The bridge to possible

All You Need to Know About Forwarding on the Catalyst 8000 platforms

BRKENT-2653

David Roten
Technical Marketing Engineer / Technical Leader

Cisco Webex App

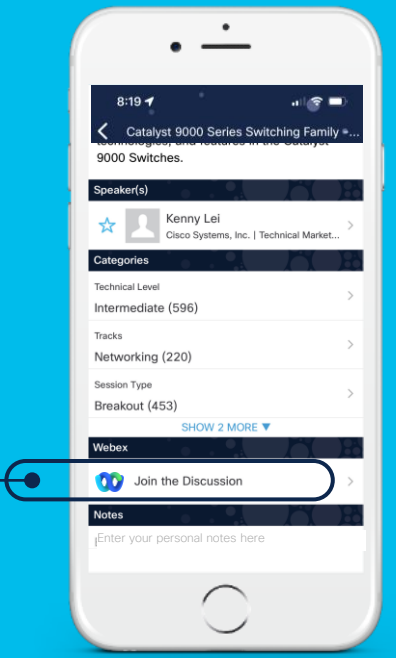
Questions?

Use Cisco Webex App to chat
with the speaker after the session

How

1. Find this session in the Cisco Live Mobile App
2. Click “Join the Discussion”
3. Install the Webex App or go directly to the Webex space
4. Enter messages/questions in the Webex space

Webex spaces will be moderated until February 24, 2023.



Agenda

- What are the forwarding mechanisms?
- Where is each mechanism used & why?
- Specific considerations for each?
- How to observe resource utilization for each mechanism?
- Conclusion

Hardware architectures

IOS XE routing hardware architectures

All IOS XE routing platforms use a multi-core and multi-threaded data plane architecture.

224 cores
896 threads

3.0

Catalyst 8500 platforms

ESP100-X
ESP200-X

64 cores
256 threads

2.0

ASR1002-HX
ASR1001-HX

ESP100
ESP200

4+ cores
& threads

Catalyst 8500L-8S4X
Catalyst 8300 platforms
Catalyst 8200 platforms
Catalyst 8200L platforms

Catalyst 8000v
ISR1100-4G / 6G
ISR4300

4+ cores
& threads

ISR1000 platforms

ISR4400

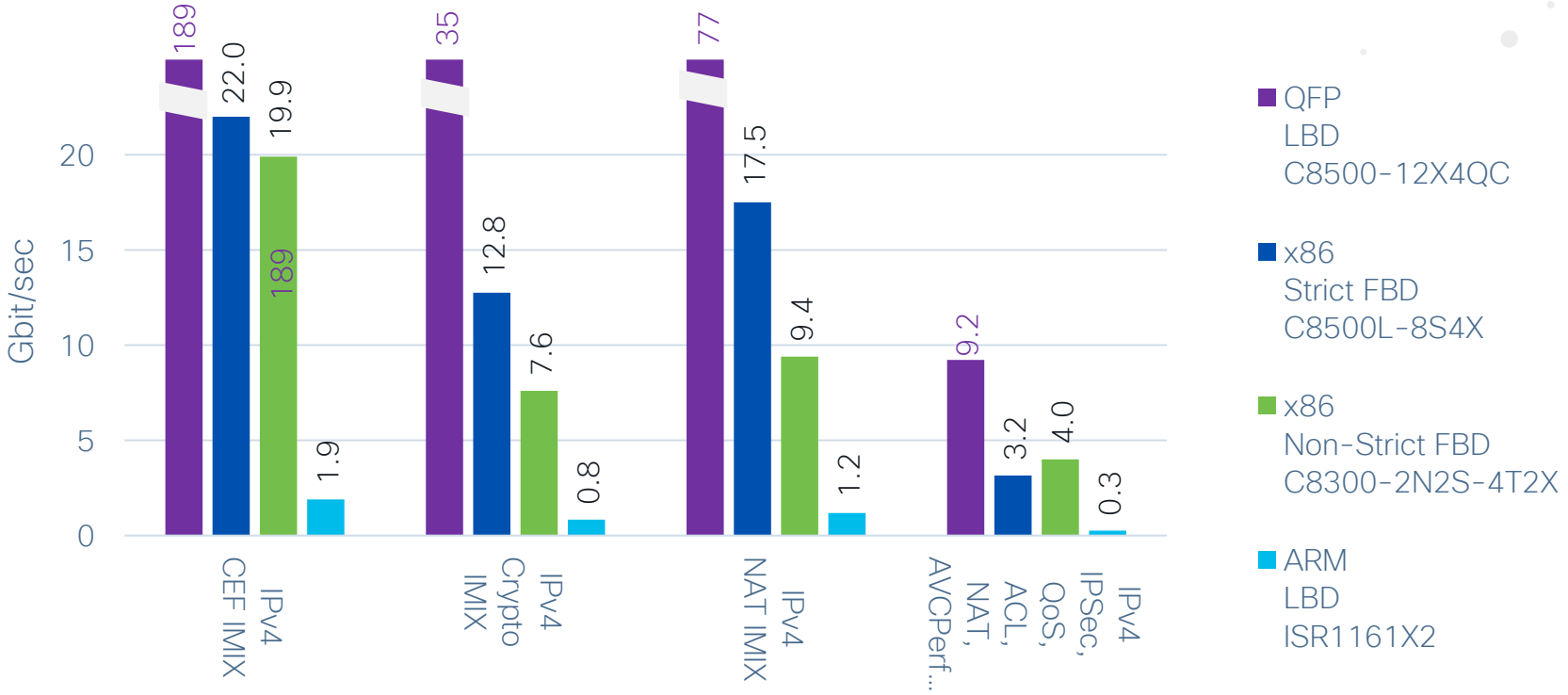
Basic data plane function implementation

- Receiving traffic
- Distributing traffic
  - Load based distribution
  - Non-strict flow based distribution
  - Strict flow based distribution
- Crypto processing
- Forwarding action
- Queuing and scheduling

Baseline platforms comparison

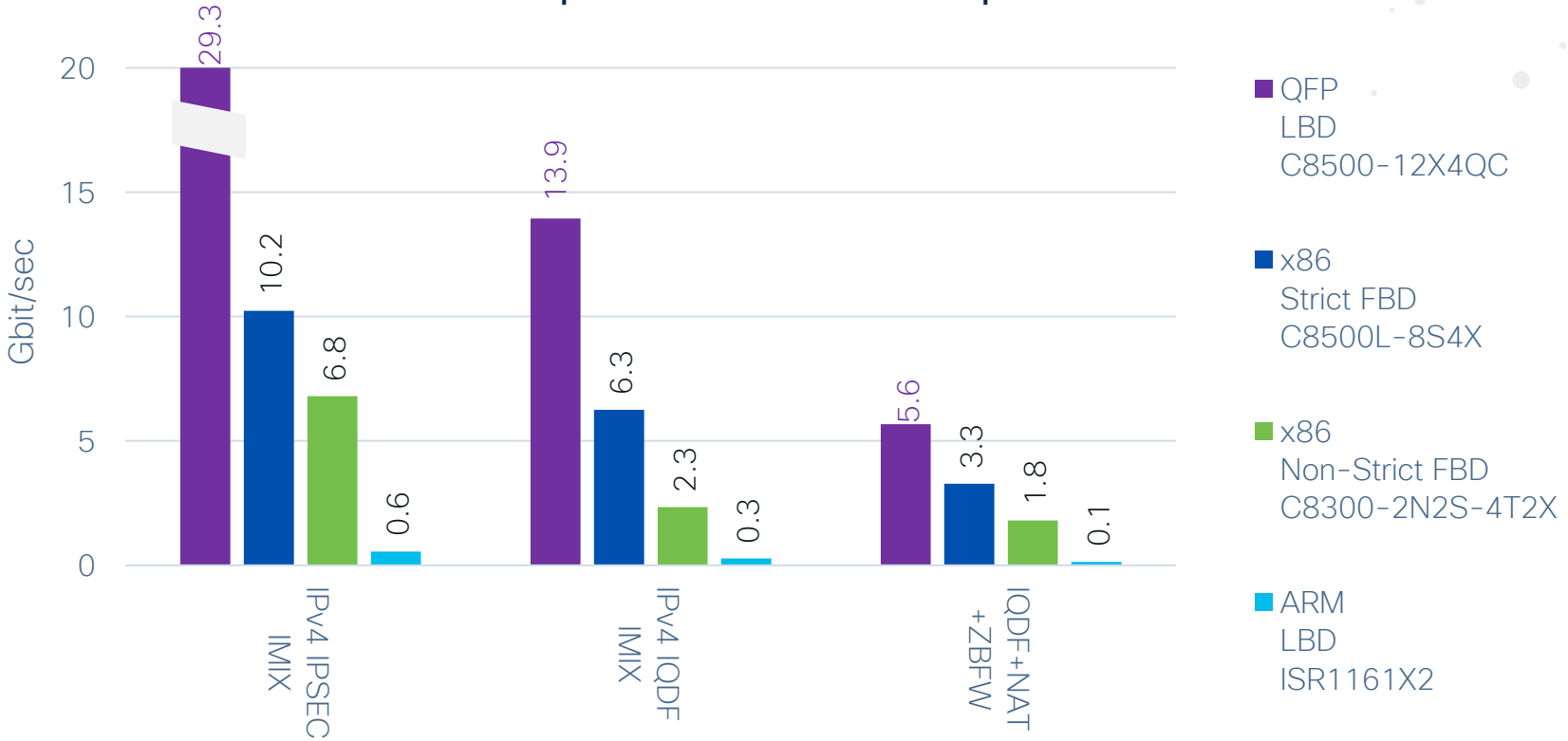
Autonomous mode	**ARM LBD** ISR1161X	**x86 Non-Strict FBD** C8300-2N2S-4X2T	**x86 Strict FBD** C8500L-8S4X	**QFP 3.0** C8500-12X4QC
Traffic acceptance (Rx)	Shared Rx thread	2 shared cores	2 dedicated cores	Dual priority with **hardware assist**
Traffic distribution (Rx)	Shared Rx thread 1 stage	2 shared cores 1 stage	2 dedicated cores **2 stage**	**Hardware** distributor
Crypto processing	shared thread	2 shared cores	**1 encrypt core 1 decrypt core**	16 engines **dedicated hardware**
Forwarding (PPE)	2 threads	10 hyperthreads	**12 hyperthreads**	896 threads
Queuing (TM)	1 shared thread	2 shared cores	2 dedicated cores	**Hardware**

