experience what's inside™

Hyperscan, Turbo Network Security via SW Algorithm

HYPERSCAN.IO

Legal Disclaimer

Agenda

- Hyperscan Overview
- Hyperscan Internals
- Hyperscan Future
- Case study: Snort and Suricata integrations

Intel Ingredients for Workload Optimization

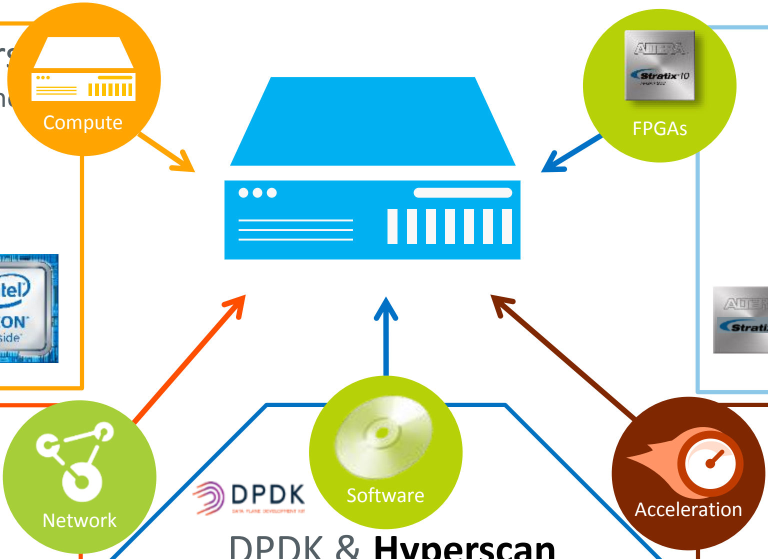
Intel® Xeon® Scalable Processors

Intel® Resource Director Technology makes the processor cache the primary destination and source of I/O data rather than main memory

Intel® Communications Chipset 89xx
Intel® C610 Series Chipset

Best-in-Class FPGA Technology

Intel's leading-edge programmable logic solutions enable users to develop customized systems based on specific application requirements.



Intel® Ethernet Controller 710 Family

10GbE, 25GbE, & 40GbE connectivity for Enterprise, Cloud and Communications

Intel Ethernet Converged Network Adapter X710/XXV710 / XL710 Family

DPDK

DPDK & **Hyperscan**

Packet Processing Software creates the foundation for NFV / SDN, server virtualization and vSwitch optimizations. Hyperscan offers application level DPI.

Intel® QuickAssist Technology

Offloads packet processing technology thereby reserving processor cycles for application and control processing

HYPERSCAN OVERVIEW

Hyperscan Overview

- Hyperscan is a regular expression matching library
  - Released by Intel under a 3-clause BSD license (permissive open source)
  - Available at https://www.hyperscan.io/
  - Software-only, IA specific (requires SSSE3 as a baseline!)
  - Multiple pattern matching and streaming (definitions to follow)
- Customers:
  - 40+ commercial customers including Tier 1 networking vendors
  - Widely used in network security solutions including WAF, IDS/IPS, etc.
  - One of the best algorithms available, and it is free

Regular Expressions

A few samples just so we know what we're talking about

- `/abc.*def/s` – "abc followed by def"
- `/\s+/s` – "one or more white space characters"
- `/foo[^\n]{400,600}bar/s` – "foo followed by 400 to 600 characters that aren't a newline, followed by bar"
- `/[^a]...[^e]...[^i]...[^o]...[^u]/s` – "something that isn't a 'a', followed by three characters, followed by something that isn't a 'e', followed by three characters, followed by something that isn't a 'i', etc.

The `libpcre` library is our standard; we use this for a semantic basis for fuzzing in automated testing

Hyperscan use case: anywhere from 1 to tens of thousands of these

Replace PCRE with Hyperscan for Performance needs.
Please find how-to on DPDK wechat blogs.

Many Dimensions of Performance Optimization

Not just raw matching speed… we also focus on:

- **Streaming and non-streaming** ("block mode") **performance**
- **Small writes** (64 byte packets == ~12 byte scanning payloads, pattern matching on individual fields – e.g. "User Agent")
- Performance under **high match rates** (1 match per byte)
- **Scaling to multiple cores/threads** (goal is close to linear)
- Pattern **compile time** (goal: keep under 10s, 1s for moderate pattern set sizes)
- **Bytecode size, stream state size, scratch space size**
- …

Software Consumption

Applications

Integrated Open Source Solutions

SNORT® SURICATA ntop

Use Cases

Firewall, IDS/IPS, SD-WAN, Email Virus, Web Security, Database, Network visibility

Language Bindings

Java python™ golang

Operating Systems

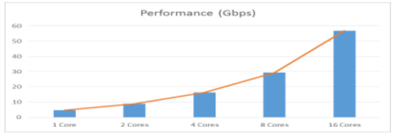
FreeBSD® fedora ubuntu debian gentoo linux™ Windows OS X

