



# **HYPERSCAN IN SURICATA: STATE OF THE UNION**

Suricon 2016  
November 10, 2016  
Washington, DC

**Geoff Langdale**  
**Principal Engineer**  
**Intel Corporation**

# Legal Disclaimers

## **General Disclaimer:**

© Copyright 2016 Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, the Intel Inside logo, Intel. Experience What's Inside are trademarks of Intel. Corporation in the U.S. and/or other countries. \*Other names and brands may be claimed as the property of others.

## **Technology Disclaimer:**

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

## **Performance Disclaimers:**

Results are provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

# Welcome and Goals

- Goals:
  - Show what we've done with Suricata
  - Show where we're hitting the limits
  - Show the future of Hyperscan
- Who are you and why are you telling us about your vision?
  - Former CTO of Sensory Networks
  - Principal Engineer at Intel
  - Chief Architect of the Hyperscan software-based pattern matcher
- Hyperscan is a software library
  - Aim to be best-of-breed software regular expression matching for DPI
  - Will present Suricata integration



# History

## *Sensory Networks (2003-2013)*

- **Hardware**-based PCI-X cards  
2003-2006 for Pattern Matching
- **Vaporware**, prototypes, GPGPU,  
2007-2008

- **Software** - “Hyperscan” 2009-

## *Intel acquired Sensory October 2013*

- **Open Source Software**: 2015-
- **Suricata Patch**: 2015-
  - Uses Hyperscan for scanning in Suricata
  - Officially supported since 3.1



# Hyperscan

Hyperscan is a software-based library for regex and literal matching

- `libpcre` is the syntax: semantics slightly different
- **Multiple regular expressions**
  - We've had commercial deployments with 20K+ regular expressions
  - Tested 2M regular expressions in the lab (once...)
- **Streaming** (aka 'cross packet inspection')
- **High performance** (and scales well on multicore)
- **Low overheads** (stream state, bytecode size, compile time)

Hyperscan 2.0 -> 3.4: closed source library, in commercial use at a range of Tier 1 IPS/IDS and NGFW vendors

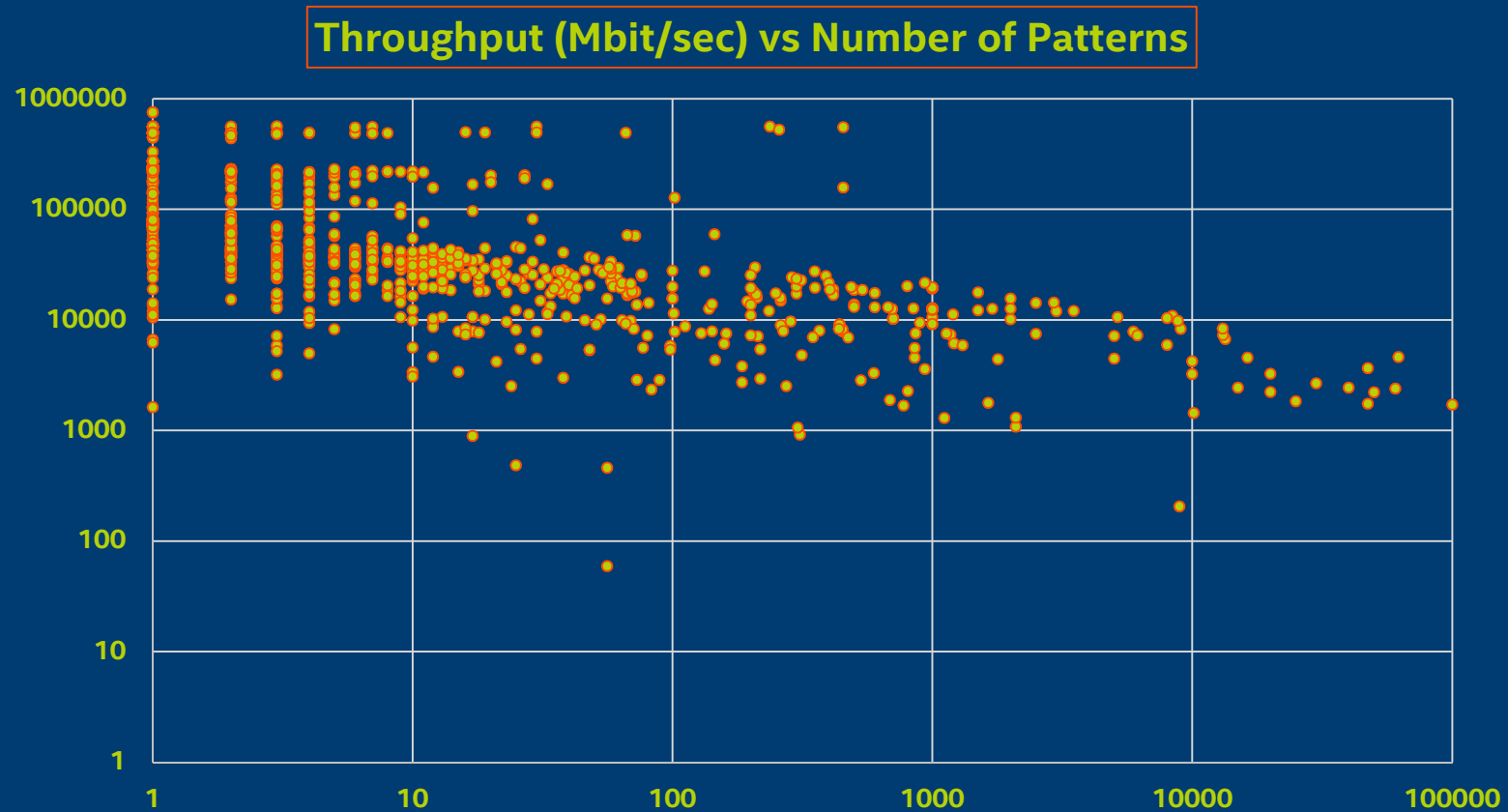
Hyperscan 4.0 -> 4.3 (current release): open source