Baseline platform comparison



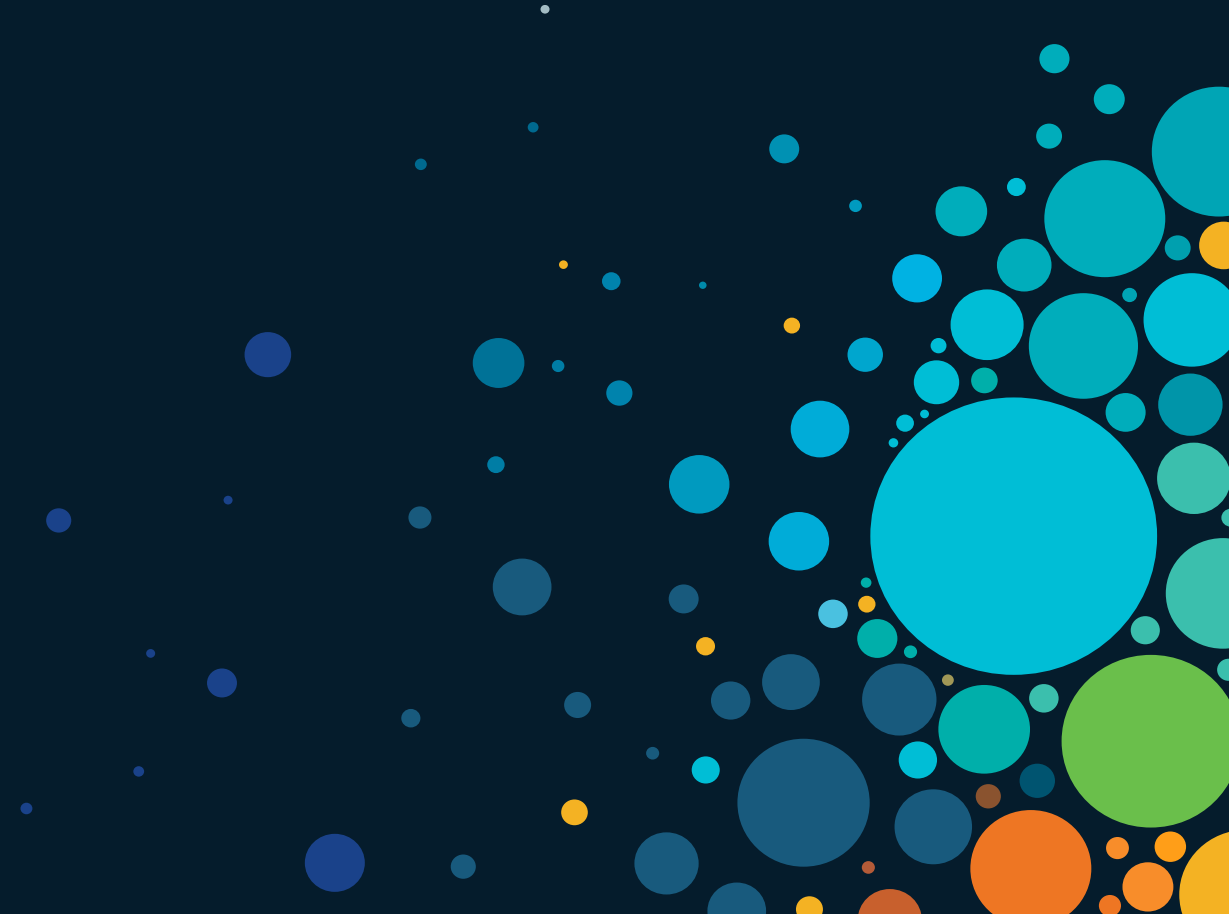
This data should not be used for official performance guidance.

SDWAN Baseline platform comparison



This data should not be used for official performance guidance.

Concepts



Lexicon (1/2)

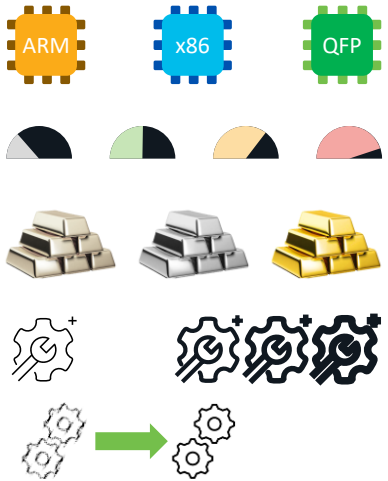
- **Rx** (receive):
  Function that receives packet from an ingress interface and either [first pass hashes to PP or PP-Rx function | or distributes packet to PP based on load]
- **PP-Rx** (packet processor – receive):
  Function that does second pass hashing of traffic to PP function. Specific to FBD.
- **PP** (packet processor):
  Function that does network processing of protocol traffic (NAT, MPLS, Firewall, ACLs, classification, etc)
- **COFF** (crypto):
  Function that encrypts and decrypts traffic
- **Tx** (transmit):
  Function that schedules and transmits a packet on an interface

Lexicon (2/2)

- Stateful flow features
  - NAT, ZBFW
  - NBAR, Flexible Netflow, UTD
- Cores and threads
  - Non-hyperthreaded x86 systems: a single thread per CPU core
  - Hyperthreaded x86 systems: two threads associated with a given CPU core. Not all cores in a hyperthreaded system must use hyperthreading. Two threads per core **does not** double compute performance. Typically, two hyper threads on a single core will have 20-30% more compute power as a single non-hyperthreaded core.

What are forwarding mechanisms?

- IOS XE routing architecture is capable of using a multi-core and multi-threaded data plane –
- Multiple metrics involved:
  - Hardware architecture in a given platform
  - Targeted performance for a given product
  - Targeted price point for a given product
  - Economy of scale in build components
  - Optimizations in software development



Routing platform consistencies

- IOS XE platforms use the same software for dataplane and control plane features.
- Features can be delivered across platforms using the same source software providing consistency and compatibility across platforms.
- **Distribution of traffic amongst the dataplane threads varies.**
- Various platforms offer differing amount of compute power.
- The forwarding architecture:
  - **does not** affect feature availability
  - **does** affect the forwarding capacity and potential performance advantages as well as pinch points

Detrimental influencers

Detrimental influencers

- Packet Ordering and Ordered Mutual Exclusion (ordered access to data)

  If packets from a given flow are handled by different threads, the cores holding on to the 2 through n packets might have to wait as the system deals with all the complications of the first packet to hand stateful processing of the flow.

- Stateful features have to allocate and initialize structures when encountering a new flow. This makes the processing time for the first packet longer than for subsequent packets.

- Crypto acceleration hardware differs across platforms. Some crypto hardware requires walking through the packet feeding its data into the hardware and getting a result back. On these platforms, larger packets take longer to process and if a flow has packets varying in size, this waiting situation can occur.

Detrimental influencers

- Memory contention

  If many threads are handling a flow, they may all need access to the same memory to update statistics, do flow rate calculations, get and update stateful information about the flow, etc.

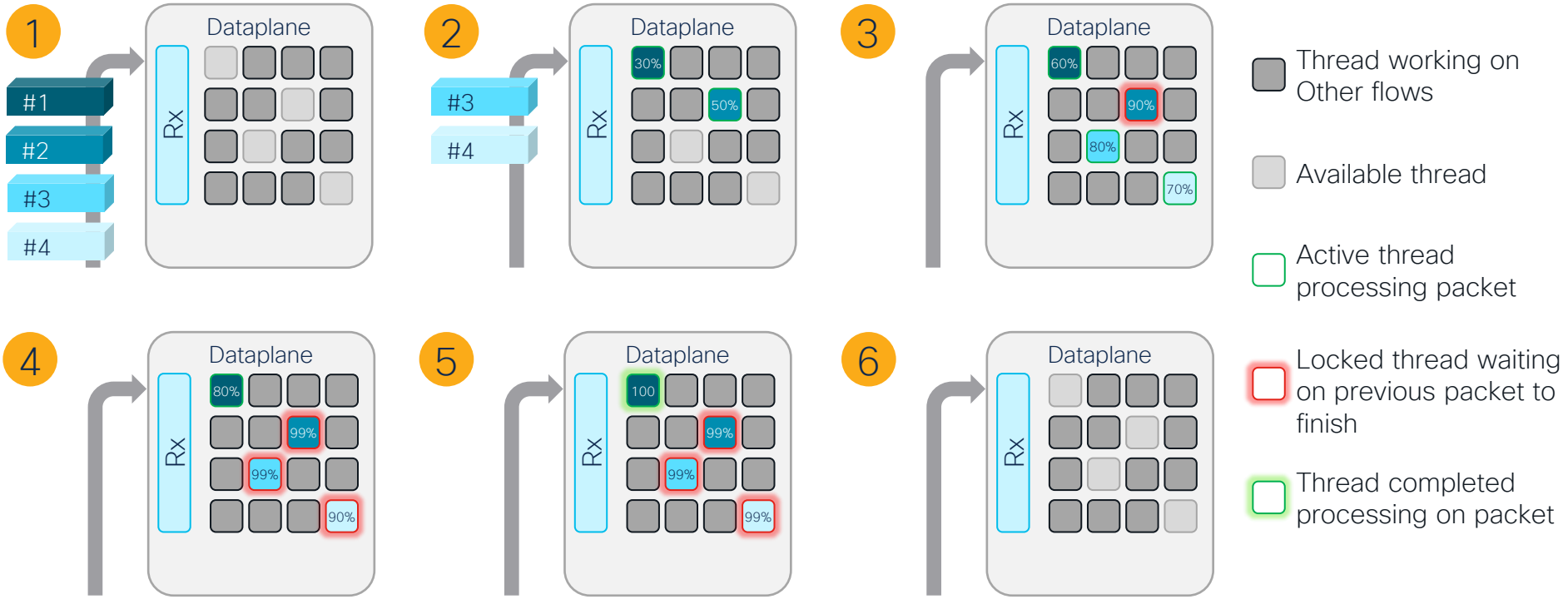
  This overlaps somewhat with the issues of ordered mutual exclusion.

