HYPERSCAN INTRODUCTION

NPG PAE: Jerry Zhang
jerry.zhang@intel.com

Legal Disclaimer

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant
Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.
No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.
The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata
are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at http://www.intel.com/ or from the OEM or retailer.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
No computer system can be absolutely secure.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Intel, Intel logo, Intel Core, Intel Inside, Intel Inside logo, Intel Itanium, Xeon, Atom, Intel ME, Intel VT-X, Intel VT-d, Intel Advanced Vector Extensions (Intel AVX), Intel AVX2, Intel AVX-512, Intel Turbo Boost Technology, Intel QuickAssist Technology, Intel Data Direct I/O Technology (Intel DDIO), Intel OP Fabric, Intel NetBurst, Intel HT Technology, Intel Volume Management Device (Intel VMD), Intel Ultra Path Interconnect (Intel UPI), Intel Speed Shift Technology, Intel Intelligent Systems Alliance (Intel ISA), Intel QuickData Technology, and Intel Platform Trust Technology (Intel PTT) are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

“Some people, when confronted with a problem, think ‘I know, I'll use regular expressions.’ Now they have two problems.”

– Jamie Zawinski, 1997 (*Old thinking about regular expressions?*)

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

Software Pattern Matching library

- Regex and fixed-string matching
- High performance, portable and easy to integrate
- Low scan latency (good small packet performance)
- Low overhead: pattern compile time, bytecode size and stream state size

Scales on Intel® processor family from Atom to Xeon

- Uses Intel SIMD instructions (from SSE2 to AVX 512) for high performance

Adopted in leading IPS/IDS engines

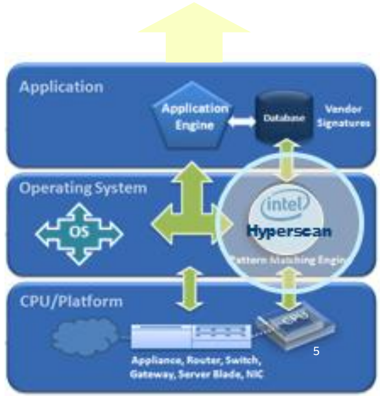
- Open source integrations available for Suricata, Snort (and Snort++) and DSPAM

Wide application

- Network Security and infrastructure
- Virtualized environments (Cloud, SDN/NFV)

Content Inspection Performance for:

- *Firewalls, IPS/IDS*
- *DPI, Anti-X*
- *Content Filtering*
- *Any application using Regex patterns*



Physical and NFV Platforms

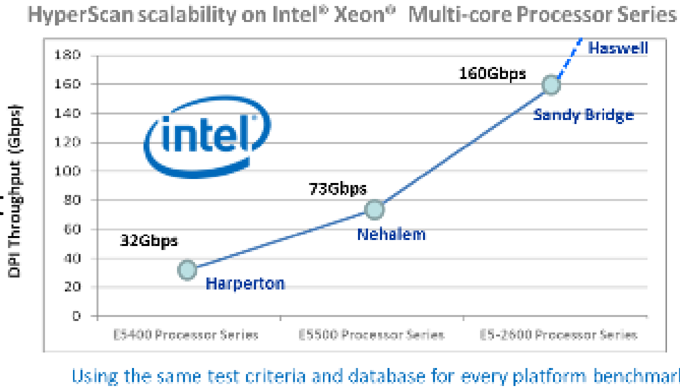
Hyperscan

Feature Rich

- Cross-compilation (cross-compile across architectures)
- Block mode scanning
- Streaming mode (allows matching across multiple 'data' writes to a stream)
- Start/End of Match, Ordering
- Vectoring (multiple block data writes, all present at the same time, different memory locations)

