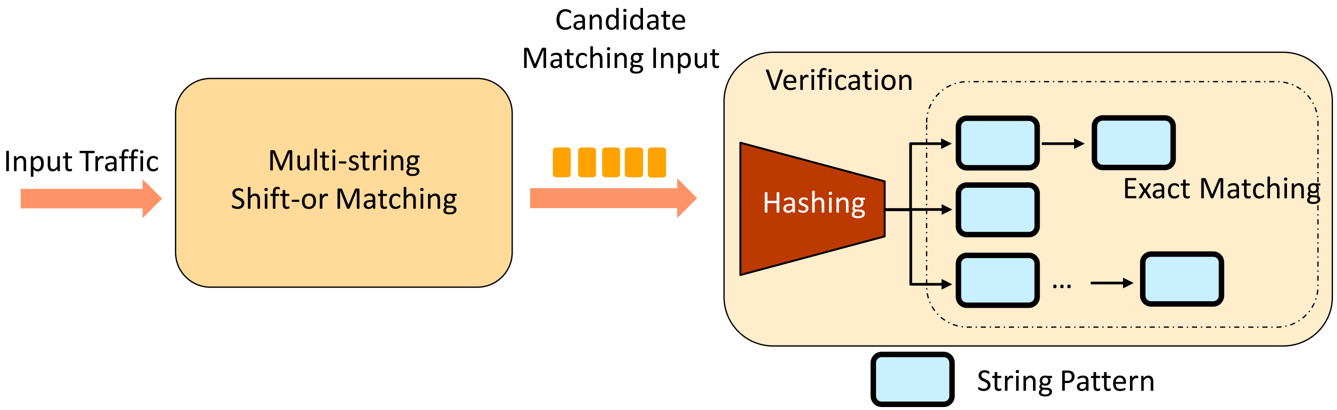
- Modern CPUs support SIMD (Single Instructions Multiple Data) to exploit data level parallelism
- SIMD instructions can boost database pattern matching by 2x [1]
- Accelerates both multi-string and FA matching with SIMD as the goal



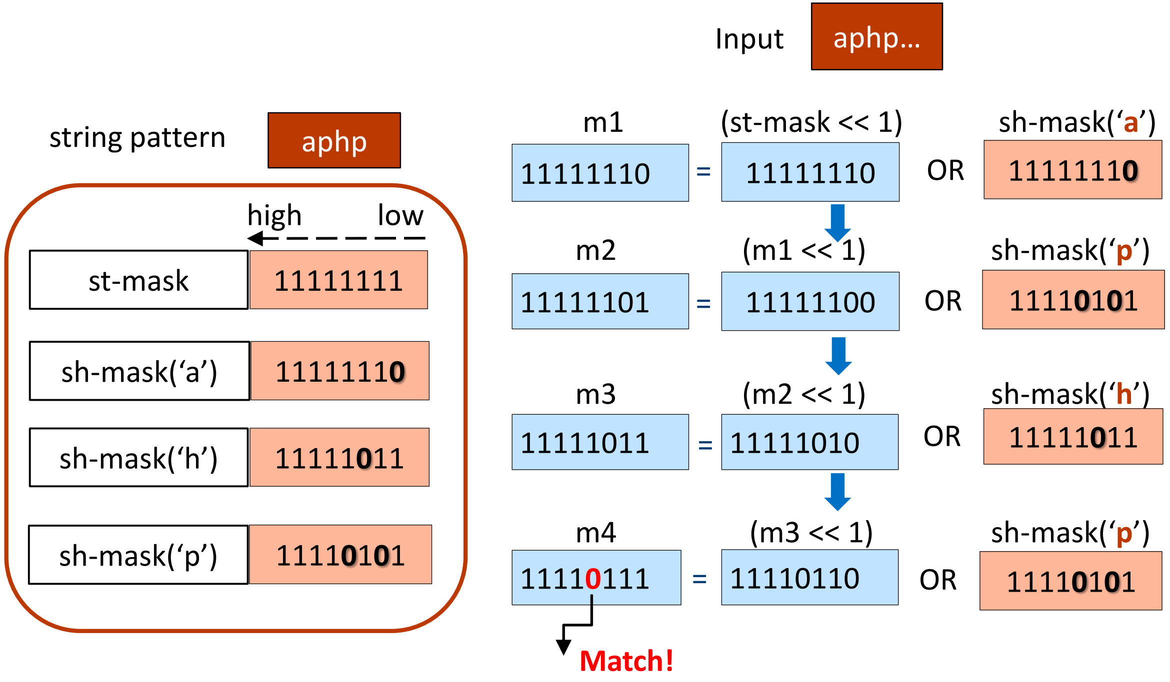
[1] E. Sitaridi, O. Polychroniou, and K. A. Ross. SIMD-accelerated regular expression matching. In Proceedings of the Workshop on Data Management on New Hardware (DaMoN), 2016