Detrimental influencers

- L1 and L2 x86 / ARM caching

  Caching is not optimized if different threads need to pass information through the L1 and L2 caches instead of keeping it local

Packet ordering issues

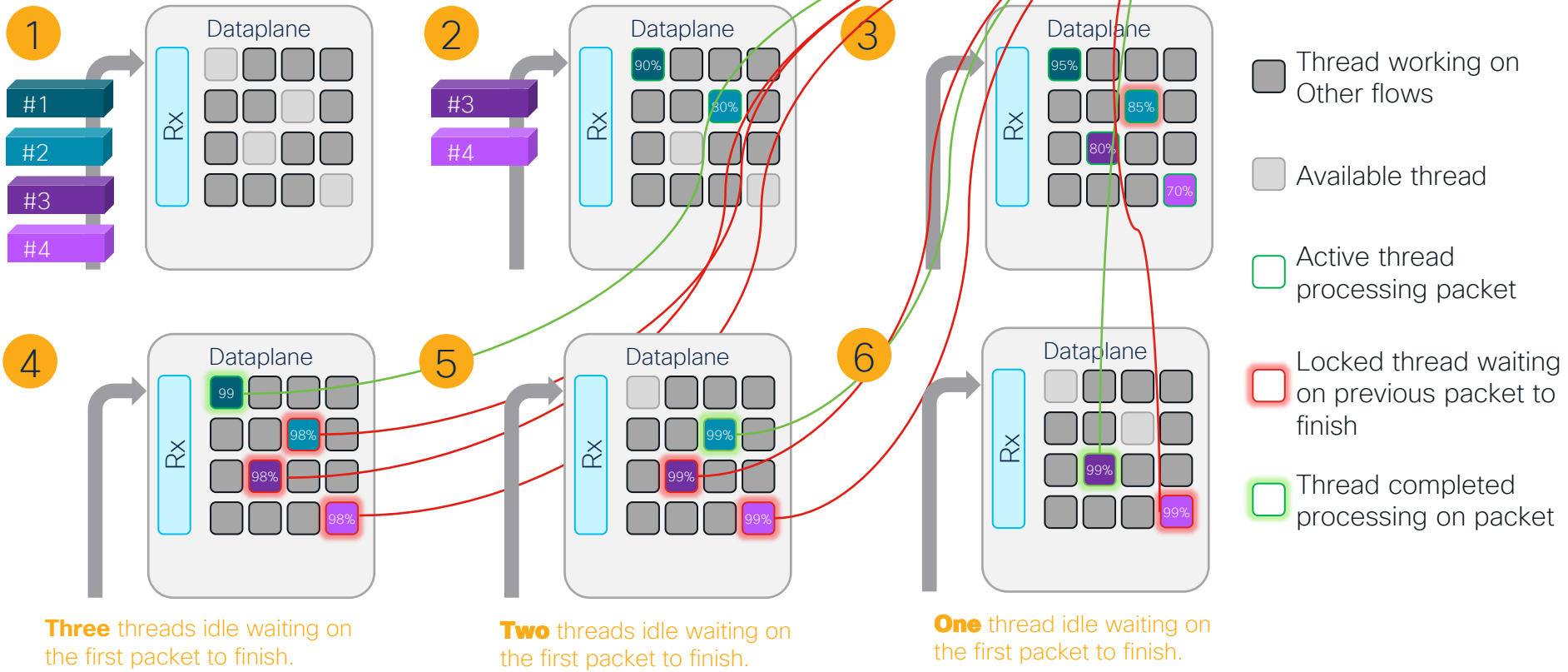


Two cores idle waiting on the first packet to finish.

Three cores idle waiting on the first packet to finish.

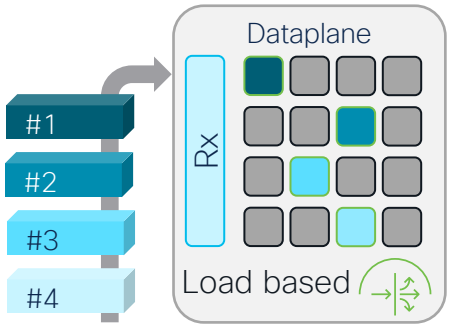
All four cores clear locks out as the first packet finishes and is forwarded.

Example of shared state across cores – ordered mutual exclusion



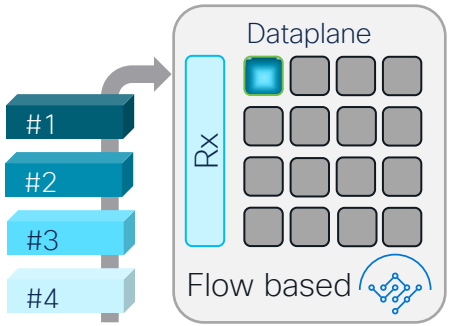
L1/L2 cache efficiency

- **QFP based platforms have an optimized design** that makes L1/L2 caching less of a concern as various threads process the same flows
- General purpose CPUs have L1/L2 caches that are dedicated to each thread/core. The caches are **not** shared amongst the threads/cores. L3 caches may be shared depending on the implementation



Memory delay #1: 2 ns
Memory delay #2: 12 ns
Memory delay #3: 12 ns
Memory delay #4: 12 ns

Total delay: **38 ns**



Memory delay #1: 2 ns
Memory delay #2: 2 ns
Memory delay #3: 2 ns
Memory delay #4: 2 ns

Total delay: **8 ns**

Serialization is actually more efficient!

Forwarding Mechanisms

Forwarding architectures

Load based (LBD)

Packets are be handled by any available data plane thread.

Generally used on platforms with more threads than interfaces.

Strict Flow based (S-FBD)

Packets are **strictly** distributed based on flow hashing.

Available data plane threads **will not** assist busy data plane threads.

Non-strict Flow based (NS-FBD)

Packets can be handled by any available data plane thread.

Efforts are made to keep packets from a given flow on a given thread but no guarantees.

Forwarding architectures – IOS XE 17.9

The three forwarding architectures available today are and there uses

Load based (LBD)

- QFP
  - ASR1000
  - C8500
- MIPS
  - ISR4400

Load based (LBD)

- x86
  - C8000V
  - ISR4300
- ARM
  - ISR1000

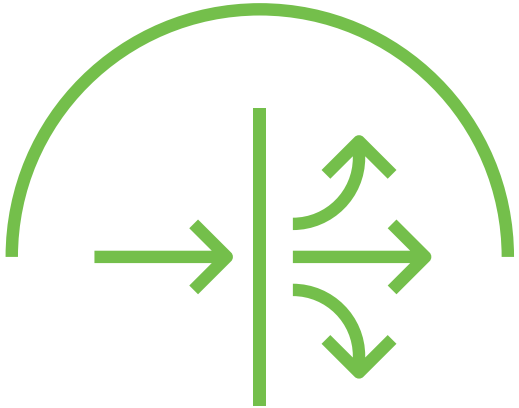
Strict Flow based (SFBD)

- C8500L

Non-strict Flow based (NSFBD)

- C8300
- C8200
- C8200L
- ISR1000-4G/6G

Load based architecture for x86 and QFP



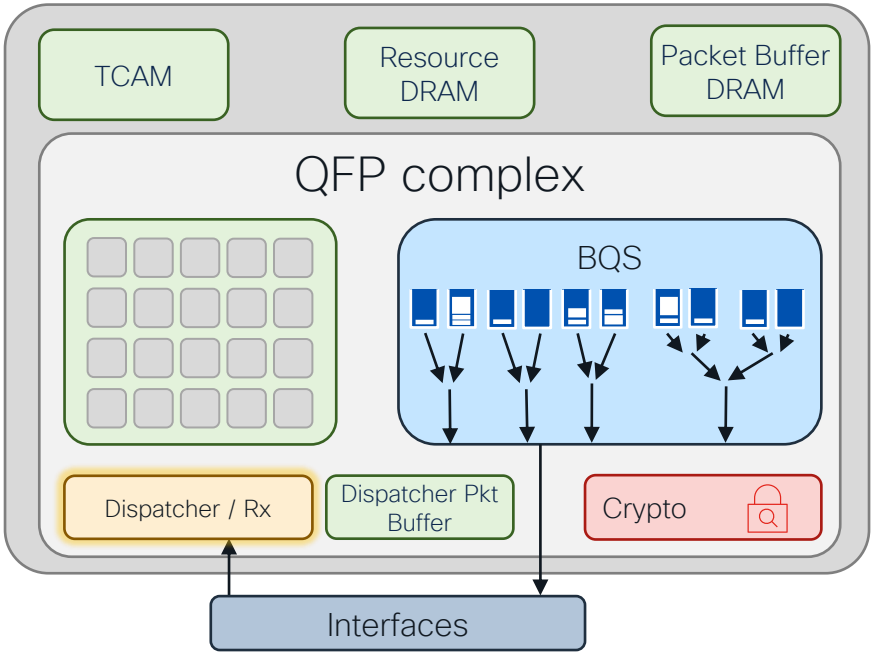
Load Based Distribution

- Packets are distributed to threads strictly based on availability by a hardware distributor (QFP) or Rx thread (x86/ARM based systems)
- Efforts (*but not guarantees*) are made at ingress to keep traffic for a given ingress port on the same thread.
  - Challenging if there are a son software platforms mall number of active ports/queues
  - Challenging with a larger number of threads available vs ingress ports
- Load based distribution allows parallelism of processing for a single flow
  - Strict FBD does not. Non-strict FBD runs between the two.