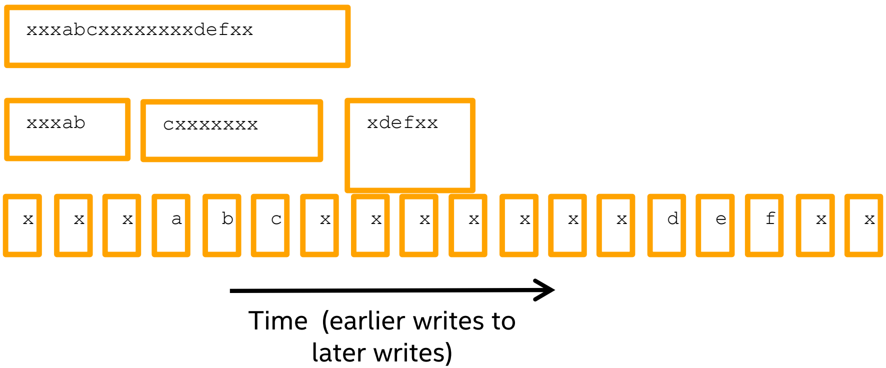
Robust, expressive feature

- Use a wide range of regex constructs
- No 'combinational explosion', no backtracking
- Use large numbers of constructs such as .* or or [^>]* without any problems



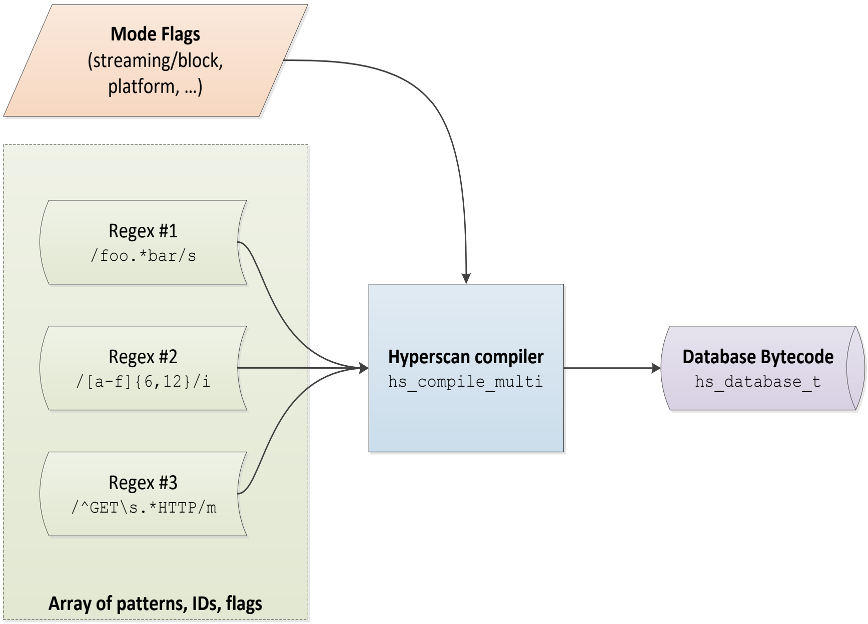
Streaming

- Streaming operation means scan can work “as if” all data was present even 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 – all three sequences below are the same in terms of expected matches:



Hyperscan Operation: Compiler

- 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



Compiler API

```
hs_error_t
hs_compile_multi(const char * const *expressions,
                 const unsigned int *flags,const unsigned int *ids,
                 unsigned int elements, unsigned int mode,
                 const hs_platform_info_t *platform,
                 hs_database_t **db,
                 hs_compile_error_t **error);
```

expressions, flags and ids are all arrays of length elements, each element describing a pattern.

mode contains database-global flags: block mode, streaming mode, platform information.

Errors result in a return value other than HS_SUCCESS and a more detailed message in the **hs_compile_error_t** structure.

Compiler API Pattern Flags(1)

HS_FLAG_CASELESS: case-insensitively match alphabetic characters

HS_FLAG_DOTALL: interpret '.' as matching any character, rather than any character except newline

HS_FLAG_MULTILINE: interpret '^' and '$' as matching at newline characters as well as at start and end of data

HS_FLAG_SINGLEMATCH: produce at most one match from this pattern

HS_FLAG_UTF-8: operate in UTF-8 mode

HS_FLAG_UCP: interpret character classes with UCP properties

Compiler API Pattern Flags(2)

HS_FLAG_ERROREOD: raise an error at compile-time if the expression can match at end-of-data (e.g. /foobar$/)

HS_FLAG_ALLOWEMPTY: allow expressions that can match against empty buffers (e.g. /.*/)

HS_FLAG_ORDERED: guarantee that matches from this pattern will be returned in order

HS_FLAG_SOM_LEFTMOST: report leftmost start of match

HS_FLAG_PREFILTER : Compile pattern in prefiltering mode.