- Release cadence around 4/year +/- a release or two

# Hyperscan Performance (General)

Plotting throughput in Mbps vs number of patterns for all our signature sets

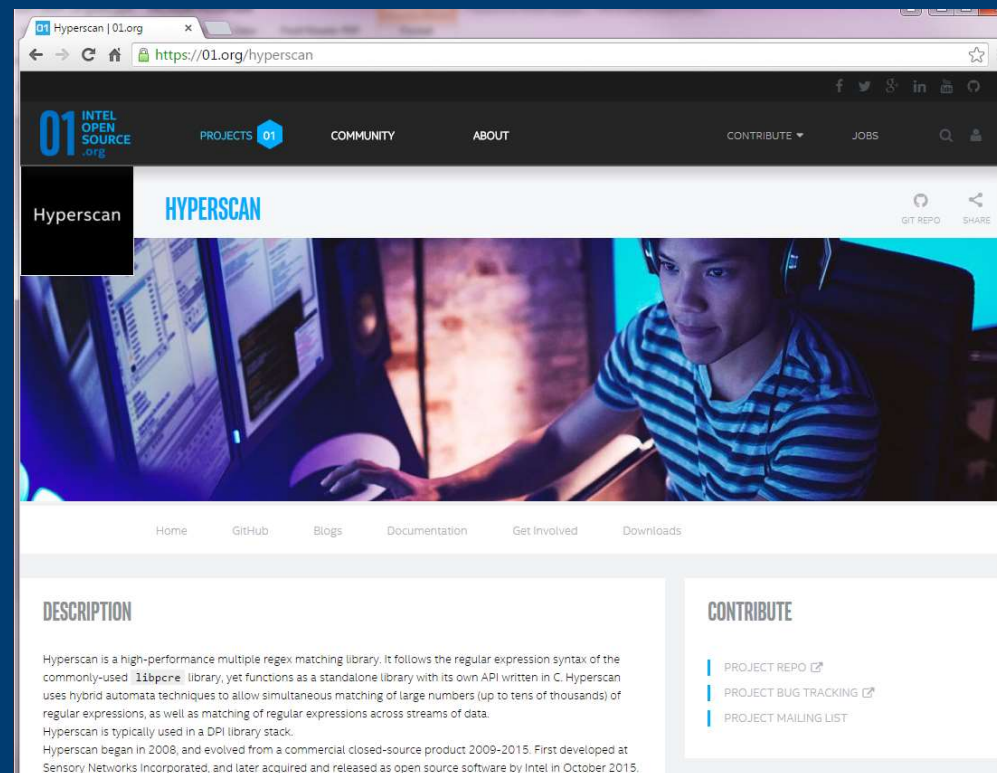
- Non-streaming, http response input, match rate < 0.125 matches per byte



