INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2021/CHICAGO/USA
acm In-Cooperation
sighpc
AUGUST 9–12, 2021

# *Teddy: An Efficient SIMD-based Literal Matching Engine*

Persenter:Wenjun Zhu

Kun Qiu, Harry Chang, Yang Hong, Wenjun Zhu, Xiang Wang, Baoqian Li

Intel Asia-Pacific Research & Development Ltd.

China, Shanghai

# Agenda

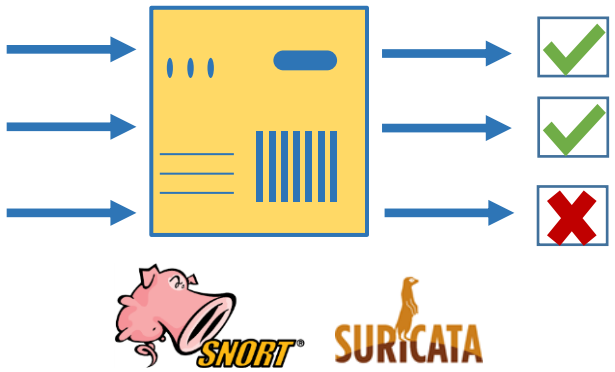
- Background
- Design
  - Grouping strategy
  - Extended Shift-or
  - SIMD Parallelizing
- Evaluation
  - Numerical
  - Theoretical
- Conclusion

# Background

Deep packet inspection (DPI) – key component of L7 traffic monitoring

1. Regular expression (regex) matching: PCRE, RE2
2. Multi-string matching (as regex's prefilter): Snort, Suricata, Hyperscan
   - Aho-corasick (AC): variant of DFA
   - FDR: extended Shift-Or algorithm (3.2x~8.8x boost over AC)

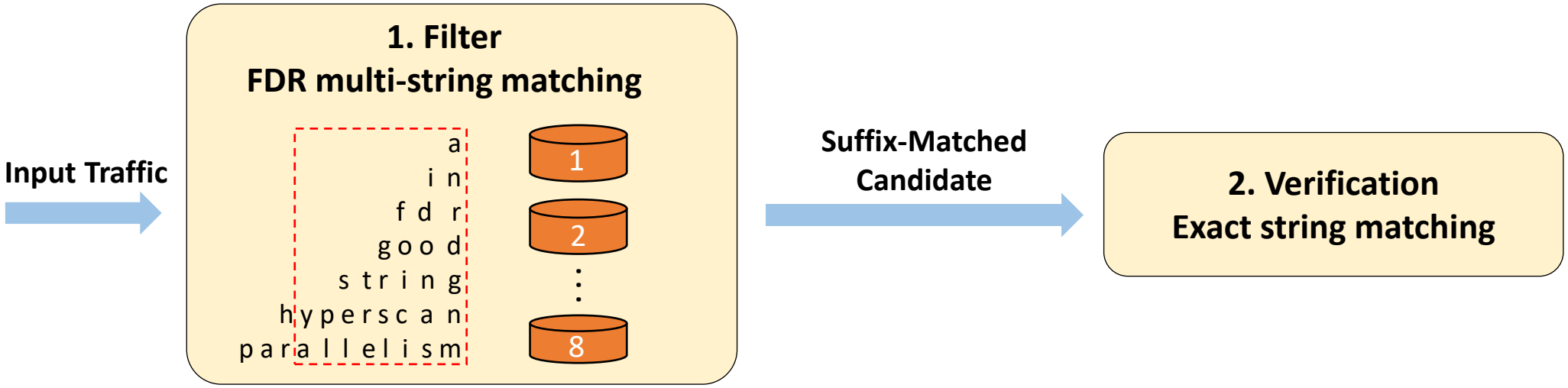
IPS/IDS




WAF


Application Identification

# Multi-string matching




Limitation of FDR[1]:

1. Grouping by string length order

   good for large scale (60 or more), bad for small scale

2. Handle 8 bytes by using SSSE3 instruction

   not fully utilize instruction width: 16 byte

| | ModSecurity[2] |
|---|---|
| Large-scale | <10% |
| Small-scale | >90% |

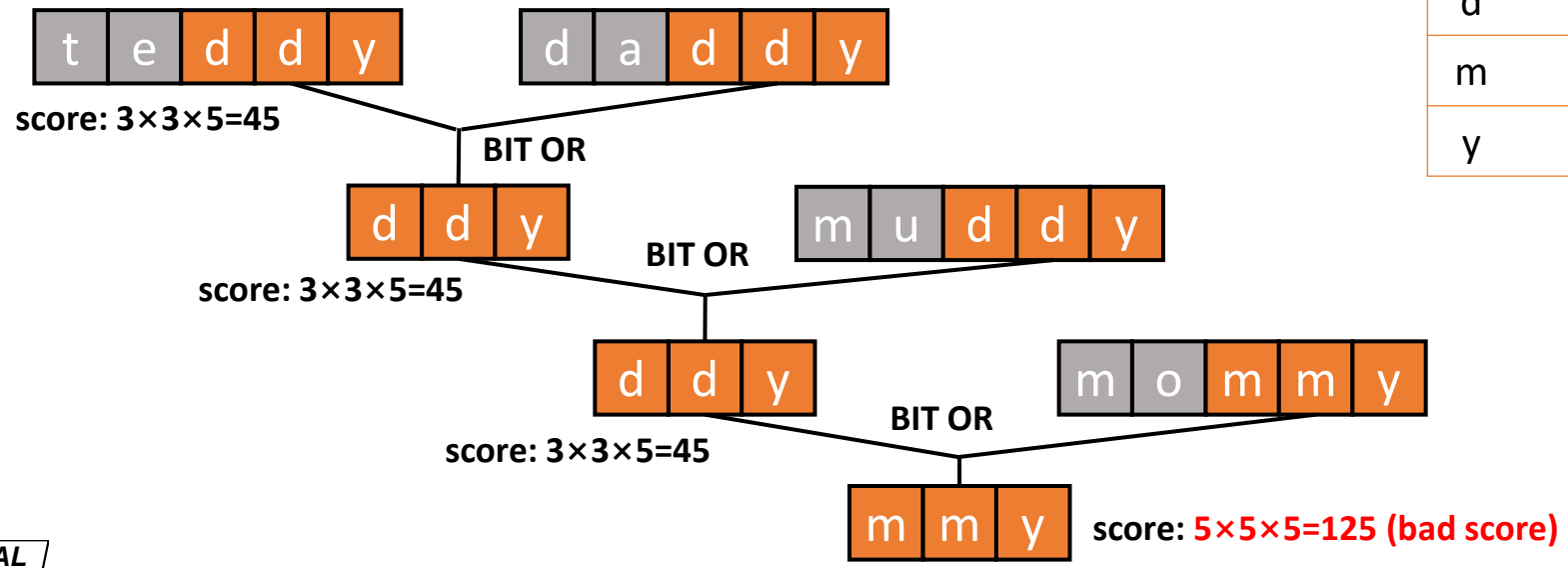
need a faster filter: Teddy

[1] https://www.usenix.org/conference/nsdi19/presentation/wang-xiang
[2] https://owasp.org/www-project-modsecurity-core-rule-set/