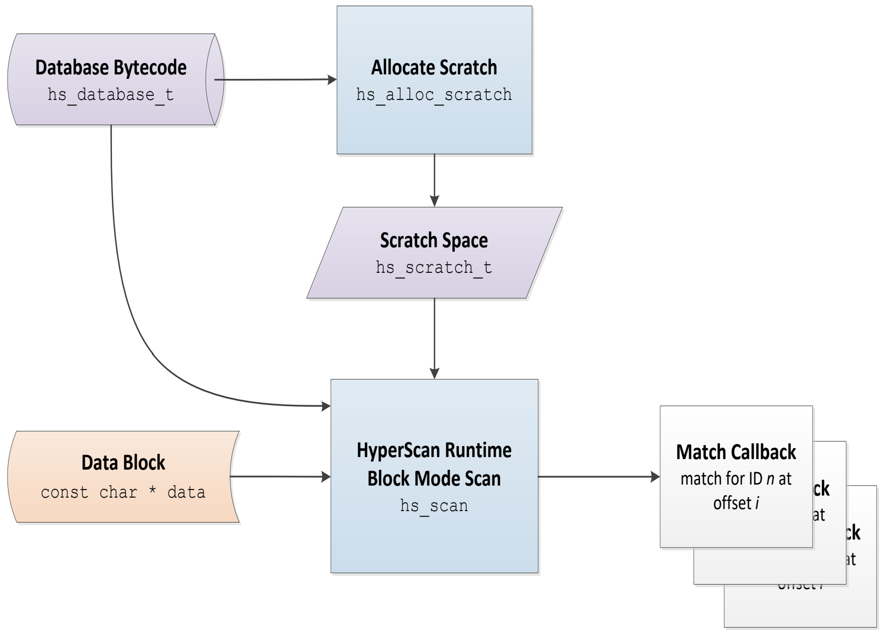
Compiler API Mode Flags and Platform

Mode: HS_MODE BLOCK, HS_MODE_STREAM, HS_MODE_VECTORED

Platform: If not NULL, the platform structure is used to determine the target platform. If NULL, a database suitable for running on the current host platform

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



Block Mode API

```
hs_error_t hs_scan(const hs_database_t *db, const char *data,
        unsigned int length, unsigned int flags, hs_scratch_t *scratch,
        match_event_handler onEvent, void *ctx);
```

Simplest scanning mode: scans length bytes of data against the databa-- db.

Matches result in the function onEvent being called, with the context pointer ctx.