Intel Architectures

Support range from Atom to Xeon processor

intel inside ATOM … intel inside CORE i5 … intel inside XEON

Linear core scalability

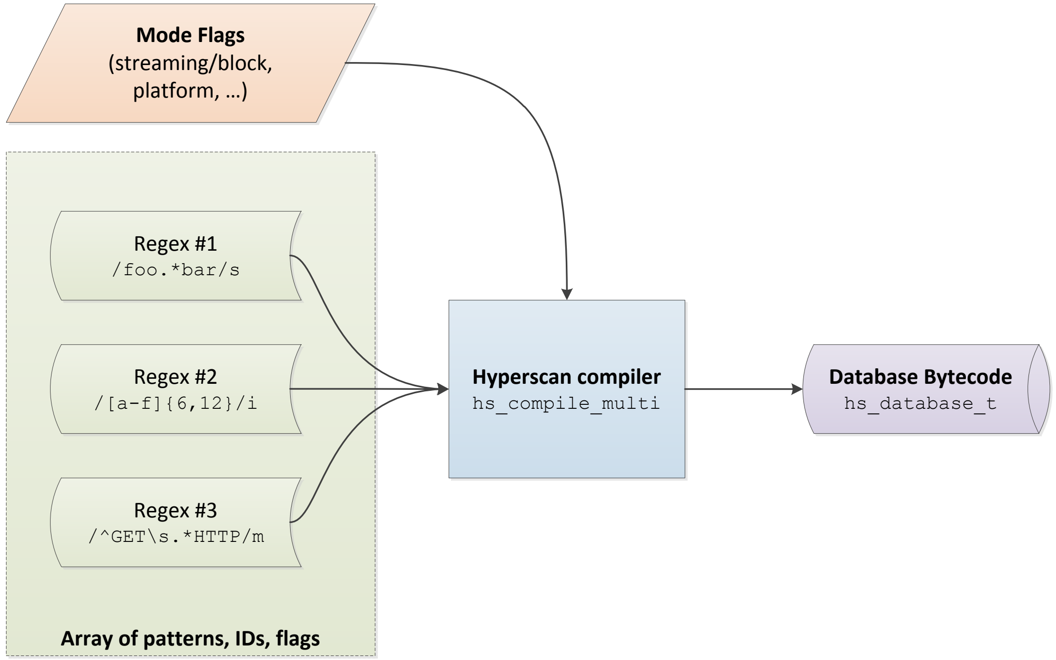


Get Started: Mode Selection

- Block Mode
  - All data to scan is obtained in a block before scanning
- Vectoring Mode
  - All data to scan is obtained and scattered into a set of blocks before scanning
- Streaming Mode
  - Not all data is obtained before scanning, instead the data is present on a stream of sequential writes
  - Can’t hold on to old data! Old stream writes are gone
  - Streaming requires only a small, fixed amount of stream state (can throw away old writes)
  - Streaming works without compromise (no fixed size windows, no limited number of writes)
  - Identical semantics regardless of stream write grouping

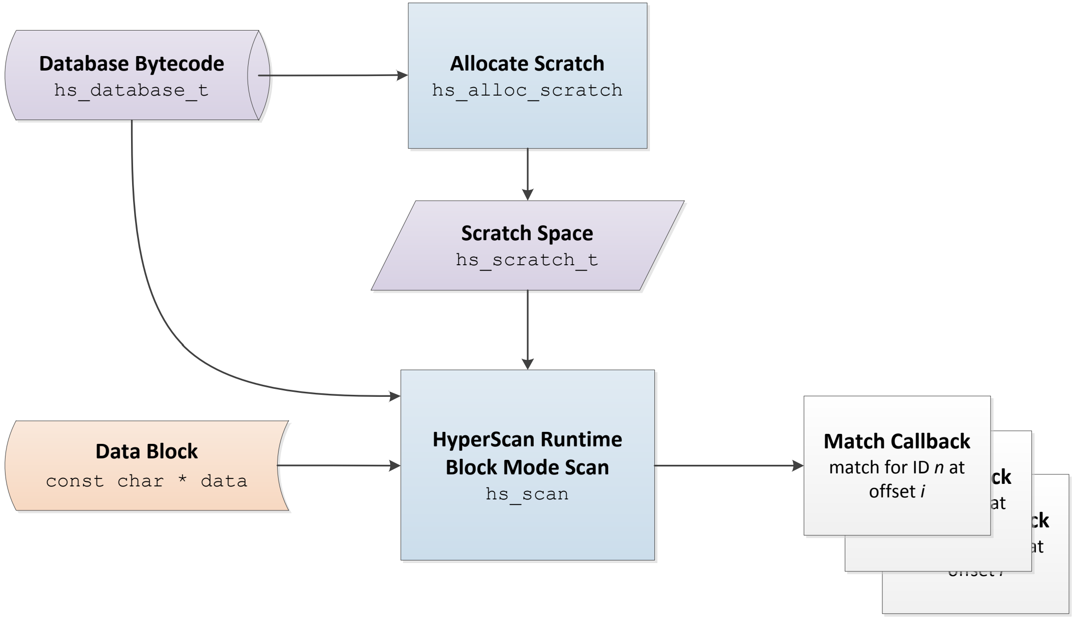
Hyperscan Operation: Compile Time

- Take an input set of patterns
- Compile to a 'bytecode' (optimizing as we go)
- Compiler:
  - C++ implementation (C API)
  - Dynamic memory allocations
  - Unpredictable compile times
  - "Bad news in advance": Unsupported patterns, large bytecode, large stream state