# Hyperscan 4.x Series

Hyperscan 4.x has been Open Source Software for over a year

- Under 3-Clause BSD
- Get it at **[01.org/hyperscan](https://01.org/hyperscan)** or **[github.com/01org/hyperscan](https://github.com/01org/hyperscan)**



# Hyperscan 4.x Contents

## Hyperscan library

- Compiler and run-time modules
- Examples
- Simple tools and benchmarks
- Documentation
  - We use Sphinx

## Coming soon (EOY 2016)

- Sophisticated benchmarker: “hsbench”
- 4.4 release

## Platforms

- Linux: Many modern variants
- FreeBSD
- Windows
- MacOSX
- *Requires modern C++ toolchain*
  - `gcc 4.8` (*must support C++11*) or *equivalent*

# Hyperscan Open Source Strategy

## Permissive license

- Use with GPL/BSD licensed software or in a proprietary closed-source system
- We welcome and encourage contributors from outside Intel
- Active 10+ member team in Intel
- Patches provided to some key OSS projects (Snort, Suricata)
- New model is **independent development of Hyperscan integrations**
  - We will help... but we're not going to do everything
  - Benchmarking, integration, testing: big tasks in unfamiliar territory

# Next release: Hyperscan 4.4

- 3 month cycle (+/- one month)
- Optimizations for performance, stream size, bytecode size
- Preliminary AVX 512 support (for Skylake Server, not Xeon Phi)
- Approximate matching
  - Match a regex within a certain Levenstein distance
- Out of sync with release (but in same timeframe): `hsbench`
  - New benchmarker
  - Higher quality statistics
  - Hyperscan-only benchmarks
  - May integrate with other regex matchers
    - *(so I can get to be King of Hacker News for a day)*

# Suricata Integration

Last year we said we'd do 3 things:

- ... and 2 out of 3 ain't bad

Our patch to support Suricata with Hyperscan (mainline release) supports:

- Multiple literal match
- Single literal match

Experimental work only:

- Single regular expression match
  - We planned this for official patch in 2016
  - Small problem: could not demonstrate benefit



# Suricata Integration: Multiple Literal Match

- Large scale literal match a 'staple' of Hyperscan
  - Three algorithms
    - **FDR**: 'bucketed super-character shift-or' (default)
    - **Teddy**: "SIMD-based bucketed matcher" (2-72 literals)
    - **Noodle**: fast SIMD matcher for 1 literal
    - More to follow
- Just literals in this pass?
  - No: we do additional constructs like `/^.{10,50}xyz/s`
    - "Anchored" patterns
    - Avoid match 'floods'
    - Hyperscan decides whether to optimize or treat as bare literal and check later

`/^.{10}a/s` vs `/^.{10,1400}teakettle/s`

# Suricata Integration: Single Literal Match

Fast single literal matcher (**Noodle**, as in previous slide)

- Tuned SIMD match on Intel – >100+Gbps typical
- Examine 1-2 characters, do the usual PCMPQ/shift/and SIMD tricks
  - Gets faster on AVX 2, AVX 512

Use Hyperscan opportunistically

- Just literals in this pass
  - Could opportunistically add bits of adjacent Suricata rules
  - “Simple matter of programming”

# Suricata Integration: Single Regex

*Unreleased code; no performance benefit demonstrated*

Simulate behavior of backtracking regular expressions adequately

- Raise the right match in the right place!
- Implement *all* regular expressions, including back-references and zero-width asserts, etc.

Standard escape hatch: Pre-filtering (false positive but not false negative)

- HS\_MODE\_PREFILTER: replace unsupported constructs with weaker substitutes  
Example `\foo(\d)+bar\1baz\` -> `\foo(\d)+bar(\d)+baz\`  
Now accepts foo123bar456baz (false positive)

If we match, we must usually call libpcr to confirm – except when we can ...

- ... support the regex constructs 100%
- ... *know* that we got the right match
  - **Static:** Fixed width patterns **Dynamic:** When there is only one match!