Multi-string Pattern Matching Overview

- Extended shift-or matching
  - Finds candidate input strings that are likely to match some string patterns
- Verification
  - Filters false positives with hashing
  - Confirms an exact match with string patterns with the same hash value



Shift-or String Matching



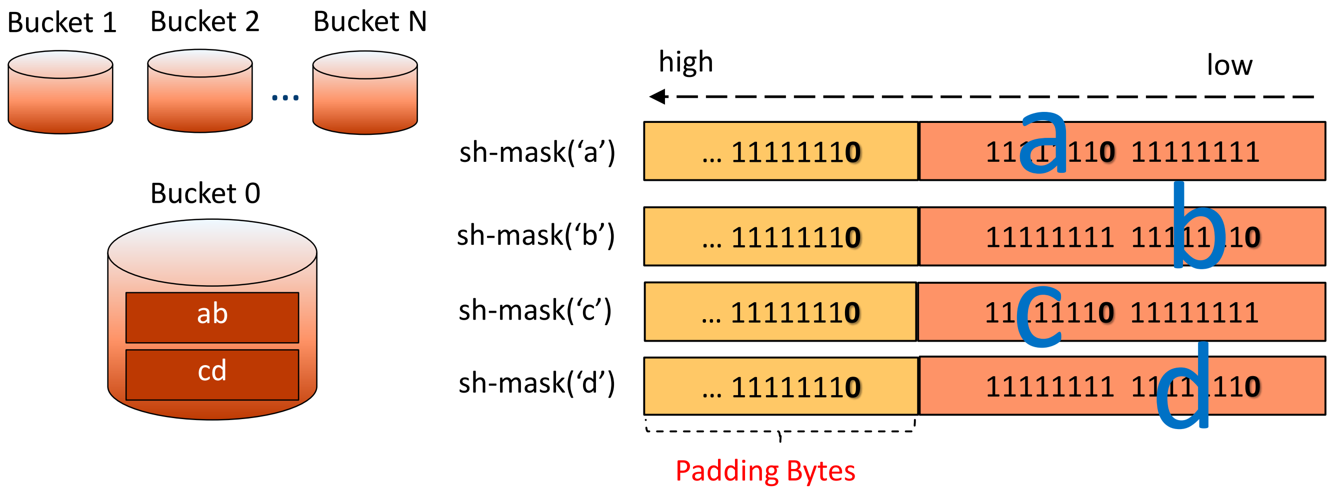
Limitations:

- Single string pattern matching only
- Cannot benefit from SIMD instructions

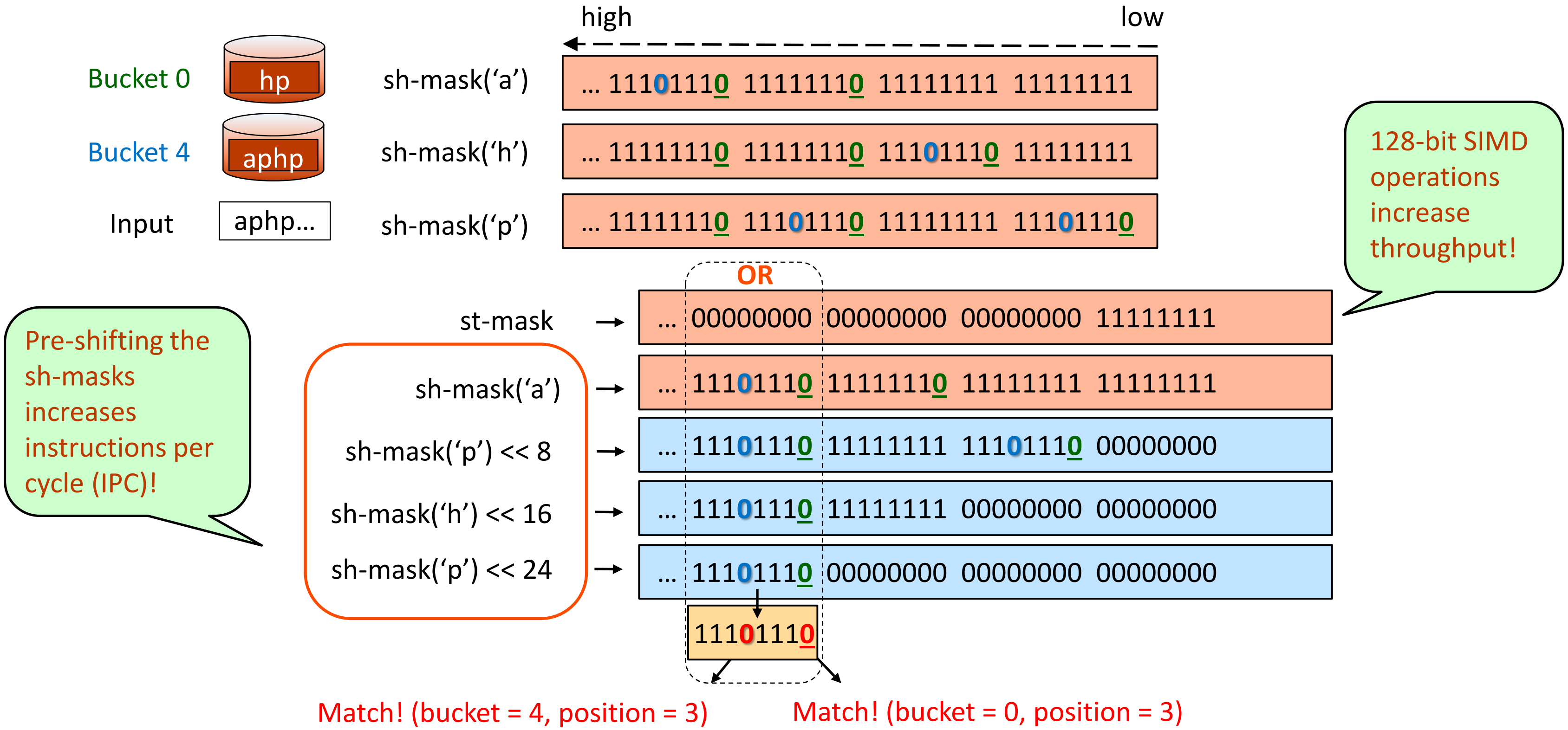
[1] Ricardo A. Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. Communications of the ACM (CACM), 35(10):74–82, 1992

Multi-string Shift-or Matching

- Pattern grouping: Groups the patterns into N buckets
- SIMD acceleration: Uses 128-bit sh-masks with 128-bit SIMD instructions (e.g., pslldq for "left shift“ and por for "or")

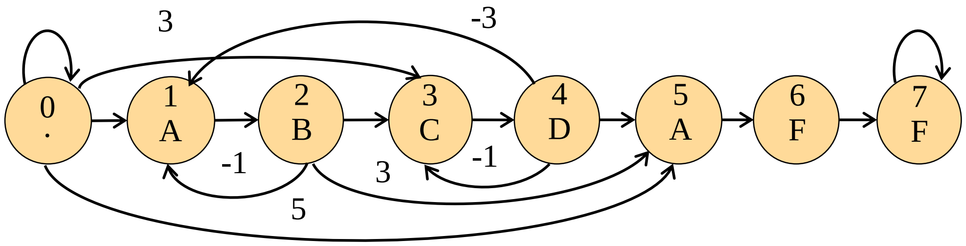


Multi-string Shift-or Matching



Bit-based NFA Matching

- Uses DFA as much as possible – but often impossible
- Classic NFA is slow - O(m) memory lookups per input character (m = # of current states)
- Represents each state with one bit in a state bit-vector
- Exploits parallel bit operations of SIMD to compute the next states