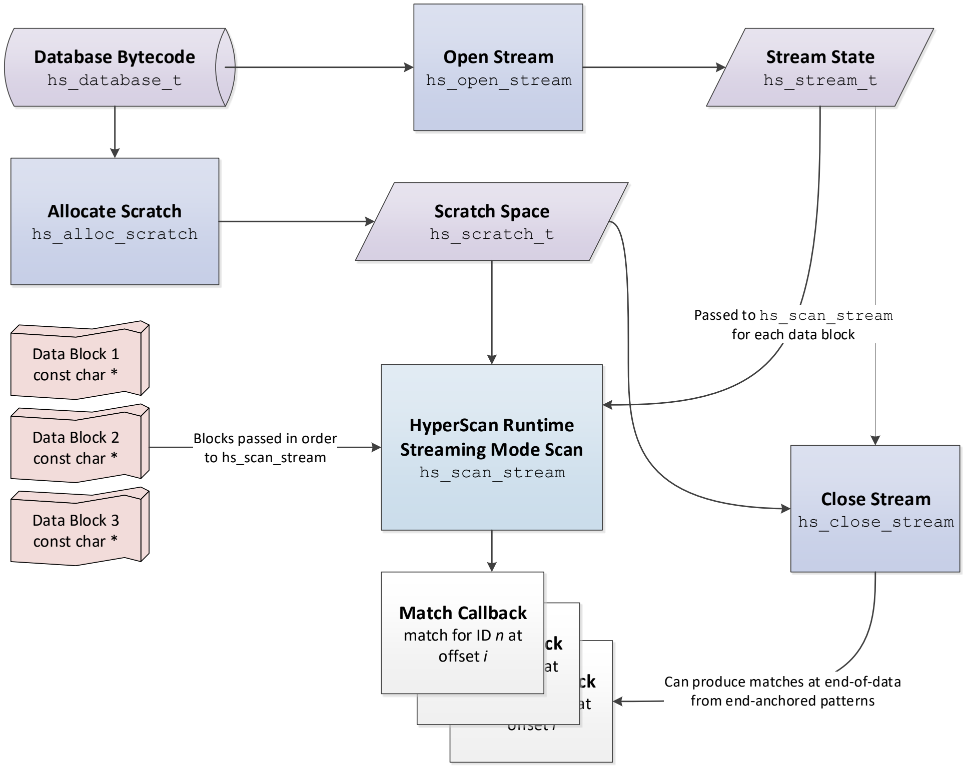
Returns HS_SUCCESS on success

HS_SCAN_TERMINATED if the user callback requested termination

Other errors for invalid arguments

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)



Stream Mode API

```
hs_error_t hs_open_stream(const hs_database_t *db,
      match_event_handler onEvent,
      unsigned int flags, void *ctx, hs_stream_t **stream);
hs_error_t hs_close_stream(hs_stream_t *id, hs_scratch_t *scratch);
hs_error_t hs_scan_stream(hs_stream_t *id, const char *data,
      unsigned int length, unsigned int flags, hs_scratch_t *scratch);
```

Persistent state is stored in an **hs_stream_t** allocated/freed by the open and close calls.

True streaming, not a rescanning approach: matches can span any distance

The amount of stream state is fixed and is dependent on the structure of the patterns.

hs_close_stream produces matches for EOD-anchored patterns, like /foobar$/

Scratch Space API

```
hs_error_t hs_alloc_scratch(const hs_database_t *db,
                            hs_scratch_t **scratch);
hs_error_t hs_clone_scratch(const hs_scratch_t *src,
                            hs_scratch_t **dest);
void hs_free_scratch(hs_scratch_t *scratch);
size_t hs_scratch_size(const hs_scratch_t *scratch);
```

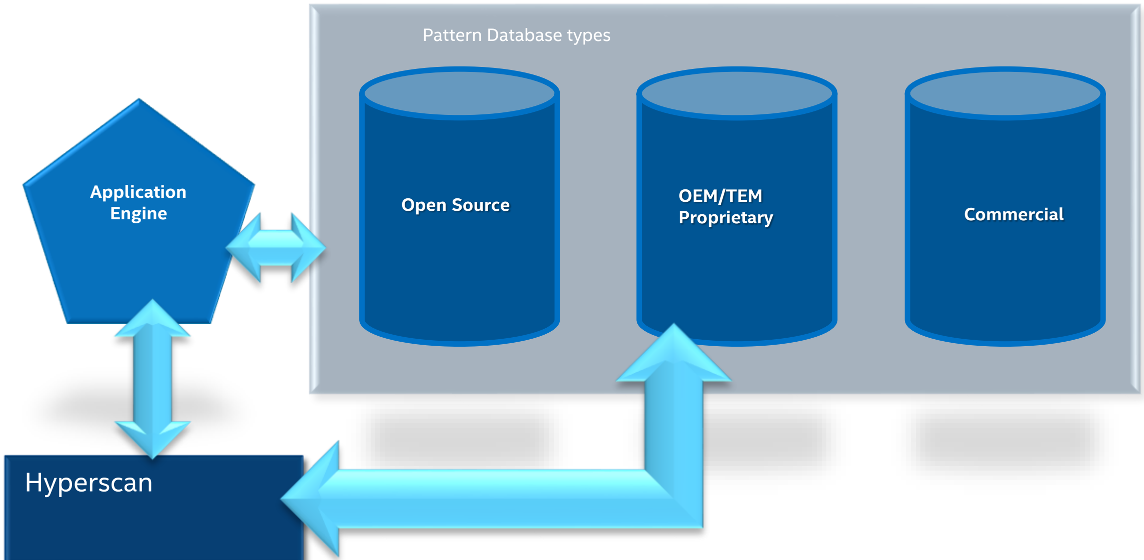
Runtime operations that scan data need a scratch region, used to store temporary data.