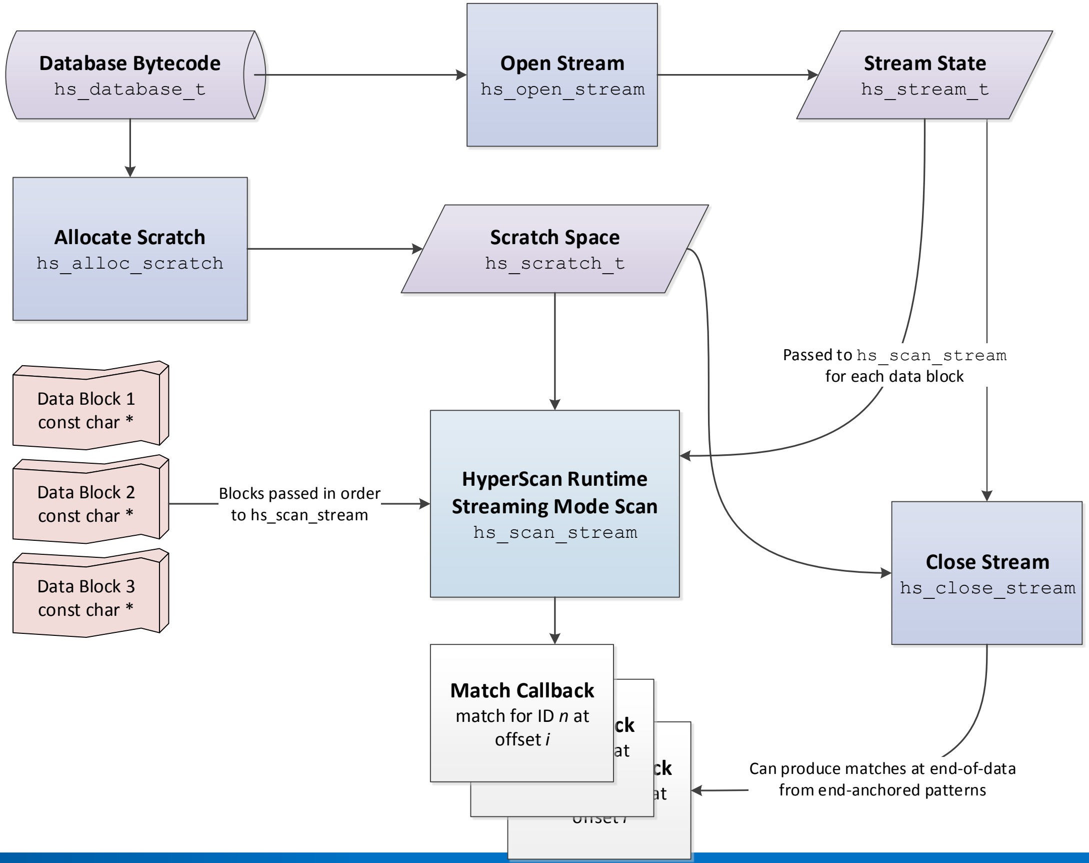
Hyperscan Operation: Run-time

- Run-time components:
  - Scratch space (working memory) – read/write
  - Compiled bytecode – read only
  - Input data
- Matches returned via callback
- Only predictable memory allocations (nothing dynamic)
- Runtime in C only



Hyperscan Operation: Run-time (streaming)

- As per block mode, but we must maintain stream state
  - Open, write, close operations
  - Also various shortcuts (reset a stream)