Load Based Distribution – characteristics

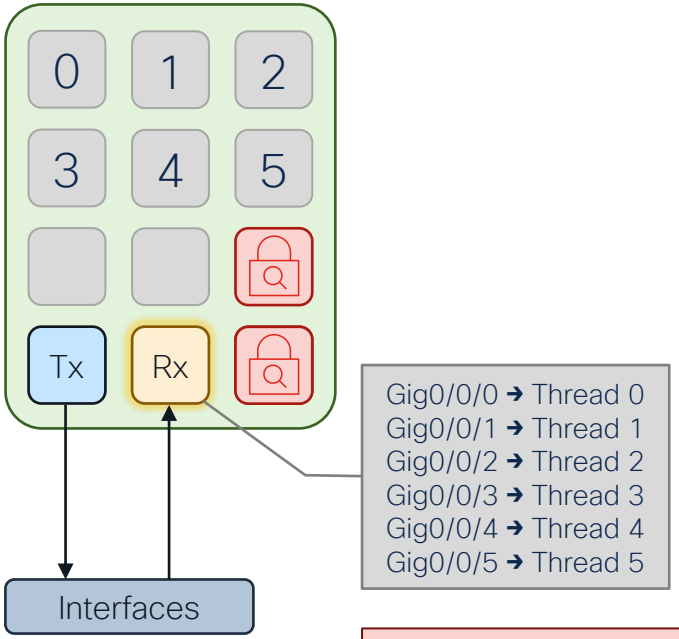
- A packet from any given flow can land across multiple threads
- State for a flow must be readily available to any core at any time
- **QFP platforms provide hardware assist** for this functionality making it easier and simple to implement without significant performance hit
- x86 and ARM platforms do not have hardware assist
  The following are valid in x86 and ARM systems but may not cause significant determent.
  - Packet ordering and ordered access to data – challenge
  - Memory contention for tracking flow state – challenge
  - X86/ARM L1 and L2 cache implications – challenge

QFP based load distribution



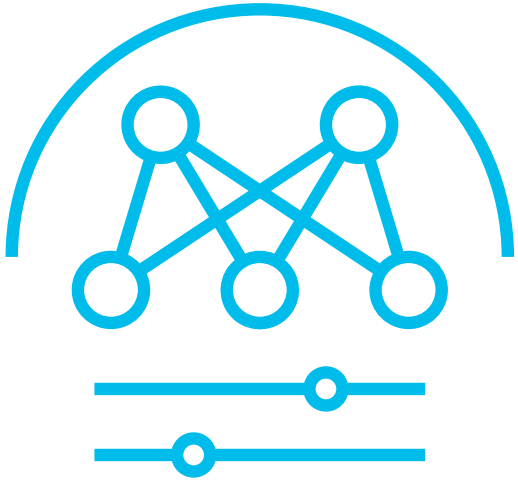
- QFP has Dispatcher Hardware which finds an available thread to process packets
- Dedicated resource so no other contention
- Dispatcher has excess capacity so **no "Rx" bottleneck concerns in QFP platforms**

x86 based load distribution



- x86 and ARM platforms use a generic thread for the Rx function. This thread is responsible to queue packets for distribution to PPEs.
- Since this is not a dedicated resource **in x86 and ARM platforms, there is the potential that the Rx thread can be a bottleneck**.

Non-strict flow based architecture



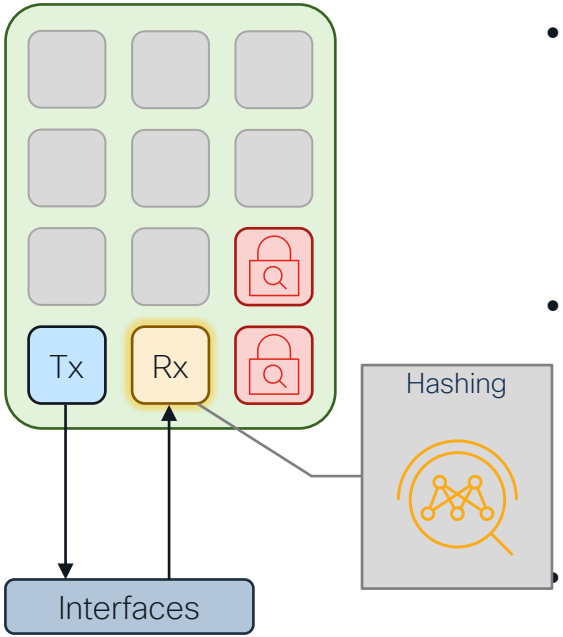
Non-Strict Flow Based Distribution - characteristics

- Packets from a flow may, but not always, land on different threads
- State for a stateful flow must be readily available to any core at any time *just in case*
- x86 platforms are **do not have hardware assist**
  The following are valid in x86 systems but may not cause significant detrement.
  - Packet ordering and ordered mutual exclusion in software – challenge
  - Memory contention for memory tracking flow state – challenge
  - x86 lower L1 and L2 cache implications – challenge

Non-Strict Flow Based Distribution

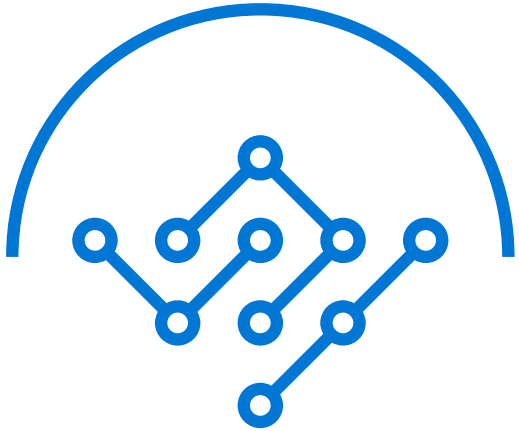
- Packets are classified on a tuple from the **outermost** encapsulation by Rx thread
- Fragments use 3-tuple classification
- If that initially targeted thread is busy (elephant flow scenario / uneven flow hashing), the packet can be handled by a PPE thread that has free cycles
- In heavy load situations with few flows, the behavior is very similar to a load based distribution scenario
- In the case we are the destination of an IPSEC tunnel, hashing is done based on the interior address after decryption

Non-Strict Flow Based Distribution



- x86 platforms use a generic thread for the Rx function. This thread is responsible to find an available data plane thread to process the packet.
- On non strict flow based platforms, this requires the additional step of hashing the outermost header of the packet (higher cost than the load based scenario)
- Since this is not a dedicated resource **in x86 platforms, there is the potential that the Rx threads can be a bottleneck** (CEF / racetrack).

Strict flow based architecture



Strict Flow Based Distribution - characteristics

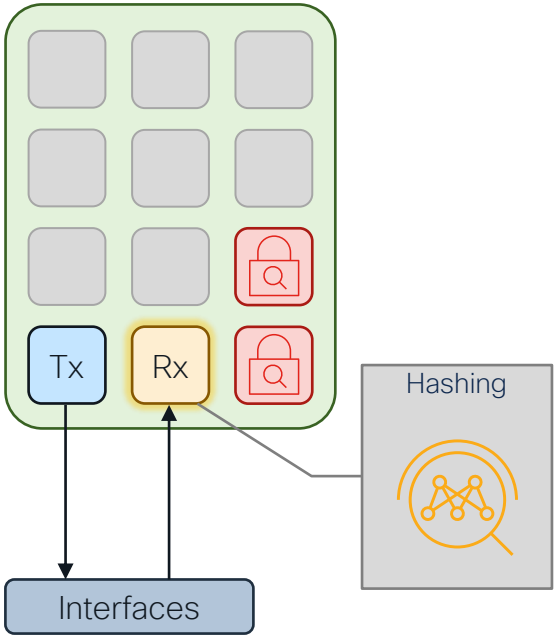
- Packets from a flow always land on the same PP thread
- State for a stateful flow must be readily available to only a single thread

- x86 platforms **are optimized**
  - Packet ordering and ordered access handled by hardware – optimized
  - Less memory contention for memory tracking flow state – optimized
  - More efficient x86 L1 and L2 cache usage – optimized
- **Elephant flows are a concern** as flows are strictly mapped to a thread with no offload for high-utilization

Strict Flow Based Distribution

- Packets are classified on a tuple on the outermost encapsulation by Rx thread in the first pass by the Rx function
- Fragments use a 3-tuple classification
- Packets are assigned to a PPE-Rx thread based on the initial hashing by the Rx thread
- PPE-Rx function will check the flow database and based on that potentially redirect the packet to an alternate core
- There is no offloading at PPE-Rx if a thread is 100% utilized because there is no sharing of stateful flow information between threads.
- In the case we are the destination of an IPSEC or GRE tunnel, hashing is done based on the interior address after decryption

Strict Flow Based Distribution



- x86 platforms use a generic thread for the Rx function. This thread is responsible to find an available data plane thread to process the packet.
- On strict flow based platforms, this requires the additional step of hashing the header of the packet (higher cost than the load based scenario)
- Rx thread inspect the outermost header and PPE-Rx thread may also inspect inner header.
- Since Rx compute is not a dedicated resource **in x86 platforms, there is the potential that the Rx thread can be a bottleneck**.