This can be called multiple times with several databases, which will result in a scratch region that can be used with any of them.

Every thread of control needs its own scratch, though they can share a (read-only) database.

Pattern Sets (Signatures)

- Hyperscan is designed to support most security signatures
- For 'exotic' signatures, Hyperscan can be adapted



NOTE: Intel does not develop or supply security signatures

Integration with Industry leading IPS/IDS engines

Snort https://www.snort.org/

- Most widely deployed IPS/IDS in the industry;
- Cisco (Sourcefire) owns the GPL
- Hyperscan patch for 2.9 now available

Suricata http://suricata-ids.org/

- Contender to Snort although much smaller community
- Delivers greater performances
  - Multi-threaded architecture
- Fast-growing adoption in the industry
- Hyperscan is upstreamed in Suricata 3.1

Parameter	Snort	Suricata
Developer	Sourcefire, Inc.	Open Information Security Foundation (OISF)
Availability	Since 1998	Since 2009
Coded Language	C	C
Operating System	Cross-platform	Cross-platform
Stable Release	2.9.6.2 (16 July 2014)	2.0.3 (8 August 2014)
Threads	Single-threaded	Multi-threaded
IPv6 Support	Yes	Yes
Snort (VRT) Rules Support	Yes	Yes
Emerging Threats Rules Support	Yes	Yes
Logging Format	Unified2	Unified2
Aanval Compatible	Yes	Yes

*Other brands and names are the property of their respective owners

Intel's Pattern Matching Strategy

INTEL OPEN SOURCE .org

Hyperscan

01.org/hyperscan

- *Source access via BSD License*
- *Easy to customize and Integrate*
- *High performance*

Network Security Industry

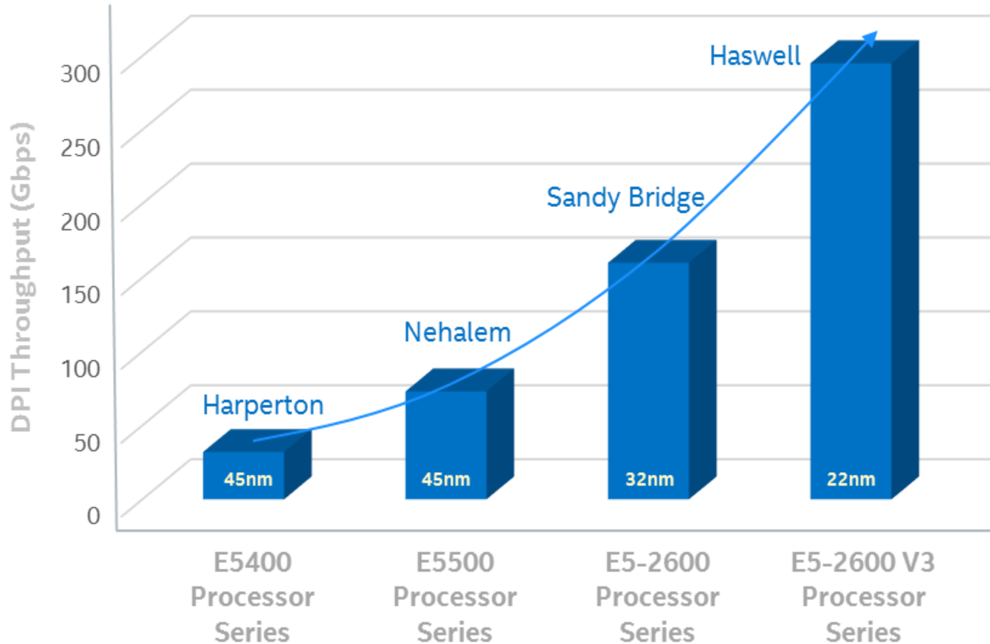
Appliances, Servers, Networking platforms