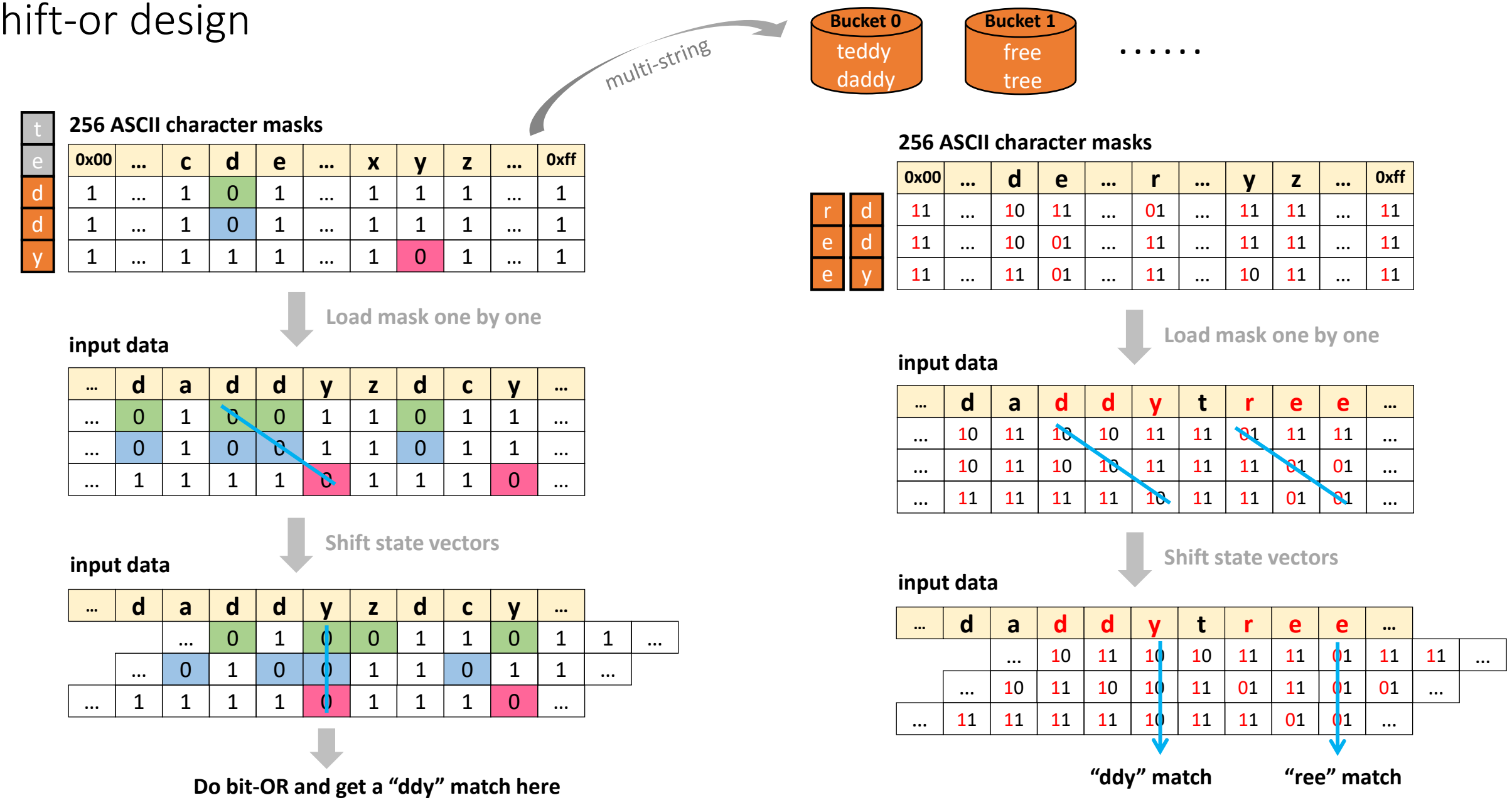
# Teddy grouping strategy

1. New grouping principles
   - Suffix similarity: for “abcde”, “efgde” is better than “abcgf”
   - Suffix length: σ ∈ {1, 2, 3, 4} (empiric value)
2. Heuristic scoring algorithm
   - Score: product of number of BIT 1 per suffix byte
   - Target: 8 buckets

| | ASCII code | #BIT 1 |
|---|---|---|
| d | 0110 0100 | 3 |
| m | 0110 1101 | 5 |
| y | 0111 1001 | 5 |



# Shift-or design



# Shift-or implementation

**256 ASCII character masks**

| | 0x00 | … | c | d | e | … | x | y | z | … | 0xff |
|---|---|---|---|---|---|---|---|---|---|---|---|
| d | 1 | … | 1 | 0 | 1 | … | 1 | 1 | 1 | … | 1 |
| d | 1 | … | 1 | 0 | 1 | … | 1 | 1 | 1 | … | 1 |
| y | 1 | … | 1 | 1 | 1 | … | 1 | 0 | 1 | … | 1 |

(256-byte vector access are not supported)

Byte decouple

| | | 0x0 | 0x1 | 0x2 | 0x3 | 0x4 | 0x5 | 0x6 | 0x7 | 0x8 | 0x9 | 0xa | 0xb | 0xc | 0xd | 0xe | 0xf |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| d 0x64 | 0x4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0x6 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| d 0x64 | 0x4 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0x6 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| y 0x79 | 0x9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| | 0x7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

(16-byte vector access are supported: PSHUFB(src, ctl) )