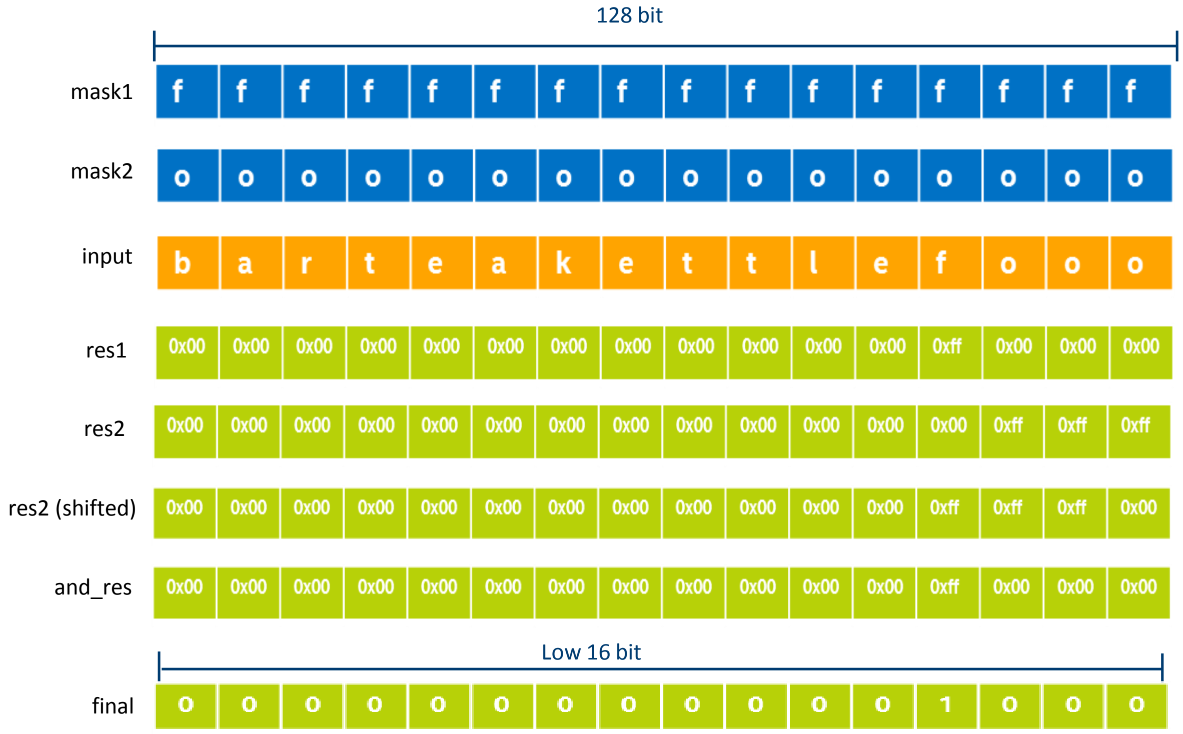


HYPERSCAN internals

SIMD Example: Literal Matching Acceleration

Accelerate searching literal "foo"

```
// Compare data with masks
res1 = _mm_cmpeq_epi8(input, mask1)
res2 = _mm_cmpeq_epi8(input, mask2)

// Shift result2 right by 1 byte
_mm_srli_si128(res2, 1)

// And result1 and result2
and_res = _mm_and_si128(res1, res2)

// move mask to get bit representation
final = _mm_movemask_epi8(and_res)

// Get trailing number of zeros
pos = __builtin_ctz32(final);
```



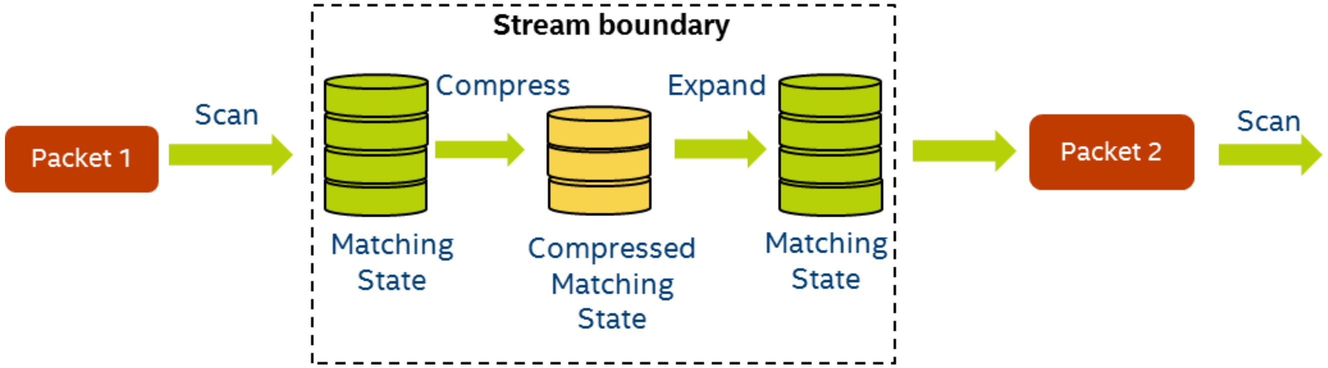
mask1	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f	f
mask2	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
input	b	a	r	t	e	a	k	e	t	t	l	e	f	o	o	o
res1	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xff	0x00	0x00	0x00
res2	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xff	0xff	0xff
res2 (shifted)	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xff	0xff	0xff	0x00
and_res	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xff	0x00	0x00	0x00
final	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Stream State Compression