# Suricata Integration – Performance results

Machine under test: Intel® Core i7 6700K CPU @ 4.0 GHz

Software versions:

- Hyperscan 4.3.1
- Suricata 3.1RC1

The following disclaimer applies to the results presented in the next slides:

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit <http://www.intel.com/performance>.

# Suricata Integration – Performance results

## Rule sets

- Emerging Threats public set (“emerging-all-20161102.rules”)
- ET Pro set (“etpro-all-20161102.rules”)

Input PCAPs: Alexa Top 100 browsing PCAP file

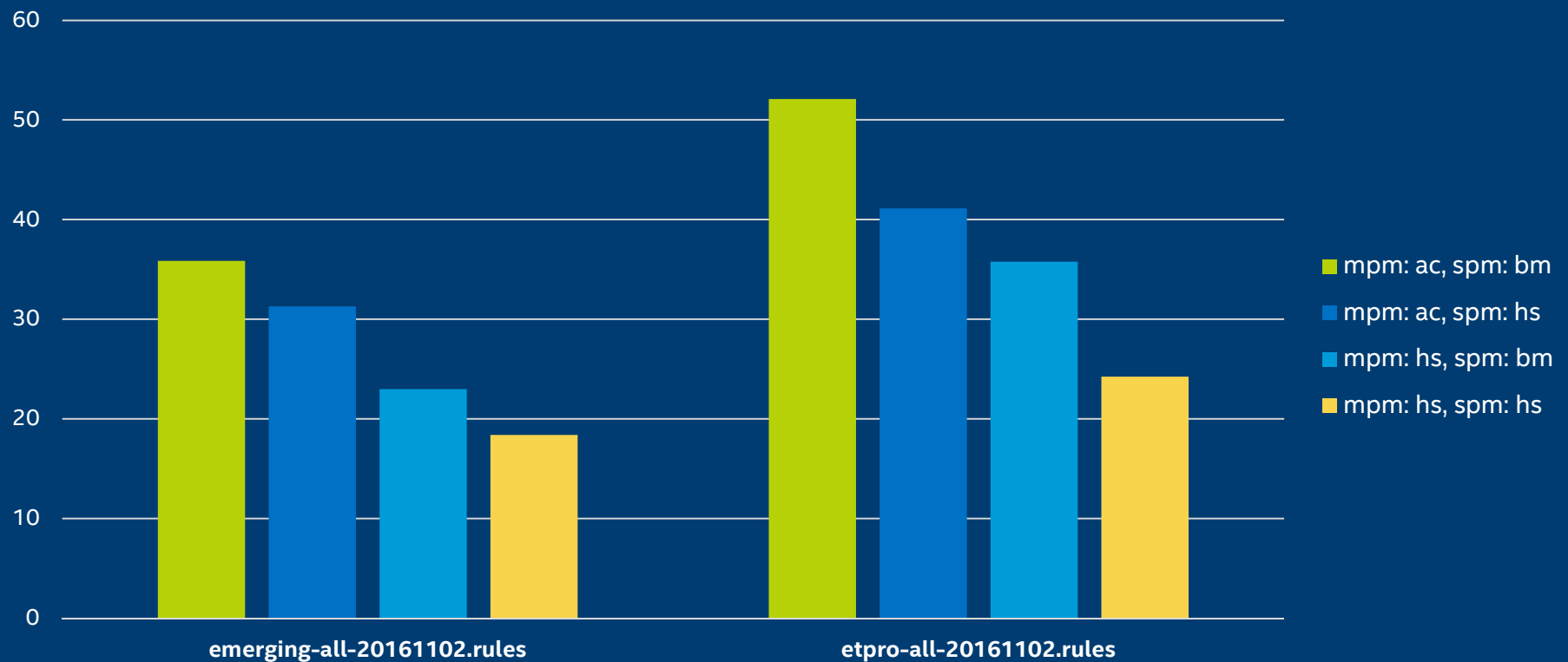
- Measurement is “all processing time” (not just scan)
  - But it’s not true end-to-end (not a network measurement)
  - If we compared head to head against Aho-Corasick the Hyperscan advantage would be bigger

# Hyperscan vs default Aho-Corasick

*Rule set without -event rule files*

**All Hyperscan vs AC+BM: Emerging Threats: 1.95x, ET Pro: 2.15x speedup**

**Measured in elapsed time in seconds for our PCAP file**



# Performance: Regular Expression “Acceleration”

Rules	PCRE elapsed time	Hyperscan elapsed time
emerging-all-20161102.rules	18.3	18.5
etpro-all-20161102.rules	24.1	24.3