**16-byte input data**

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | - | - | d | d | y | - | - | - | - | - | - | - |
| low 4b | - | - | - | - | - | - | 4 | 4 | 9 | - | - | - | - | - | - | - |
| high 4b | - | - | - | - | - | - | 6 | 6 | 7 | - | - | - | - | - | - | - |
| lo_1 | - | - | - | - | - | - | 0 | 0 | 1 | - | - | - | - | - | - | - |
| hi_1 | - | - | - | - | - | - | 0 | 0 | 1 | - | - | - | - | - | - | - |
| lo_2 | - | - | - | - | - | - | 0 | 0 | 1 | - | - | - | - | - | - | - |
| hi_2 | - | - | - | - | - | - | 0 | 0 | 1 | - | - | - | - | - | - | - |
| lo_3 | - | - | - | - | - | - | 1 | 1 | 0 | - | - | - | - | - | - | - |
| hi_3 | - | - | - | - | - | - | 1 | 1 | 0 | - | - | - | - | - | - | - |

**"ddy" match**

- Handle 16 bytes simultaneously
- Fully leverage SSSE3 instruction width
- More efficient than FDR

# Theoretical evaluation: byte scanning speed

| | Operation | SIMD | Latency | Ports | Cycles | Lower | Upper[1] |
|---|---|---|---|---|---|---|---|
| *Teddy-128* | Load 16-byte input | MOVDQA | 2 | p23 | 2 | $\frac{16}{5\sigma+3}$ (byte/cycle) $\sigma \in \{1,2,3,4\}$ | $\frac{16}{6} = 2.67$ (byte/cycle) |
| | Get low 4bit vector | PAND | 1 | p015 | 1 | | |
| | Get high 4bit vector | PSLQ , PAND | 1 , 1 | p01 , p015 | 2 | | |
| | Get lo/hi masks for 3 offsets | PSHUFB × 2σ | 1 | p15 | 2σ | | |
| | Full byte matching | POR × σ | 1 | p015 | σ | | |
| | Shift | PALIGNR × (σ - 1) | 1 | p5 | (σ - 1) | | |
| | Or | POR × (σ - 1) | 1 | p015 | (σ - 1) | | |
| *FDR-128* | Load 8-byte input | PANDN × 8 | 1 | p23 | 8 | $\frac{8}{48} = 0.17$ (byte/cycle) | $\frac{8}{9} = 0.89$ (byte/cycle) |
| | Load mask per input byte | MOVQ × 8 | 3 | p015 | 24 | | |
| | Shift | PSLLDQ × 8 | 1 | p01 , p015 | 8 | | |
| | Or | POR × 8 | 1 | p15 | 8 | | |

[1] **Refer to APPENDIX A on instruction execution pipeline analysis table.**

# Numerical evaluation

1. Environment
   - Intel Xeon Icelake Engineering Sample
     - 2.2Ghz, dual sockets, 24 cores per socket, 187GB DDR4 memory
   - Ubuntu 18.04, GCC 7.5
   - Hyperscan 5.4.0

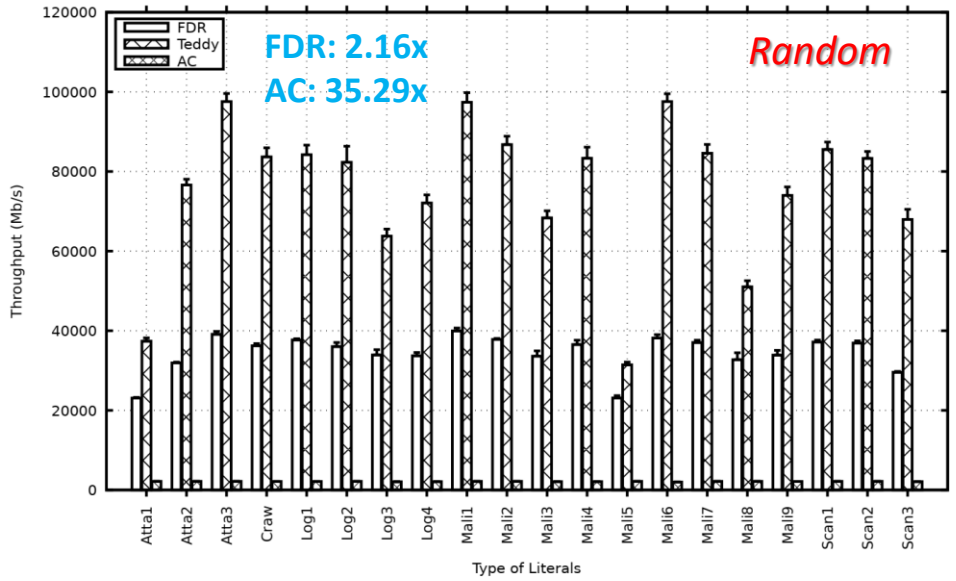
2. Dataset
   - Ruleset:

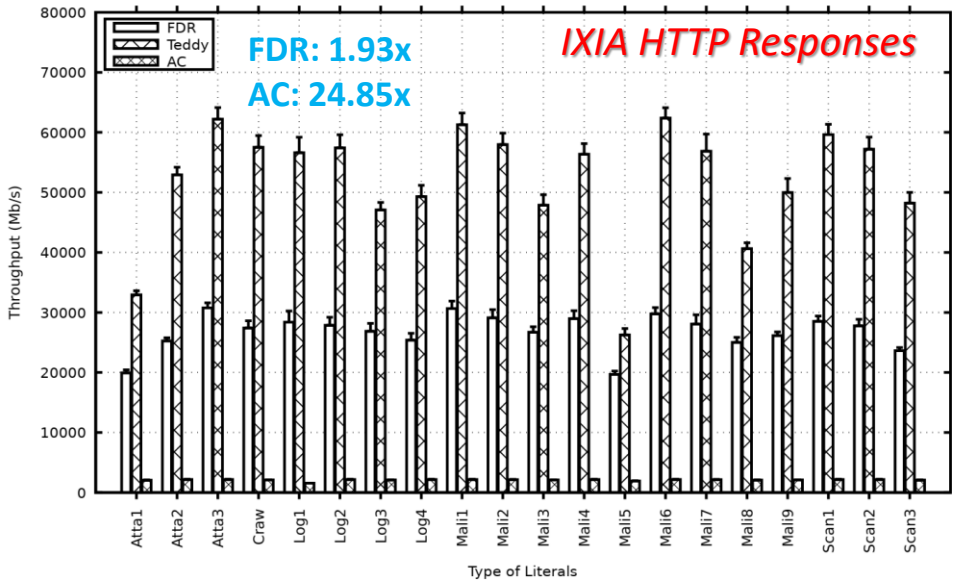
     ModSecurity
   - Input data

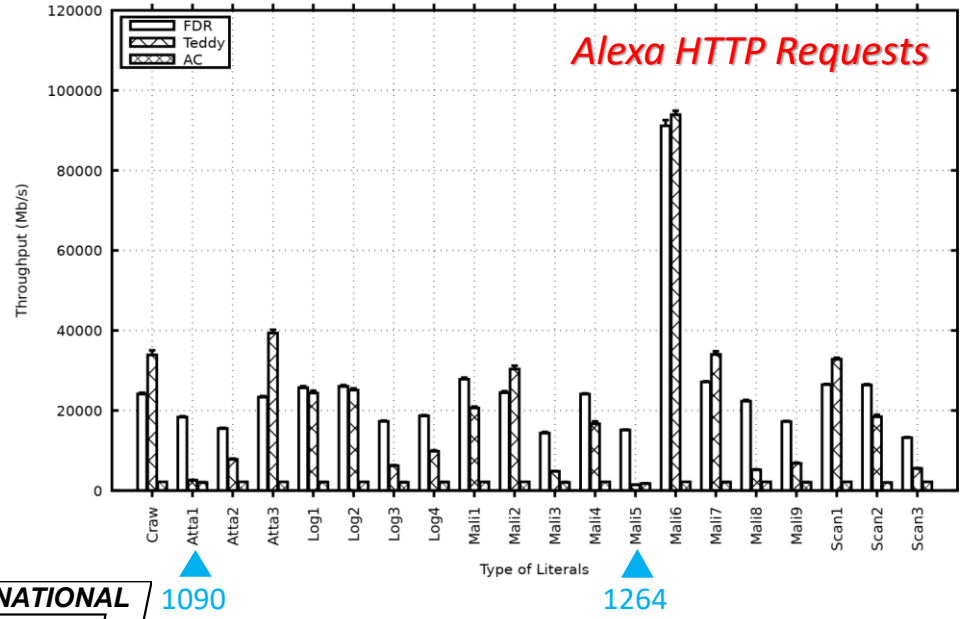
| Index | Dataset | n | Literal Type |
|---|---|---|---|
| Craw | Crawlers-user-agent | 16 | Crawlers |
| Log1 | IIS-errors | 13 | Log and Error |
| Mali1 | Java-code-leakages | 17 | Malicious Code |
| Log2 | Java-errors | 10 | Log and Error |
| Mali2 | Java-classes | 10 | Malicious Code |
| Atta1 | Lfi-os-files | 1,090 | Malware Attack |
| Mali3 | PHP-config-directives | 276 | Malicious Code |
| Log3 | PHP-errors | 276 | Log and Error |
| Mali4 | PHP-function-names-1 | 44 | Malicious Code |
| Mali5 | PHP-function-names-2 | 1,264 | Malicious Code |
| Mali6 | PHP-variables | 19 | Malicious Code |
| Atta2 | Restricted-files | 127 | Malware Attack |
| Atta3 | Restricted-upload | 17 | Malware Attack |