The tuple classifications

	VRF ID	Src IP v4/v6	Dest IP v4/v6	Protocol	Src port	Dest port	SPI	GRE key	Dst MAC	Src MAC
Fragment	✓	✓	✓	✓						
UDP / TCP	✓	✓	✓	✓	✓	✓				
ESP	✓	✓	✓	✓			✓			
GRE	✓	✓	✓	✓				✓		
L2									✓	✓
Default	✓	✓	✓	✓						

MPLS will generally be inspected for recognized payload encapsulation and MPLS tags.

Placement guidance for S-FBD platforms

- Strict flow-based distribution platforms are best suited for network locations where there is a diversity of flows.
  - High flow count generic routing
  - High flow count internet gateway
  - **High flow count** crypto action
  - **High flow count** SDWAN edge router
- The theme here is **High flow count**!

Per internal flow performance will always be limited by single thread forwarding capacity.

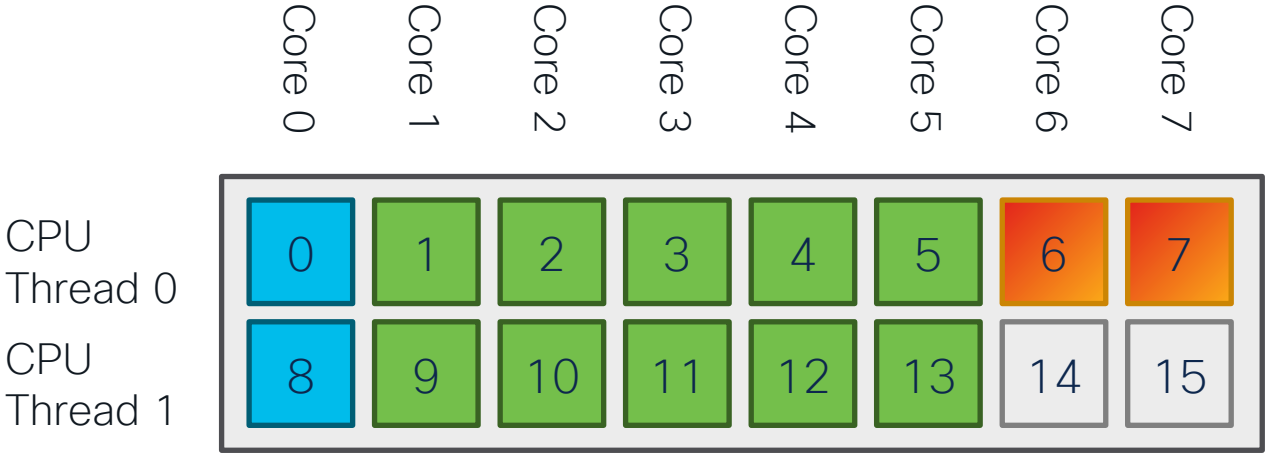
Encapsulated traffic as a transit router will be seen as a single flow between tunnel endpoints.

High → 1000s, 10000s, 100000s
Maximum supported flow count is 2 million. A bidirectional flow will consume two entries.

Core distribution

DONE

Core dist – C8300-1N1S-4T2X – DP heavy



This is the allocation for IOS XE 17.9 software. Other software versions may have different allocations.