Best-effort stream state compression

- New feature to reduce the memory footprint for streaming mode
- Allow stream state compression and decompression at stream boundaries
- New APIs: *hs_compress_stream()* for compression

  hs_expand_stream() for decompression

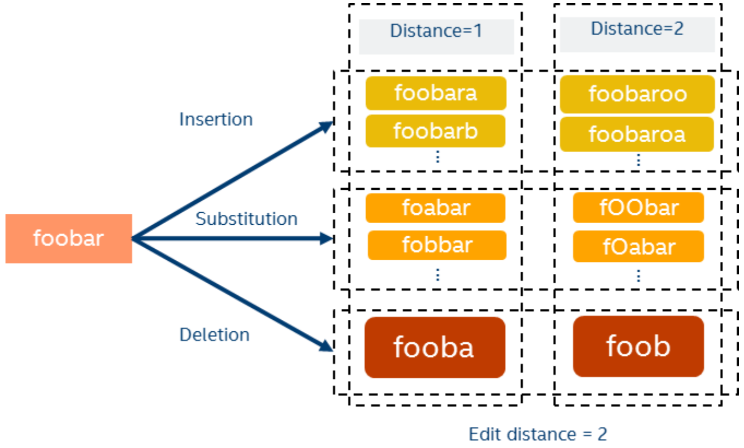


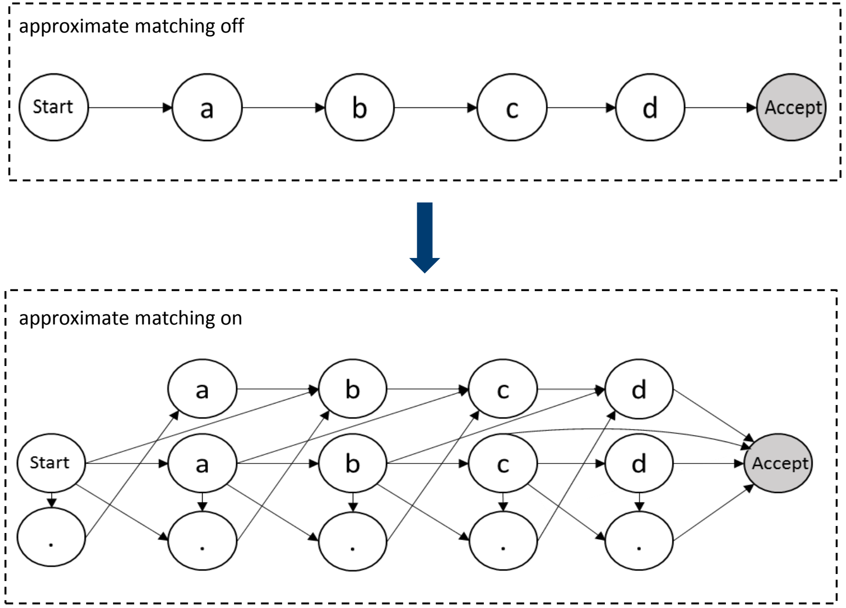
Approximate Matching

Levenshtein distance

- Allow the user to request all matches that are a given edit distance from an exact match for a pattern.

  e.g. Pattern */foo(bar)+/* within edit distance 2.
  It matches when scanned against foobar, foob0r, fobar, fooba, f0obar, and anything else that lies within edit distance 2 for the original pattern (foobar in this case).



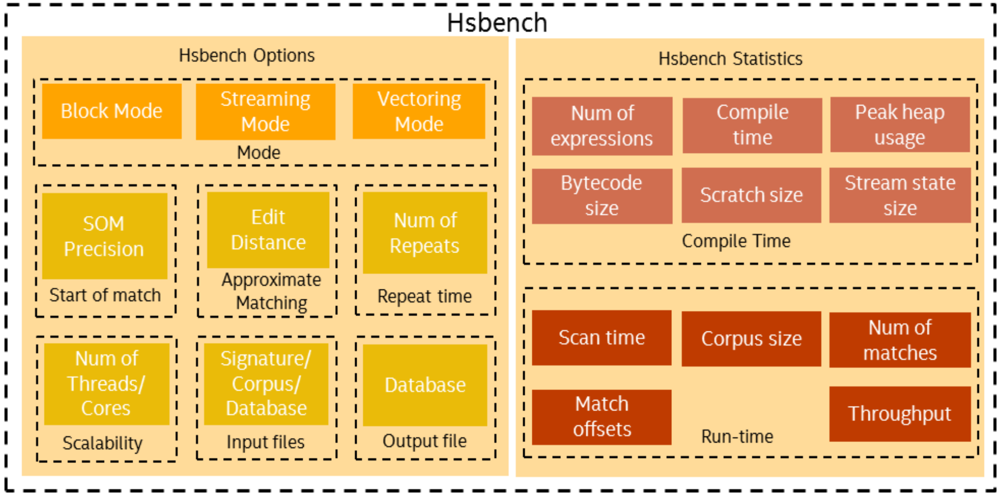


Example of conversion for approximate matching:
Pattern:/^abcd/ Edit distance = 1

Hsbench, Hyperscan Performance utility

- A new standard Hyperscan benchmarking tool - Hsbench
  - Provide an easy way to measure Hyperscan's performance for a particular set of patterns and corpus of data to be scanned.
  - Sample pattern-sets and corpora are available at https://01.org/downloads/sample-data-hyperscan-hsbench-performance-measurement