## World's Saddest Table!

No benefit from Hyperscan over libpcre for 1-at-a-time-regex



# Why No Regex Speedup?

There's not much time spent in regex anyway

Single-rule time != regex time

- 2 of the top 4 rules by 'ticks' don't have regular expressions in them!
- They also get 1% slower! ☹️

`libpcre-jit` is pretty good for single patterns

Hyperscan is big and not optimized for single pattern scans

- Self-interference and interference with other Suricata code
  - i-cache and d-cache effects, TLB effects

Possible next steps:

- Complain about the benchmark?
- Cherry-pick some new cases?
- Move regex portion of workload to more appropriate place (multiple matching)

# Bigger Performance Issue: Rescanning

We rescan the same data over and over again (reassembly)

- 1100 patterns, ~9Gbps “in isolation”
- Most of our data goes through 3 Hyperscan scans at ~70Gbps, ~15Gbps and ~9Gbps each
  - But rescanning means aggregate throughput is way worse than the 5.2Gbps you would expect

Better idea:

- Hyperscan can scan in streaming mode
- Don't need to start from the beginning every time! Reuse old scan results
  - Streaming ‘literal’ matching not hard even without Hyperscan (not really any persistent state)

```
new packet
hs_scan: db=0x9f38910, buf=0x7f53a026c302, buflen=1400
hs_scan: db=0x9d03c90, buf=0x3087d68, buflen=2800
hs_scan: db=0x955d040, buf=0x7f53a043f4f0, buflen=366
hs_scan: db=0x30f7c30, buf=0x7f53a043c710, buflen=368
hs_scan: db=0x8ea7270, buf=0x7f53a04406d8, buflen=3
hs_scan: db=0x9db2460, buf=0x7f53a0496260, buflen=40960
```

```
new packet
hs_scan: db=0x9f38910, buf=0x7f53a026c302, buflen=1400
hs_scan: db=0x955d040, buf=0x7f53a043f4f0, buflen=366
hs_scan: db=0x30f7c30, buf=0x7f53a043c710, buflen=368
hs_scan: db=0x8ea7270, buf=0x7f53a04406d8, buflen=3
```

```
new packet
hs_scan: db=0x9f38910, buf=0x7f53a026c302, buflen=1420
hs_scan: db=0x9d03c90, buf=0x3087d68, buflen=2800
hs_scan: db=0x955d040, buf=0x7f53a043f4f0, buflen=366
hs_scan: db=0x30f7c30, buf=0x7f53a043c710, buflen=368
hs_scan: db=0x8ea7270, buf=0x7f53a04406d8, buflen=3
hs_scan: db=0x9db2460, buf=0x7f53a0496260, buflen=49152
```

```
new packet
hs_scan: db=0x9f38910, buf=0x7f53a026c302, buflen=1380
hs_scan: db=0x955d040, buf=0x7f53a043f4f0, buflen=366
hs_scan: db=0x30f7c30, buf=0x7f53a043c710, buflen=368
hs_scan: db=0x8ea7270, buf=0x7f53a04406d8, buflen=3
```

```
new packet
hs_scan: db=0x9f38910, buf=0x7f53a026c302, buflen=1400
hs_scan: db=0x9d03c90, buf=0x3087d68, buflen=2800
hs_scan: db=0x955d040, buf=0x7f53a043f4f0, buflen=366
hs_scan: db=0x30f7c30, buf=0x7f53a043c710, buflen=368
hs_scan: db=0x8ea7270, buf=0x7f53a04406d8, buflen=3
hs_scan: db=0x9db2460, buf=0x7f53a0498260, buflen=16384
```

```
new packet
hs_scan: db=0x9f38910, buf=0x7f53a026c302, buflen=615
hs_scan: db=0x955d040, buf=0x7f53a043f4f0, buflen=366
hs_scan: db=0x30f7c30, buf=0x7f53a043c710, buflen=368
hs_scan: db=0x8ea7270, buf=0x7f53a04406d8, buflen=3
hs_scan: db=0x9db2460, buf=0x7f53a0498260, buflen=16384
```

# Performance Summary

You can integrate a *free* (as in speech, and as in beer) library into Suricata ...