Service plane

Data plane

Control plane

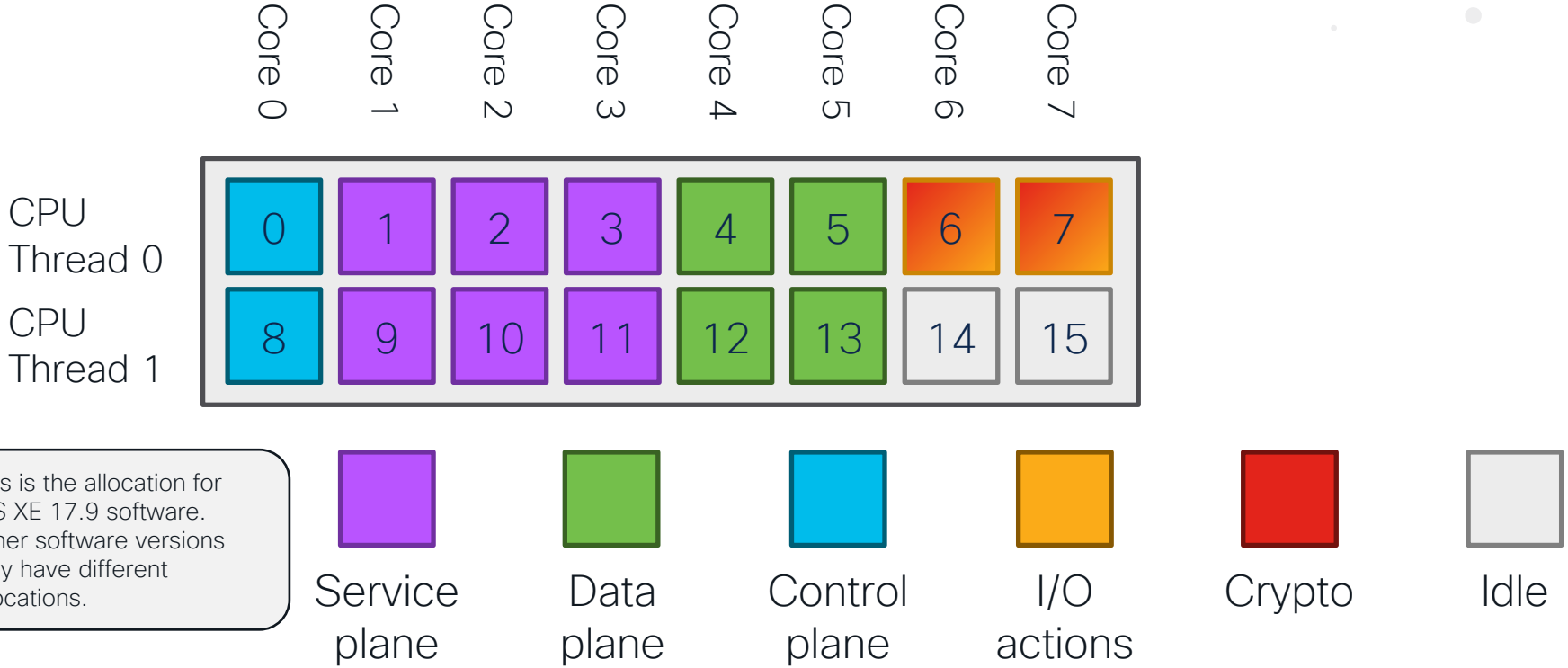
I/O actions

Crypto

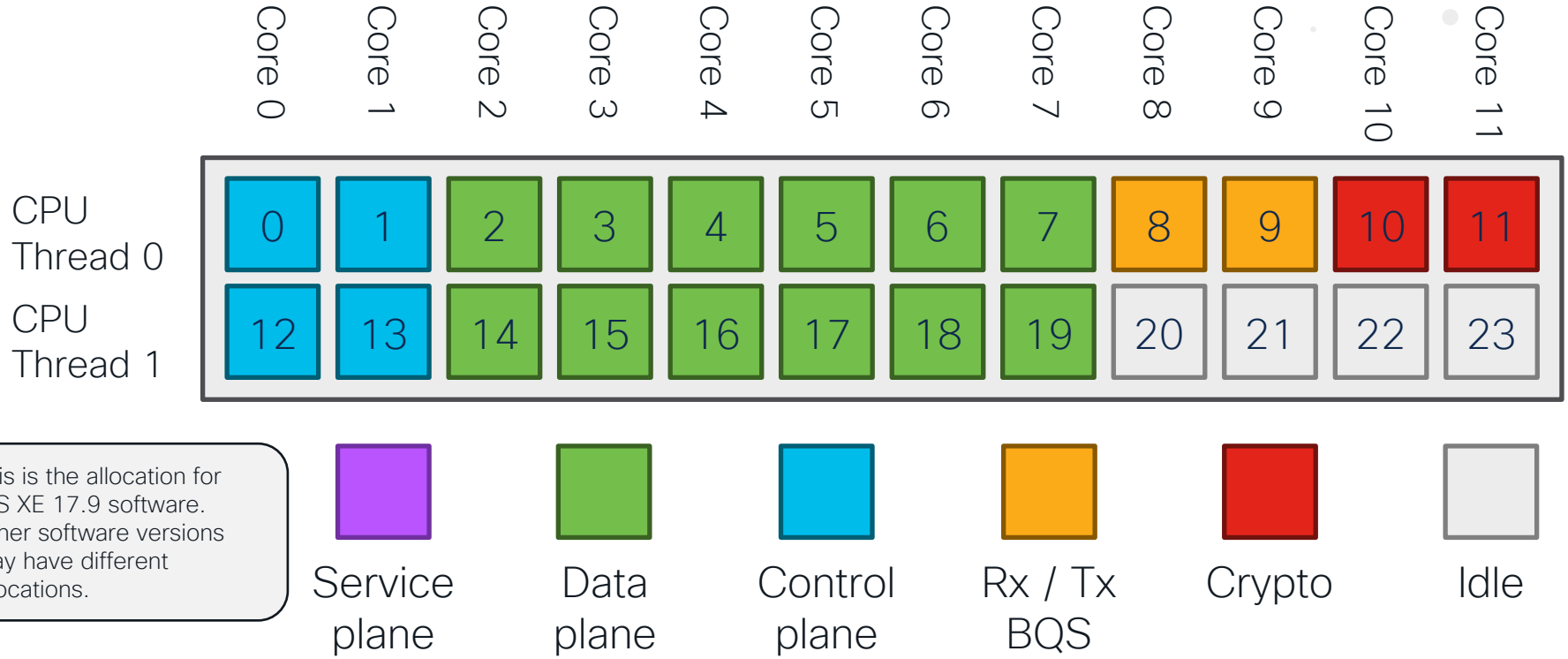
Idle

DONE

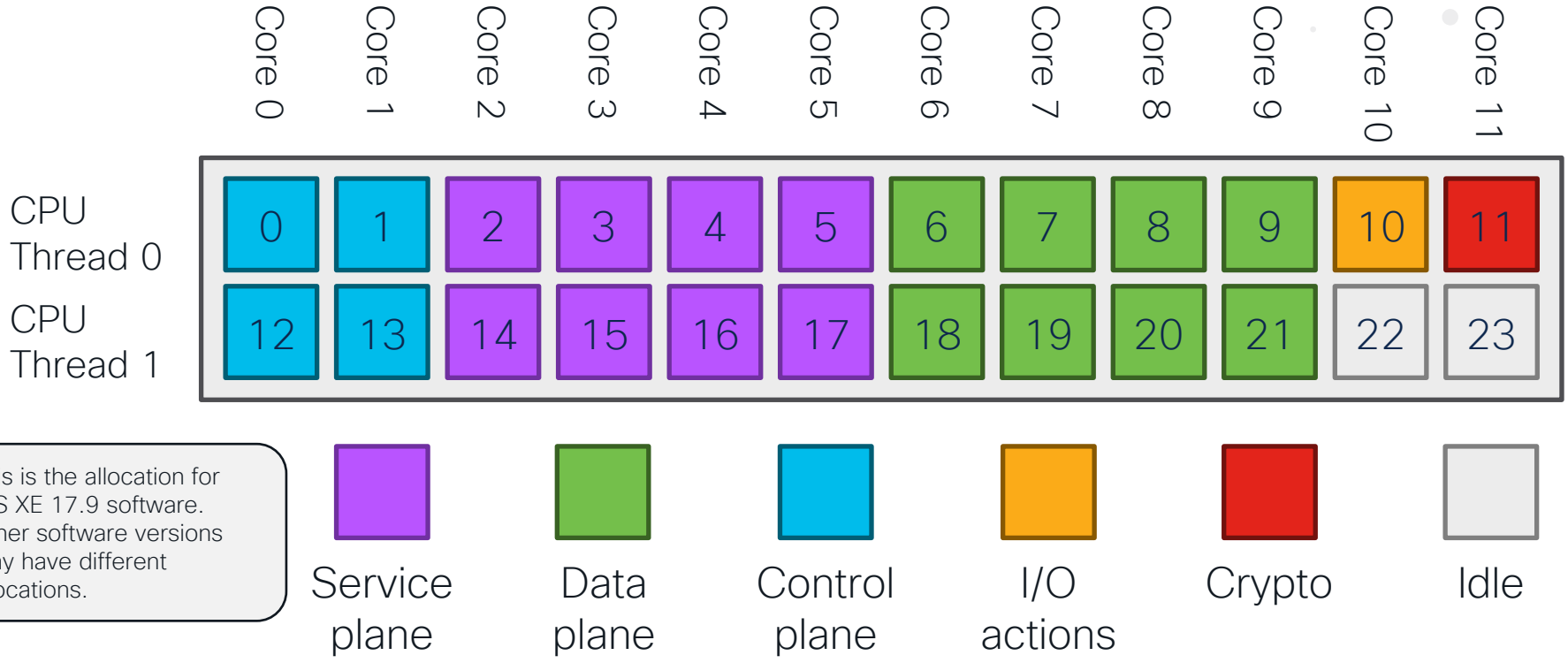
Core dist – C8300-1N1S-4T2X – SP heavy



Core distribution – C8500L-8S4X – DP heavy

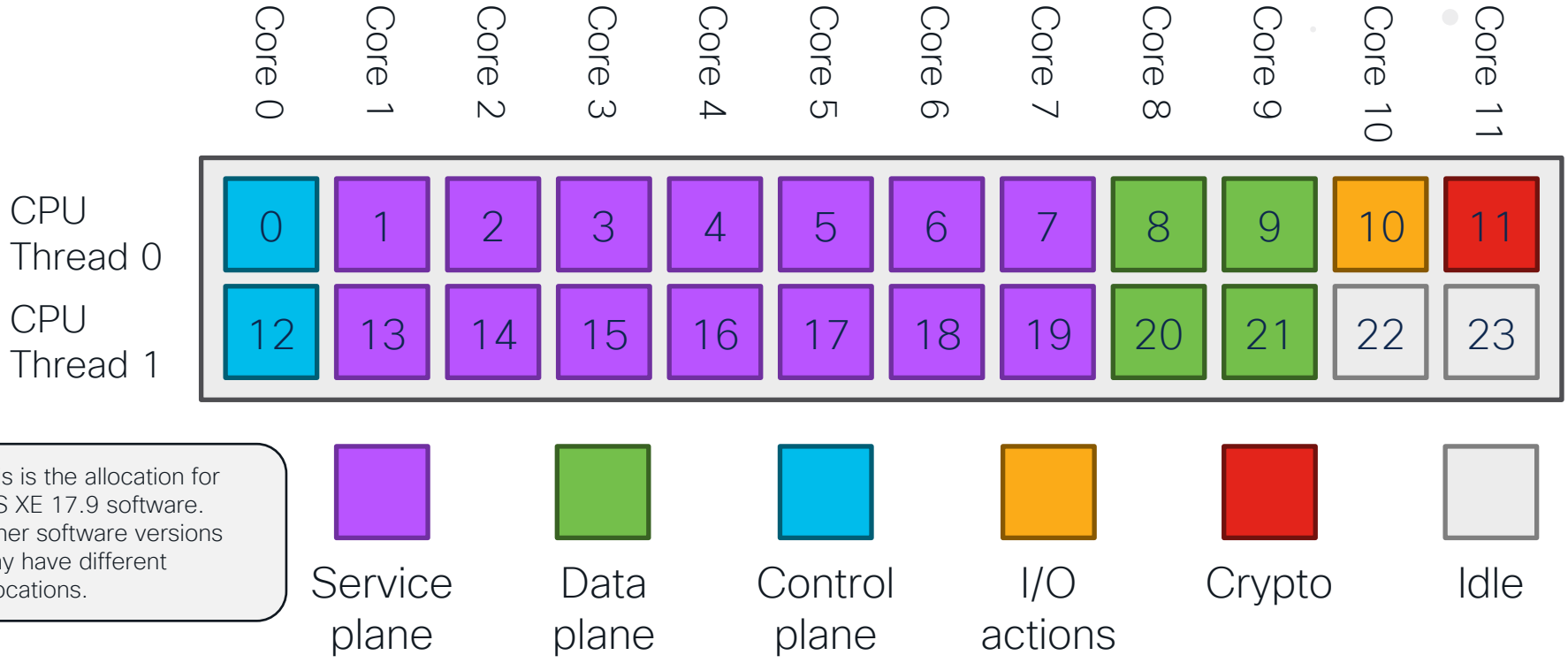


Core distribution – C8500L-8S4X – SP heavy



This is the allocation for IOS XE 17.9 software. Other software versions may have different allocations.

Core distribution – C8500L-8S4X – App heavy



This is the allocation for IOS XE 17.9 software. Other software versions may have different allocations.