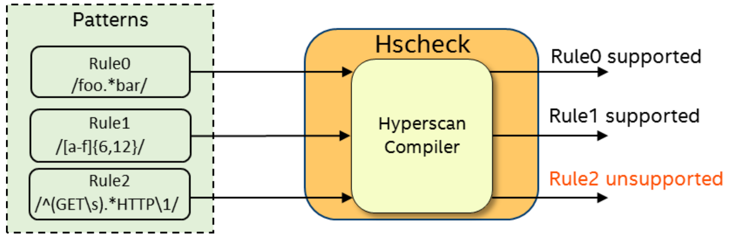
```
$ bin/hsbench -e snort_literals -c hsbench-alexa200.db -N
Signatures:          snort_literals
Hyperscan info:      Version: 4.7.0  Features: AVX2 Mode: BLOCK
Expression count:    3,116
Bytecode size:       923,384 bytes
Database CRC:        0x911ecd2f
Scratch size:        5,545 bytes
Compile time:        0.115 seconds
Peak heap usage:     195,702,784 bytes

Time spent scanning:          6.716 seconds
Corpus size:                  177,087,567 bytes (130,957 blocks)
Matches per iteration:        637,380 (3.686 matches/kilobyte)
Overall block rate:           389,958.01 blocks/sec
Mean throughput (overall):    4,218.59 Mbit/sec
Max throughput (per core):    4,630.85 Mbit/sec
```



Hscheck and Hscollider

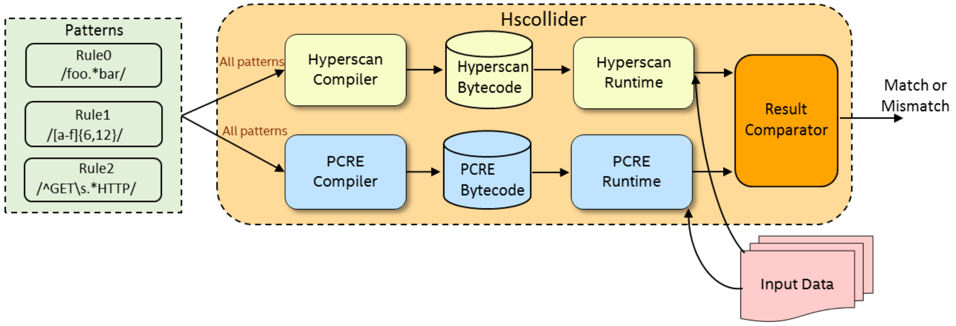
Hscheck

- Allow the user to quickly check whether Hyperscan supports a group of patterns.



Hscollider

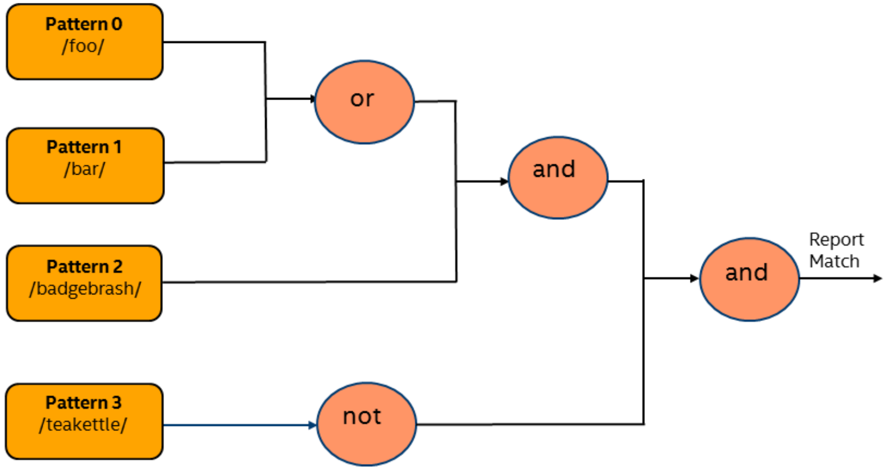
- Provide a way to verify Hyperscan's matching behavior against PCRE.



Logical combination of patterns (In Developme…)