| Scan1 | Scanners-headers | 8 | Scanners |
| Scan2 | Scanners-urls | 17 | Scanners |
| Scan3 | Scanners-user-agents | 87 | Scanners |
| Mali7 | Scripting-user-agents | 11 | Malicious Code |
| Log4 | SQL-errors | 80 | Log and Error |
| Mali8 | UNIX-shell | 115 | Malicious Code |
| Mali9 | Windows-Powershell | 253 | Malicious Code |

| Type | Size |
|---|---|
| Random | 763 KB |
| IXIA HTTP Responses | 121 MB |
| Alexa HTTP Requests | 2.12 MB |
| IMDB(Oct. 2020) | \ |

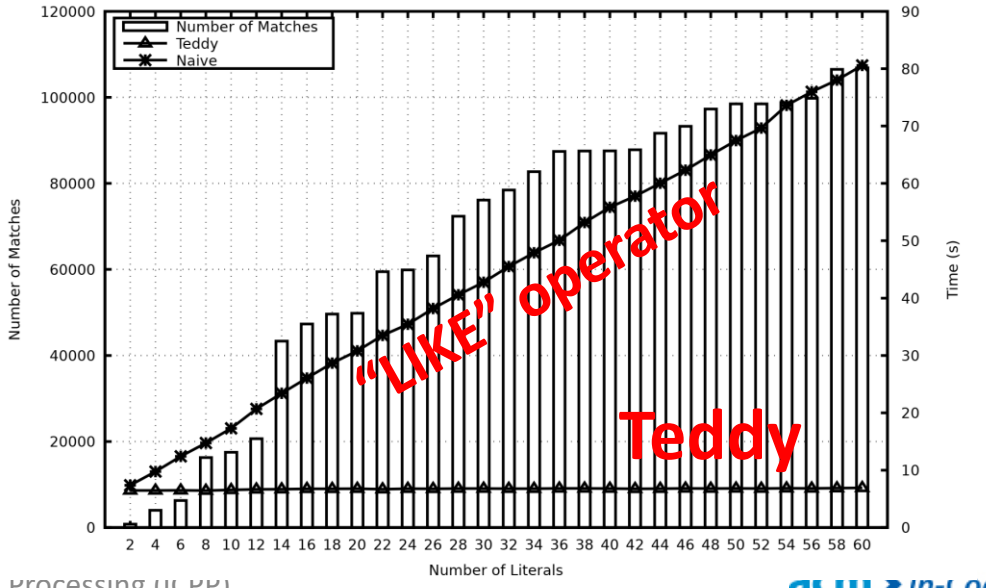
# AC vs FDR vs Teddy




## Teddy vs Database "LIKE"



# Conclusion

- Propose Teddy for efficient small-scale string matching
  1. Group strings by suffix similarity to apply to small scale case
  2. Design better parallel algorithm to fully leverage SIMD instruction width
  3. Overcome implementation restriction through byte decoupling

# Q & A