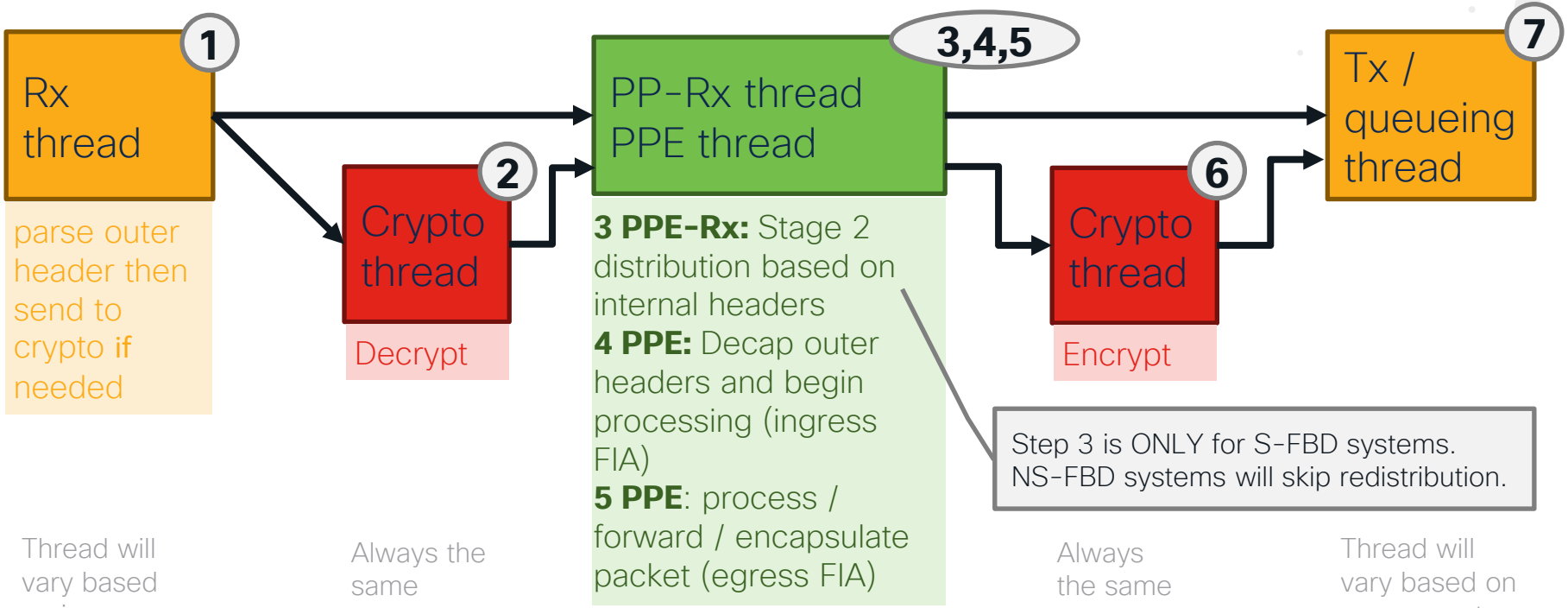
C8500L x86 thread allocation (XE 17.9)

```
C8500L-8S4X#show platform software cpu alloc
CPU alloc information:

  Control plane cpu alloc: 0-1,12-13

  Data plane cpu alloc: 2-11,14-19

  Service plane cpu alloc: 0

  Slow control plane cpu alloc:
  Template used: CLI-data_plane_heavy
```

```
C8500L-8S4X#show platform software cpu alloc
CPU alloc information:

  Control plane cpu alloc: 0-1,12-13

  Data plane cpu alloc: 6-11,18-21

  Service plane cpu alloc: 2-5,14-17

  Slow control plane cpu alloc:
  Template used: CLI-service_plane_heavy
```

Packet flow and analysis

Ingress packet flow for Strict FB



Testbed config for following data

- C8500L-8S4X running IOS XE 17.8.01a
- Data-plane heavy mode
- Using TenGig 0/1/2 and TenGig0/1/3
- Single IP address configured on main-interfaces
- Injected traffic is 500 byte packets, 20% line rate on each port
- Randomized UDP source and destination ports
  - 0x00FF mask for UDP ports, 256x256x2 = 131072 flows total

Step 1a – Ingress processing

```
C8500L-8S4X#show plat hardware qfp active datapath infrastructure bindings
Port Instance Bindings:
```

ID	Port	IOS Port	WRKR12	WRKR13
1	rcl0	rcl0	Rx	Tx
2	ipc	ipc	Tx	Rx
3	vxe_punti	vxe_puntif	Tx	Rx
4	**fpe0**	**GigabitEthernet0/0/0**	Tx	**Rx**
5	fpe1	**GigabitEthernet0/0/1**	Tx	**Rx**
6	fpe2	**GigabitEthernet0/0/2**	**Rx**	Tx
7	fpe3	**GigabitEthernet0/0/3**	Tx	**Rx**
8	fpe4	**GigabitEthernet0/0/4**	**Rx**	Tx
9	fpe5	**GigabitEthernet0/0/5**	Tx	**Rx**
10	fpe6	**GigabitEthernet0/0/6**	**Rx**	Tx
11	fpe7	**GigabitEthernet0/0/7**	Tx	**Rx**
12	fpe8	**TenGigabitEthernet0/1/0**	**Rx**	Tx
13	fpe9	**TenGigabitEthernet0/1/1**	Tx	**Rx**
14	**fpe10**	**TenGigabitEthernet0/1/2**	**Rx**	Tx
15	fpe11	**TenGigabitEthernet0/1/3**	Tx	**Rx**

Tx / Rx mapping has changed across releases.
Be sure to check your platform and release.

Rx = ingress processing
Tx = TM = queuing and scheduling

Step 1b – Ingress processing

```
C8500L-8S4X#show platform hardware qfp active datapath infrastructure sw-cio
Credits Usage:

  ID      Port  Wght  Global WRKR0  WRKR1  … WRKR10  WRKR11  WRKR12  WRKR13  WRKR14  WRKR15  Total
   1      rcl0     1:    6048     0      0  …      0       0      96       0       0       0   6144
   1      rcl0   128:    6048     0      0  …      0       0      96       0       0       0   6144
   2       ipc     1:       0     0      0  …      0       0       0       0       0       0      0
   3 vxe_punti     1:     466     0      0  …      0       0       0      46       0       0    512
   4      fpe0     1:    1952     0      0  …      0       0       0      96       0       0   2048
   4      fpe0     2:    1952     0      0  …      0       0       0      96       0       0   2048
…
  14     fpe10     1:    1952     0      0  …      0       0      96       0       0       0   2048
  14     fpe10     2:    1952     0      0  …      0       0      96       0       0       0   2048
…
Core Utilization over preceding 3154.3457 seconds
-------------------------------------------------

      ID:       0       1  …    10      11      12      13      14      15
% PPE-RX:    4.29    4.54  …  4.46    4.44    0.00    0.00    0.00    0.00
    % PP:   15.26   16.45  … 16.39   16.31    0.00    0.00    0.00    0.00
    % RX:    0.00    0.00  …  0.00    0.00    5.19    6.03    0.00    0.00
    % TM:    0.00    0.00  …  0.00    0.00   31.17   35.17    0.00    0.00
  % COFF:    0.00    0.00  …  0.00    0.00    0.00    0.00    0.00    0.00
  % IDLE:   80.45   79.01  … 79.15   79.26   63.63   58.51   99.74   99.76
```

Hashing / distribution

Feature processing

Rx processing

Traffic Manager (Tx)

Crypto offload

Idle

Step 1c – Ingress processing

From the perspective the process running on the specific thread → detailed visibility.

```
C8500L-8S4X#show platform hardware qfp active datapath infrastructure sw-cio
Credits Usage:

  ID      Port  Wght  Global WRKR0  WRKR1  … WRKR10  WRKR11  WRKR12  WRKR13  WRKR14  WRKR15  Total
   1      rcl0     1:   6048      0      0  …       0       0      96       0       0       0   6144
   1      rcl0  128:   6048      0      0  …       0       0      96       0       0       0   6144
   2       ipc                                        0       0       0       0       0       0
   3 vxe_punti                                        0       0       0      46       0       0    512
   4      fpe0                                        0       0       0      96       0       0   2048
   4      fpe0                                        0       0       0      96       0       0   2048
…
  14     fpe10                                        0       0       0       0      96       0   2048
  14     fpe10                                        0       0       0       0      96       0   2048
…
Core Utilizatior
---------------
       ID:                                                  3      14      15
% PPE-RX:    4.29    4.54  …   4.46    4.44   0.00    0.00    0.00    0.00   Rx processsing
    % PP:   15.26   16.45  …  16.39   16.31   0.00    0.00    0.00    0.00   Feature processing
    % RX:    0.00    0.00  …   0.00    0.00   5.19    6.03    0.00    0.00   Rx processing
    % TM:    0.00    0.00  …   0.00    0.00  31.17   35.17    0.00    0.00   Traffic Manager (Tx)
  % COFF:    0.00    0.00  …   0.00    0.00   0.00    0.00    0.00    0.00   Crypto offload
  % IDLE:   80.45   79.01  …  79.15   79.26  63.63   58.51   99.74   99.76   Idle
```

If the credit values for `fpe0-fpeX` approach 0 then there are issues with ingress packet rate being too high for the platform.

Limiting the max packets from a given source in the fashion protects the system. It does not allow one elephant flow from one source to overwhelm the system and deny service to all other flows.

Step 1d – Ingress processing

```
C8500L-8S4X#show platform hardware qfp active datapath infrastructure sw-cio
Credits Usage:

  ID      Port                                  KR11  WRKR12  WRKR13  WRKR14  WRKR15  Total
   1      rcl0                                     0      96       0       0       0   6144
   1      rcl0                                     0      96       0       0       0   6144
   2       ipc                                     0       0       0       0       0      0
   3 vxe_punti                                     0       0      46       0       0    512
   4      fpe0                                     0       0      96       0       0   2048
   4      fpe0                                     0       0      96       0       0   2048
…
  14      fpe10                                    0       0       0      96       0   2048
  14      fpe10      2:   1952     0     0  …     0       0       0      96       0   2048
…
Core Utilization over preceding 3154.3457 seconds
-------------------------------------------------

      ID:       0       1  …     10      11      12      13      14      15
% PPE-RX:    4.29    4.54  …   4.46    4.44    0.00    0.00    0.00    0.00
    % PP:   15.26   16.45  …  16.39   16.31    0.00    0.00    0.00    0.00
    % RX:    0.00    0.00  …   0.00    0.00    5.19    6.03    0.00    0.00
    % TM:    0.00    0.00  …   0.00    0.00   31.17   35.17    0.00    0.00
  % COFF:    0.00    0.00  …   0.00    0.00    0.00    0.00    0.00    0.00
  % IDLE:   80.45   79.01  …  79.15   79.26   63.63   58.51   99.74   99.76
```

Low values for the %IDLE for the Rx/Tx cores does **not** absolutely indicate no more processing power for ingress packets. %IDLE can be zero, while there is still additional capacity available due to grouping of packets for egress queuing. If there is some idle % then there is clearly additional packet per second capacity.