- ... and roughly double your performance

Still low-hanging fruit in terms of rescanning data

- Need to start using streaming scan capabilities in Hyperscan

The world of *regex* acceleration is murkier

- We need more regex, earlier (bulk regex)

# Our Vision: Matching In User Space Software

- Software is flexible: handle a new matching demand in days not years
- Software is adjacent to the workloads
  - Immediately prior to the regex scan: Reassembly, normalization, etc.
  - Immediate following regex match: More rules, confirmation, etc.

Example: this is a VISA card: `/^4[0-9]{12}([0-9]{3})?$/`

Run Luhn's algorithm as quick as possible (don't flood false matches back across a bus)

- Software is agile: stop matching right away if we've "seen enough"
- Software is simple to integrate: no drivers, no OS arbitration, runs bare metal or under virtualization
- Software is happier in user space: saved SIMD registers, generous stacks
  - Not a deal breaker

# Our Vision: Multiple Match Regex Scans

- One-off 'late' regex scans not very helpful for us
- Hyperscan scales better with many patterns in a rule set
- Initial “literal match” usage with Suricata doesn't have to be literals
  - Every single Hyperscan 'literal' integration is capable of doing regex

# Our Vision: Push Regex Earlier in the Process

Multiple pattern matching beats just multiple literal matching

- **Let us make the choices about which factor literals are good!**
  - Currently – patch implements anchored matching for fixed-depth strings
  - We could push a lot more regex logic earlier
- We can implement much of what's currently done in the Big Suricata Rule Interpreter Code
- Challenge: how to do this for *optional* Hyperscan usage

# Our Vision: Use Streaming

xxxxabcxxxxxxxxdefxx

xxxxab

cxxxxxxxx

xdefxx



Time (earlier writes to later writes)

- Hyperscan allows streaming regular expression: our semantics make these three cases identical
- No limits in terms of detection window (# of writes, distance in bytes)
- Start with the performance pain point: reassembly
  - Streaming matching of literals
  - Work our way around to generalized streaming usage

# Hyperscan Roadmap

Hyperscan 4.4 is coming soon (as discussed)

- New feature – Approximate Matching (allows match within “Levenstein Distance”)
  - Expensive so far – edit distance 1 is about a 39% hit on single patterns, edit distance 2 is 55%
- “Fat binaries” – can have one version of library that picks right runtime for the processor

Hyperscan 4.5 (Q1 2017)

- Further Skylake Server optimizations for AVX 512 (Purley platform)
- ACL matcher API
  - Implements DPDK ACL API
  - Looking to be 5x faster, 10x smaller
  - Branchless/loopless ACL matching with SIMD

# Hyperscan Roadmap

## Future Hyperscan mid-term (CY 2017)

- Support for logical combination of patterns
  - Ability to specify unordered AND, OR, NOT over patterns
- Support for hybrid mode (full PCRE support in non-streaming) aka “Chimera 2”
  - Could simplify patch considerably
  - Adds capturing again

# Hyperscan Non-Roadmap (Just As Important!)

- No plans to 'own' complex open source integrations
  - Design must come from project owners, not us
- No plans to support non Intel Architecture processors
  - Software engineering effort substantial
  - Testing and toolchain issues
  - If we do anything it will be a “Hyperscan Lite” edition
- No GPGPU
  - Latency too high to devices
  - Required parallelism requires packet buffering
  - Soft errors (!)
- Xeon Phi
  - Supported but no specialization yet
  - Not currently a win against equivalent conventional Xeon socket

# Conclusions

Hyperscan doubles performance (more or less; subject to workload)

**Call to action:** are you using Hyperscan? Do you have complaints?

- Sharing signatures is helpful too (we can test performance nightly)

*(complaints are less fun than admiration, but more useful)*

**Call to action:** try Hyperscan if you haven't already

Single regex matching is an awkward fit

Plenty of low-hanging fruit in Suricata scanning model

**Call to action:** fix model to use streaming

**Questions?**

# Links

Me

[geoff.langdale@intel.com](mailto:geoff.langdale@intel.com)  
[@geofflangdale](https://twitter.com/geofflangdale)

Github Code Repository:

<https://github.com/01org/hyperscan>

Hyperscan alias:

[hyperscan@intel.com](mailto:hyperscan@intel.com)

Hyperscan Open Source Software Project:

<https://01.org/hyperscan>