Intel® Architecture + DPDK + Hyperscan -> Best in class performance

Hyperscan performance on IA

- Using commercial IPS signature database
- HTTP test traffic; real world
- Haswell-EP: 293Gbps
  - Intel® Xeon ® CPU E5-2658 v3 @ 2.20GHz
  - 2 socket, 12 cores per socket, with hyperthreading

Hyperscan scalability on Intel® Xeon® Multi-core Processor Series



- Note: Numbers are subject to change using different benchmarking
- Using Tier-1 OEM commercial IPS signatures

Hyperscan Performance on IA

Intel® Processor	CPU Freq (max GHz)	Platform Details			Peak Scan Perf (GBps)	Per Core Scan Perf (Gbps)	Approx per-core clock-for-clock perf (Gbps): scaled to 2Ghz
		Sockets	Cores / Threads	L3 Cache (MB)			
Intel® Xeon® Processor E5-2699 v3	2.3	2	36/72 (total)	45	555	21.7	18.8
Intel® Xeon® Processor D-1540	2.0	1	8/16	12	86	11.1	11.1
Intel® Xeon® Processor E3-1285 v3	3.6	1	4/8	8	76	16.3	9.0
Intel® Atom™ Processor C2758	2.4	1	8/8	4	22	1.8	1.5

Notes:
- Using database of 250 synthetic regex pattern sets (complex)
- Real world HTTP traffic
- 100% utilization, non-streaming mode
- Raw pattern matching performance, no use case

Hyperscan Performance on IA

"Total raw scanning performance" in Gbps on Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz on IPS workloads (HTTP to-server, to-client and URI traffic). Tier 1 firewall vendor rulesets.

		# of patterns	1C	2C	4C	18C	36C	36C 2T/C
streaming	to_client_1	**69**	23.9	51.3	94.0	350.7	709.2	720.0
streaming	to_client_2	**142**	21.0	40.7	79.4	275.6	562.6	577.5
streaming	to_server_1	**43**	10.8	20.6	40.4	140.1	276.4	285.3
streaming	to_server_2	**235**	6.0	11.1	22.6	75.3	155.1	147.5
block	to_server_uri_1	**13110**	3.6	5.7	11.2	45.0	90.6	102.4
block	to_server_uri_2	**8801**	4.4	8.8	17.2	73.1	142.6	149.4

IA Drives Performance(1)

General processor features

- Wide issue (tuned loops can issue 3-4 instructions per cycl of useful work)
- Cache rich architecture
  - High bandwidth to Level 1 and Level 2 cache
  - Large L2 and L3 allows matching tables for literal matching to stay cache resident
  - Large L2 is *unshared* which means, unlike much of IA competition, scaling keeps going – unshared L2 bandwidth is per-core not per-chip
- Hyperthreading enables additional performance (15-20% is typical)

IA Drives Performance(2)

Instruction sets

- Process large numbers of characters using SIMD: SSE2, SSSE3
- AVX2.0 enables processing of large amounts of input data in one step
- SIMD operations are resource friendly and fast on IA; enables large matching engines e.g. NFAs with big state counts
- BMI1/BMI2 also a 1:1 match for many pattern matching primitives: PEXT/PDEP replace a 10-30 instruction *loop* with 1 instruction

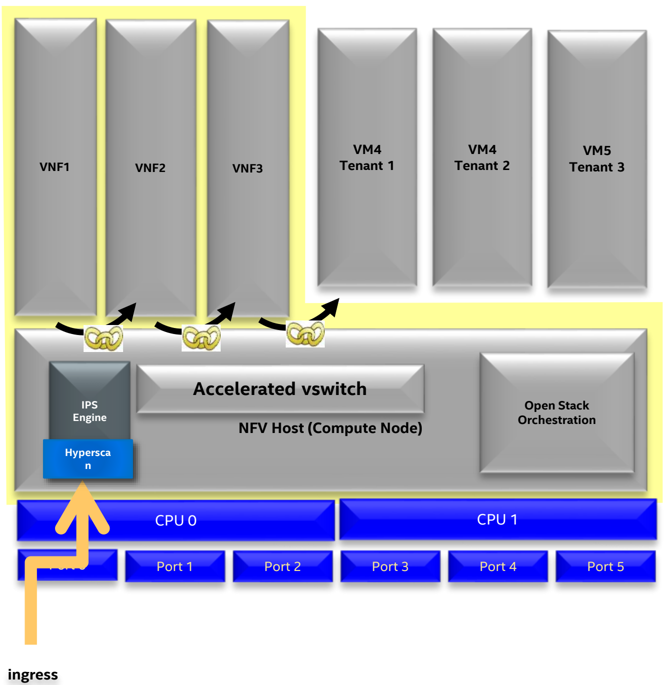
Hyperscan
(Software DPI)