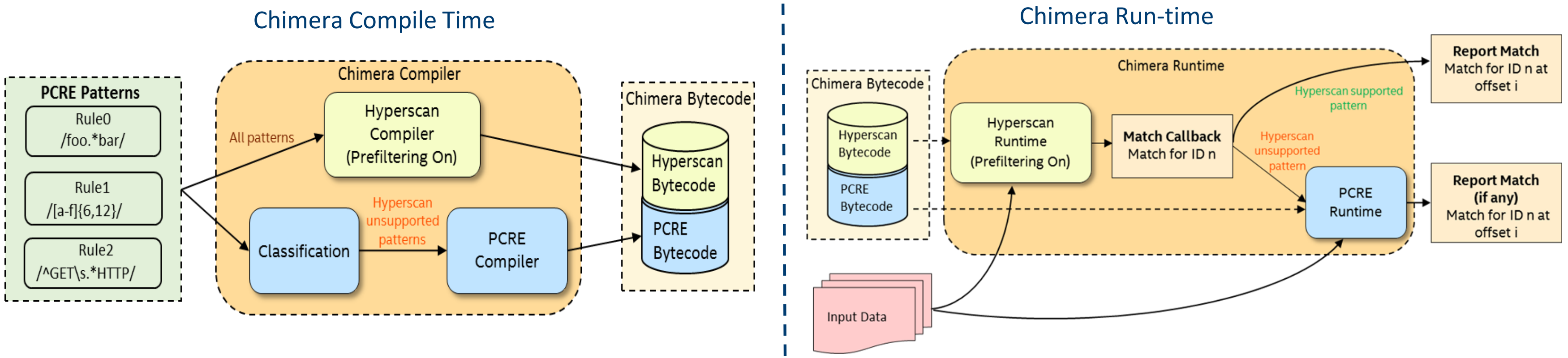
- Report matches only when find defined logical combination of patterns
- A set of patterns should match (unordered AND)
- Patterns should not match (NOT)
- Any one of a given set of patterns should match (OR)

And combinations of the above, like (pattern0 *OR* pattern1 *AND* pattern2) *AND* (*NOT* pattern3)



Chimera (In Development)

- **Chimera**: a hybrid of libpcre and Hyperscan
  - Support full libpcre syntax
  - Support multiple pattern matching at best effort
  - Take advantage of performance benefits of Hyperscan



HYPERSCAN FUTURE

Future Roadmap

Hyperscan4.7 (Shipped)

- Hscheck
- Hscollider
- Hsdump
- Bug fix for C++, fat runtime Init as share lib

Hyperscan4.8 (In development)

- Logical Combination
- Chimera
- DFA Wide State
- Windows Support

Hyperscan4.9 (in planning)

- **WAF** - ModSecurity/Hyperscan Support.
- **WAF** - Naxsi/Hyperscan Support
- Multi-Top DFA
- **AVX512 / vBMI**

Hyperscan5.0 (Directional)

- ACL Matcher
- AVX512
- vBMI

nDPI/Hyperscan Integration

New DPI/Hyperscan Solution Reference

Security Analytic in Container, Cloud Native. DPDK & FD.io Integration

Q1'18 → **Q2'18** → **Q3'18** → **Q4'18** → **2019**

* Hyperscan is supported by Fedora/Debian/Ubuntu/Gentoo/Arch Linux, FreeBSD, Homebrew(macOS), VMWARE environment already
* Hyperscan is supported in all Intel Platform from ATOM to Xeon Scalable Family, in both Virtualized & Container Environment

CASE STUDY: SNORT AND SURICATA OPEN SOURCE IDS/IPS

Snort Status

Two integrations: integration into Snort 2.9 series and Snort 3 aka Snort++

- Snort 2.9 integration (Intel)
  - Uses Hyperscan as multiple literal matcher aka “MPSE”
  - Uses Hyperscan as single literal matcher (!!)
  - Uses Hyperscan as regex matcher
  - Not upstreamed – we ship patch
- Snort 3 integration (Cisco)
  - Experimental – allows explicit regular expressions in the ‘multiple matcher’

Detection Engine Integration

Multiple literal matching code snippet:

```
typedef struct _HyperscanContext {
    hs_scratch_t *scratch;
} HyperscanContext;

typedef struct _HyperscanPm {
    hs_database_t *db;
    HyperscanContext *ctx;
    HyperscanPattern *patterns;
    …
} HyperscanPm;
```

```
typedef struct HyperscanCallbackContext_ {
    const HyperscanPm *pm;
    void *data;
    int (*match)(void *id, …);
    int num_matches;
} HyperscanCallbackContext;
```