Other Subsystems

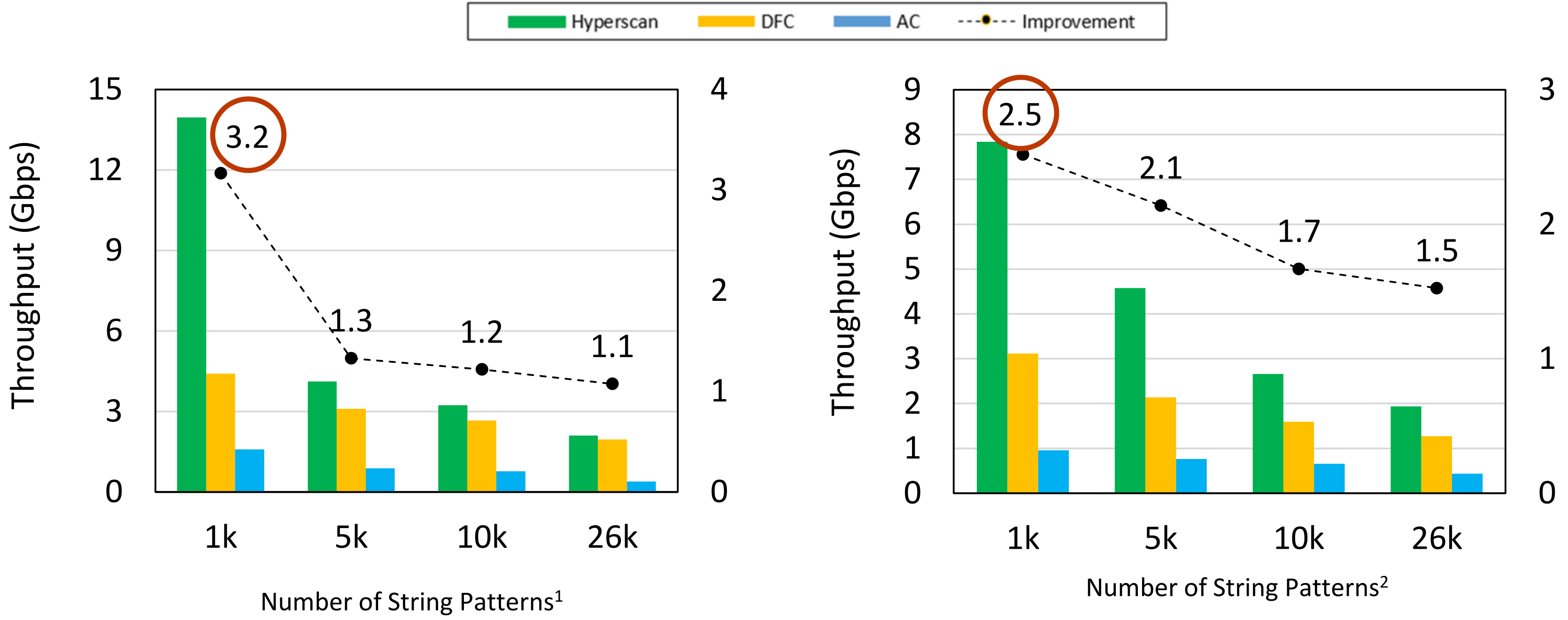
Small string-set (<80) matching
NFA and DFA cyclic state acceleration
Small-size DFA matching
Anchored pattern matching
Suppression of futile FA matching
…

Evaluation

Evaluation of Hyperscan

- Primary evaluation points:
  1. Performance of string matching and regex matching vs. state-of-the-art solutions
  2. Application-level performance improvement with Hyperscan
- Experiment setup:
  - Machine: Intel Xeon Platinum 8180 CPU @ 2.50GHz (48 GB of RAM)
    - Runs with a single core
    - GCC 5.4
  - Ruleset: Snort Talos (May 2015), Snort ET-Open 2.9.0, Suricata rulesets 4.0.4
  - Workload: random traffic, real-world web traffic