Benefits of Software DPI

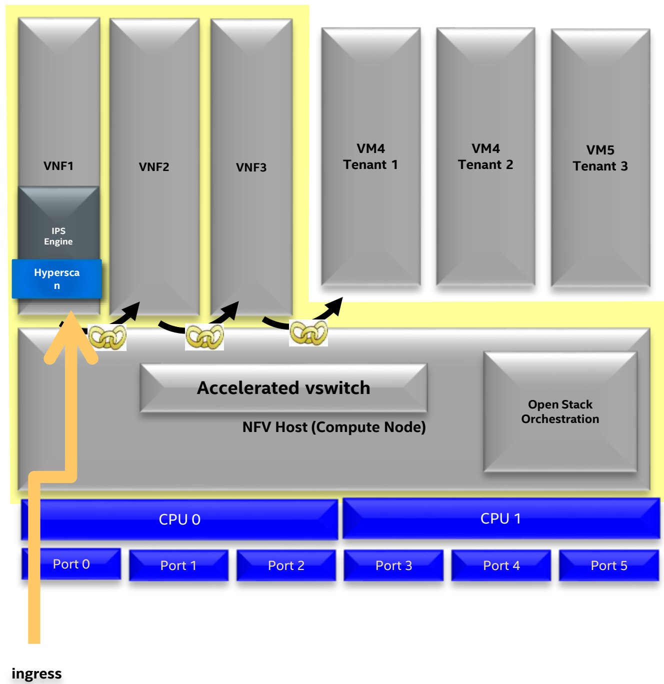
- Fastest solution of it's kind in the market that scales Intel® Architecture
- Enhance DPI performance of security products while increasing inspection intelligence, without HW re-design.
- Enable field upgrades for legacy products
- SDN/NFV deployment flexibility

Industry transformation:
- Layer 7 intelligence
- Performance
- Scalability
- NFV
  - Flexibility
  - Scalable VMs
  - Maximize CPU resources

Example: Use cases for service-chained VIPS



1. Run IPS inside the compute node



2. Run IPS inside the VM

Hyperscan 4.4

Fat runtime:

Build several variants of the Hyperscan scanning engine specialised for different processor feature sets,

and use the appropriate one for the host at runtime. This uses the "ifunc"

indirect function attribute provided by GCC and is currently available on

Linux only, where it is the default for release builds.

Hsbench:

Standard Hyperscan performance benchmarking tool

Welcome to contribute rules and data so that we can help to address your issues

Can create a Hyperscan benchmarking corpus database from a supplied

group of Project Gutenberg texts, simple text and pcap files.

Note: small corpus could cause unstable performance

Hyperscan 4.5 and beyond

Approximate matching:

Levenstein distance matching, better give an example

Stream state compression

New APIs to compress and decompress stream state

Future:

AVX512 implementation

Logical combination of patterns(and, or, not, ordered and, etc)

More pattern support(backreference, lookaround assertions)

Additional Links

For more information contact:
Jerry Zhang
jerry.zhang@intel.com

Hyperscan Open Source Software Project:
https://01.org/hyperscan

Github Code Repository:
https://github.com/01org/hyperscan

Hyperscan Intel.com Landing Page:
http://www.intel.com/content/www/us/en/communications/hyperscan.html

Hyperscan Podcast:
https://soundcloud.com/intelchipchat-networkinsights/hyperscan

intel