Detection Engine Integration

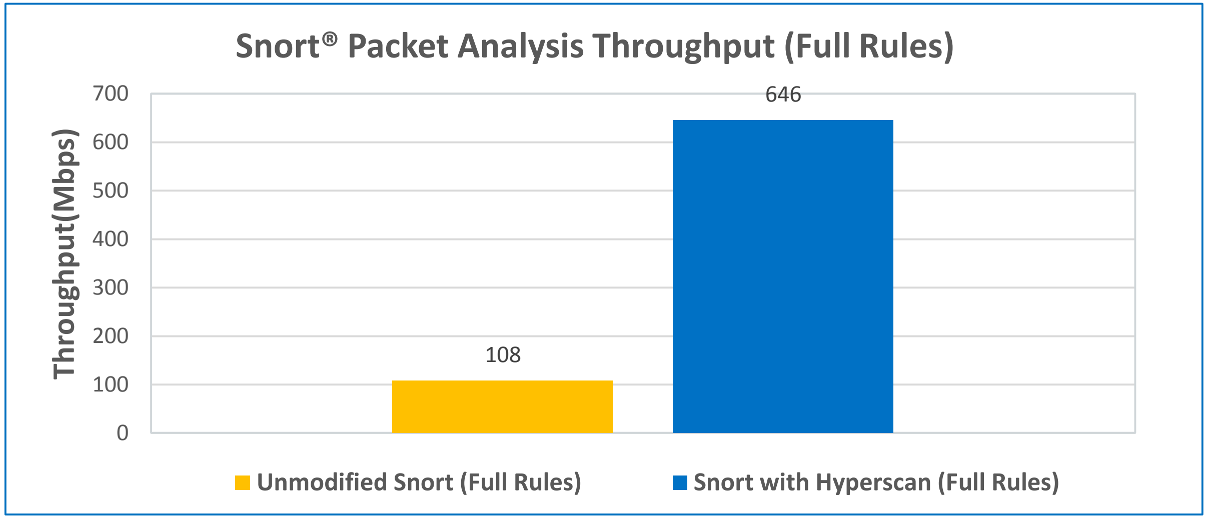
```
static int HyperscanBuild(HyperscanContext *ctx,
                          HyperscanPm *pm) {
  hs_compile_ext_multi(patterns, flags, ids, ext,
                       num_patterns,
                       HS_MODE_BLOCK,
                       NULL,
                       &(pm->db),
                       &compile_error);

  hs_alloc_scratch(pm->db,
                   &pm->ctx->scratch);
}
```

```
int HyperscanSearch(HyperscanPm *pm, …) {
  HyperscanCallbackContext ctx;
  hs_scan(pm->db, (const char *)t, tlen, 0,
          pm->ctx->scratch, onMatch, &ctx);
  return ctx.num_matches;
}
```

⬇

```
static void HyperscanCleanup(int unused,
                             void *data) {
  hs_free_scratch(contentScratch);
  contentScratch = NULL;
}
void HyperscanFree(HyperscanPm *pm) {
  hs_free_database(pm->db);
}
```

Snort Performance



Network Based/ 1C1T/ HTTP Enterprise PCAP/ 8683 Patterns
Intel(R) Xeon(R) CPU E5-2658 v4 @ 2.30GHz

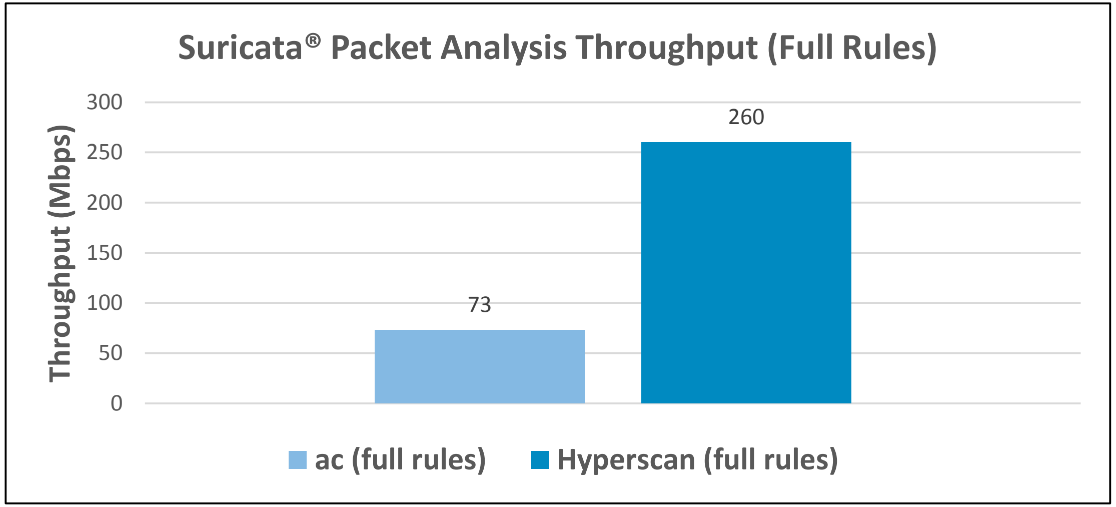
Observation: Snort with Hyperscan shows
~6x performance over Unmodified Snort
1/3 of memory footprint

Suricata Status

Integrated into Suricata mainline release

- Integrates into multiple literal matcher (MPM) – ~1000 literals (scanned at around 14Gbps in isolation)
- Integrates as a single literal matcher
- Default option on supported platforms

Suricata Performance



Network Based/ 1C1T/ HTTP Enterprise PCAP/ 13438 Signatures
Intel(R) Xeon(R) CPU E5-2695 v4 @ 2.10GHz

Observation: Suricata with Hyperscan shows **~3x** performance over Suricata with Aho-Corasick.

CONCLUSIONS AND CALL TO ACTION

Conclusion

- Solid and mature (used in large number of commercial deployments)
- Delivers substantial speedups to open source IPS/IDS systems
- Still a WIP in many senses

Intel contacts for further support

Heqing.zhu@intel.com
Xiang.w.wang@intel.com
Jerry.zhang@intel.com

experience
what's inside™