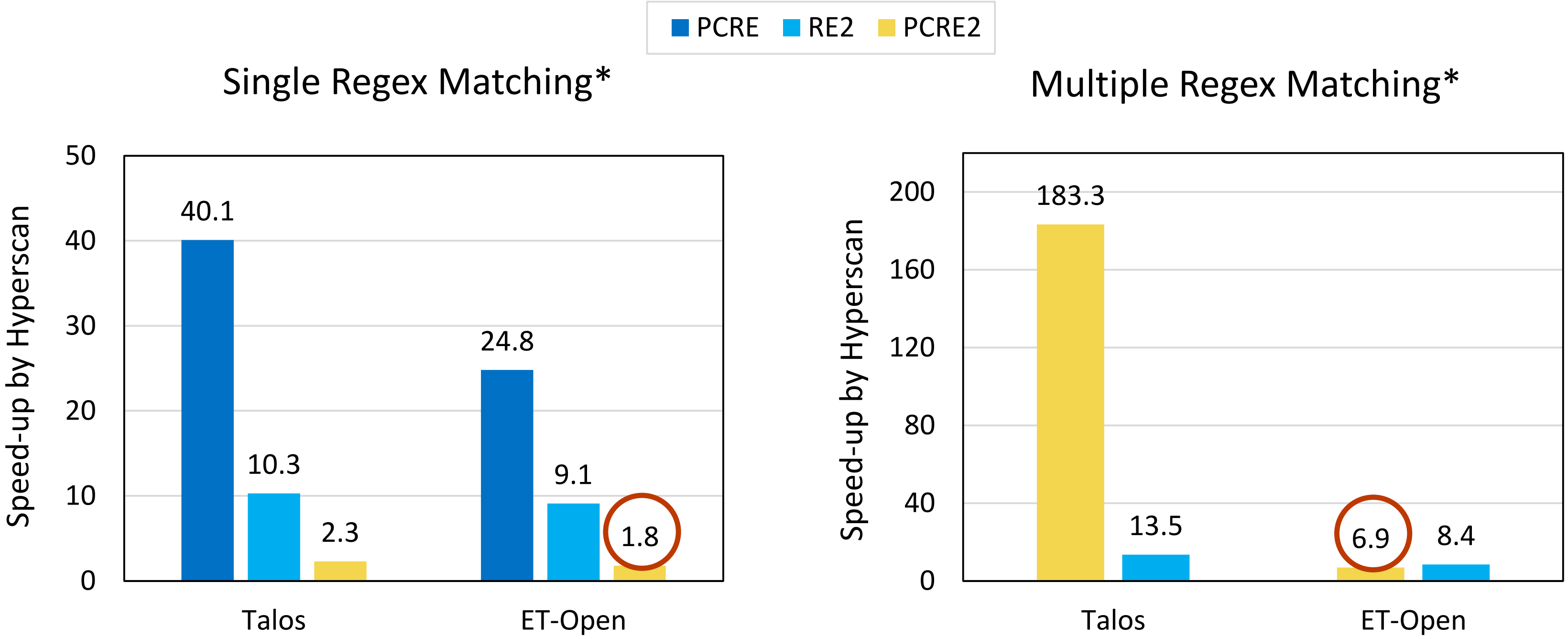
Multi-String Matching Performance with Snort ET-Open



[1] Random workload.

[2] Real web traffic trace.

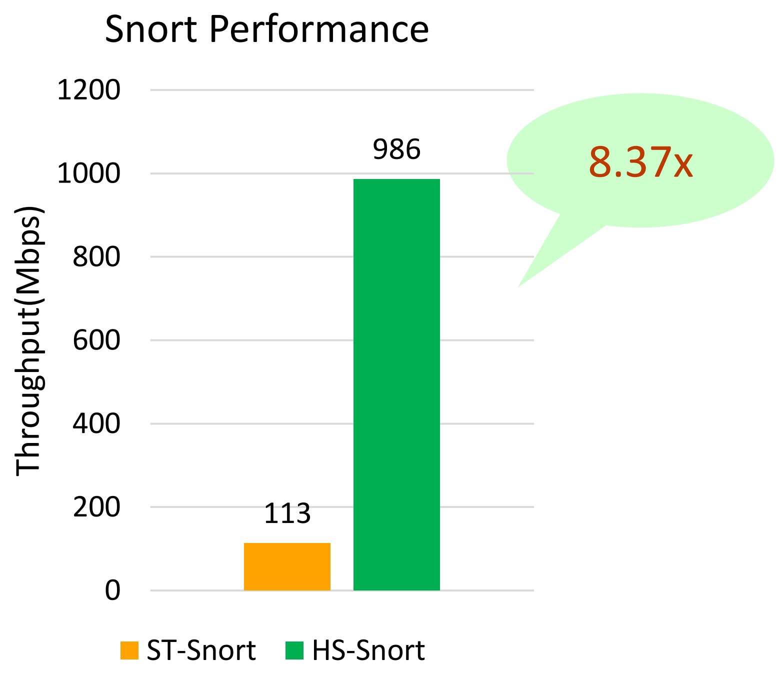
Regex Matching Performance



* Test with Snort Talos (1,300 regexes) and ET-Open (2,800 regexes) rulesets under real Web traffic trace.

Real-world DPI Application - Snort

- Stock Snort (ST-Snort) employs
  - AC for multi-string matching
  - PCRE for regex matching
  - Boyer-Moore algorithm single-string matcher
- Hyperscan-ported Snort (HS-Snort) replaced all the algorithms with Hyperscan
- Snort Talos (May 2015) with real-world web traffic



Conclusion

- Regex matching is at the core of DPI applications
- Hyperscan's performance advantage is boosted by:
  - Novel regex decomposition
  - Efficient multi-string matching and bit-based NFA implementation
- Hyperscan achieves significant performance boosts
  - 3.2x compared to DFC in multi-string matching
  - 13.5x compared to RE2 in regex matching
- Hyperscan accelerates DPI application Snort by 8.37x

Thank You

- Thanks Matt Barr, Alex Coyte and Justin Viiret for their development contribution
- Source code at https://github.com/intel/hyperscan