Rx processsing
Feature processing
Rx processing
Traffic Manager (Tx)
Crypto offload
Idle

Step 1e – Tricky information


```
C8500L-8S4X#show platform resources r0 cpu
CPU utilization for five seconds: 71%, one minute: 71%, five minutes: 71%
Core 0: CPU utilization for five seconds:  1%, one minute:  2%, five minutes:  2%
Core 1: CPU utilization for five seconds:  1%, one minute:  2%, five minutes:  2%
Core 2: CPU utilization for five seconds: 100%, one minute: 100%, five minutes: 100%
Core 3: CPU utilization for five seconds: 100%, one minute: 100%, five minutes: 100%
Core 4: CPU utilization for five seconds: 100%, one minute: 100%, five minutes: 100%
Core 5: CPU utilization for                                    inutes: 100%
Core 6: CPU utilization for                                    inutes: 100%
Core 7: CPU utilization for                                    inutes: 100%
Core 8: CPU utilization for                                    inutes: 100%
Core 9: CPU utilization for                                    inutes: 100%
Core 10: CPU utilization fo                                    nutes: 11%
Core 11: CPU utilization fo                                    nutes: 11%
Core 12: CPU utilization fo                                    nutes:  2%
Core 13: CPU utilization fo                                    nutes:  2%
Core 14: CPU utilization fo                                    minutes: 100%
Core 15: CPU utilization fo                                    minutes: 100%
Core 16: CPU utilization for five seconds: 100%, one minute: 100%, five minutes: 100%
Core 17: CPU utilization for five seconds: 100%, one minute: 100%, five minutes: 100%
Core 18: CPU utilization for five seconds: 100%, one minute: 100%, five minutes: 100%
Core 19: CPU utilization for five seconds: 100%, one minute: 100%, five minutes: 100%
```

This output is from the perspective of underlying Linux OS. More detailed visibility is available from the show platform commands.

Linux can not see the difference in polling with active or idle results.

Steps 3 and 6 – Crypto

From the perspective the process running on the specific thread → accurate.

```
C8500L-8S4X#show platform hardware qfp active datapath infrastructure sw-cio
Credits Usage:

   ID      Port  Wght  Global WRKR0  WRKR1  … WRKR10  WRKR11  WRKR12  WRKR13  WRKR14  WRKR15  Total
    1      rcl0     1:    6048      0      0  …      0       0      96       0       0       0   6144
    1      rcl0   128:                                       0      96       0       0       0   6144
    2       ipc     1:                                       0       0       0       0       0      0
    3 vxe_punti     1:                                       0       0      46       0       0    512
    4      fpe0     1:                                       0       0      96       0       0   2048
    4      fpe0     2:                                       0       0      96       0       0   2048
…
   14     fpe10     1:    1952      0      0  …      0       0       0       0      96       0   2048
   14     fpe10     2:    1952      0      0  …      0       0       0       0      96       0   2048
…
Core Utilization over preceding 60.34 seconds
-----------------------------------------------
      ID:       0       1  …    10      11      12      13      14      15
% PPE-RX:    4.29    4.54  …  4.46    4.44    0.00    0.00    0.00    0.00
    % PP:   15.26   16.45  … 16.39   16.31    0.00    0.00    0.00    0.00
    % RX:    0.00    0.00  …  0.00    0.00    5.19    6.03    0.00    0.00
    % TM:    0.00    0.00  …  0.00    0.00   31.17   35.17    0.00    0.00
  % COFF:    0.00    0.00  …  0.00    0.00    0.00    0.00    0.00    0.00
  % IDLE:   80.45   79.01  … 79.15   79.26   63.63   58.51   99.74   99.76
```

No crypto in this setup so the crypto thread stay idle.

Rx processsing
Feature processing
Rx processing
Traffic Manager (Tx)
Crypto offload
Idle

Step 4,5 – Data plane processing

From the perspective the process running on the specific thread → high detail

```
C8500L-8S4X#show platform hardware qfp active datapath infrastructure sw-cio
Credits Usage:

  ID      Port  Wght  Global W                                  R12  WRKR13  WRKR14  WRKR15  Total
   1      rcl0     1:    6048                                   96
   1      rcl0   128:    6048                                   96
   2       ipc     1:       0                                    0
   3 vxe_punti     1:     466                                    0
   4      fpe0     1:    1952                                    0
   4      fpe0     2:    1952                                    0
…
  14     fpe10     1:    1952     0     0  …     0     0     0     0     96     0    2048
  14     fpe10     2:    1952     0     0  …     0     0
…
Core Utilization over preceding 60.34 seconds
-----------------------------------------------
      ID:     0     1  …    10    11    12    13    14    15
% PPE-RX:  4.29  4.54  …  4.46  4.44  0.00  0.00  0.00  0.00   Rx processsing
    % PP: 15.26 16.45  … 16.39 16.31  0.00  0.00  0.00  0.00   Feature processing
    % RX:  0.00  0.00  …  0.00  0.00  5.19  6.03  0.00  0.00   Rx processing
    % TM:  0.00  0.00  …  0.00  0.00 31.17 35.17  0.00  0.00   Traffic Manager (Tx)
  % COFF:  0.00  0.00  …  0.00  0.00  0.00  0.00  0.00  0.00   Crypto offload
  % IDLE: 80.45 79.01  … 79.15 79.26 63.63 58.51 99.74 99.76   Idle
```

Resources used for second pass parsing of traffic. Typically, higher if some traffic was IPSEC / GRE encapsulated.

PPE-Rx is only on S-FBD systems. This function is not included in NS-FBD systems.

Resources used for actual packet processing for forwarding traffic.

Step 5 – Data plane processing

```
C8500L-8S4X#show platform hardware qfp active fbd-flowdb balance distribution

PP Flow Distribution
                    Flows
      PP    0:      10252
      PP    1:      11260
      PP    2:      11268
      PP    3:      11268
      PP    4:      11268
      PP    5:      11264
      PP    6:      10742
      PP    7:      10742
      PP    8:      10746
      PP    9:      10746
      PP    10:     10758
      PP    11:     10758
```

Ideal distribution of traffic for a C8500L system. Maximum system performance in this scenario.

Even distribution amongst all data plane threads.

All of these flows sum up to 131072 which was the 256x256x2 based on the injected traffic.

This command is not available on NS-FBD or FBD systems.

Step 7a – Egress processing

```
C8500L-8S4X#show plat hardware qfp active datapath infrastructure bindings
Port Instance Bindings:

  ID       Port                 IOS Port WRKR12 WRKR13
   1       rcl0                     rcl0     Rx     Tx
   2        ipc                      ipc     Tx     Rx
   3  vxe_punti              vxe_puntif     Tx     Rx
   4       fpe0    GigabitEthernet0/0/0     Tx     Rx
   5       fpe1    GigabitEthernet0/0/1     Tx     Rx
   6       fpe2    GigabitEthernet0/0/2     Rx     Tx
   7       fpe3    GigabitEthernet0/0/3     Tx     Rx
   8       fpe4    GigabitEthernet0/0/4     Rx     Tx
   9       fpe5    GigabitEthernet0/0/5     Tx     Rx
  10       fpe6    GigabitEthernet0/0/6     Rx     Tx
  11       fpe7    GigabitEthernet0/0/7     Tx     Rx
  12       fpe8 TenGigabitEthernet0/1/0     Rx     Tx
  13       fpe9 TenGigabitEthernet0/1/1     Tx     Rx
  14      fpe10 TenGigabitEthernet0/1/2     Rx     Tx
  15      fpe11 TenGigabitEthernet0/1/3     Tx     Rx
```

Step 7b – egress processing

From the perspective the process running on the specific thread → more visibility.

```
C8500L-8S4X#show platform hardware qfp active datapath infrastructure sw-cio
Credits Usage:

  ID      Port      KR11   WRKR12   WRKR13   WRKR14   WRKR15   Total
   1      rcl0         0       96        0        0        0    6144
   1      rcl0         0       96        0        0        0    6144
   2       ipc         0        0        0        0        0       0
   3 vxe_punti         0        0       46        0        0     512
   4      fpe0         0        0       96        0        0    2048
   4      fpe0         0        0       96        0        0    2048
…
  14     fpe10         0        0        0       96        0    2048
  14     fpe10         0        0        0       96        0    2048
…
Core Utilization over preceding 3154.3457 seconds
-------------------------------------------------

      ID:       0       1  …     10       11       12       13       14       15
% PPE-RX:    4.29    4.54  …   4.46     4.44     0.00     0.00     0.00     0.00
    % PP:   15.26   16.45  …  16.39    16.31     0.00     0.00     0.00     0.00
    % RX:    0.00    0.00  …   0.00     0.00     5.19     6.03     0.00     0.00
    % TM:    0.00    0.00  …   0.00     0.00    31.17    35.17     0.00     0.00
  % COFF:    0.00    0.00  …   0.00     0.00     0.00     0.00     0.00     0.00
  % IDLE:   80.45   79.01  …  79.15    79.26    63.63    58.51    99.74    99.76
```

Low values for the %IDLE for the Rx/Tx cores does not absolutely indicate no more processing power for ingress packets. %IDLE can be zero, while there is still additional capacity available. If there is some idle then there is clearly additional packet per second capacity.

Packets can be bundled for release which can increase efficiency but still show the same level of utilization.

Rx processsing
Feature processing
Rx processing
Traffic Manager
Crypto offload
Idle

Summary on troubleshooting x86 performance

- Use “**`sw-cio`**” output:

  To get I/O (Rx/Tx) core utilization guidance

  To get data plane thread utilization

  To get crypto thread utilization

- Use “**`show plat resources r0 cpu`**”:

  To get control plane core utilization guidance

  To get service plane core utilization guidance

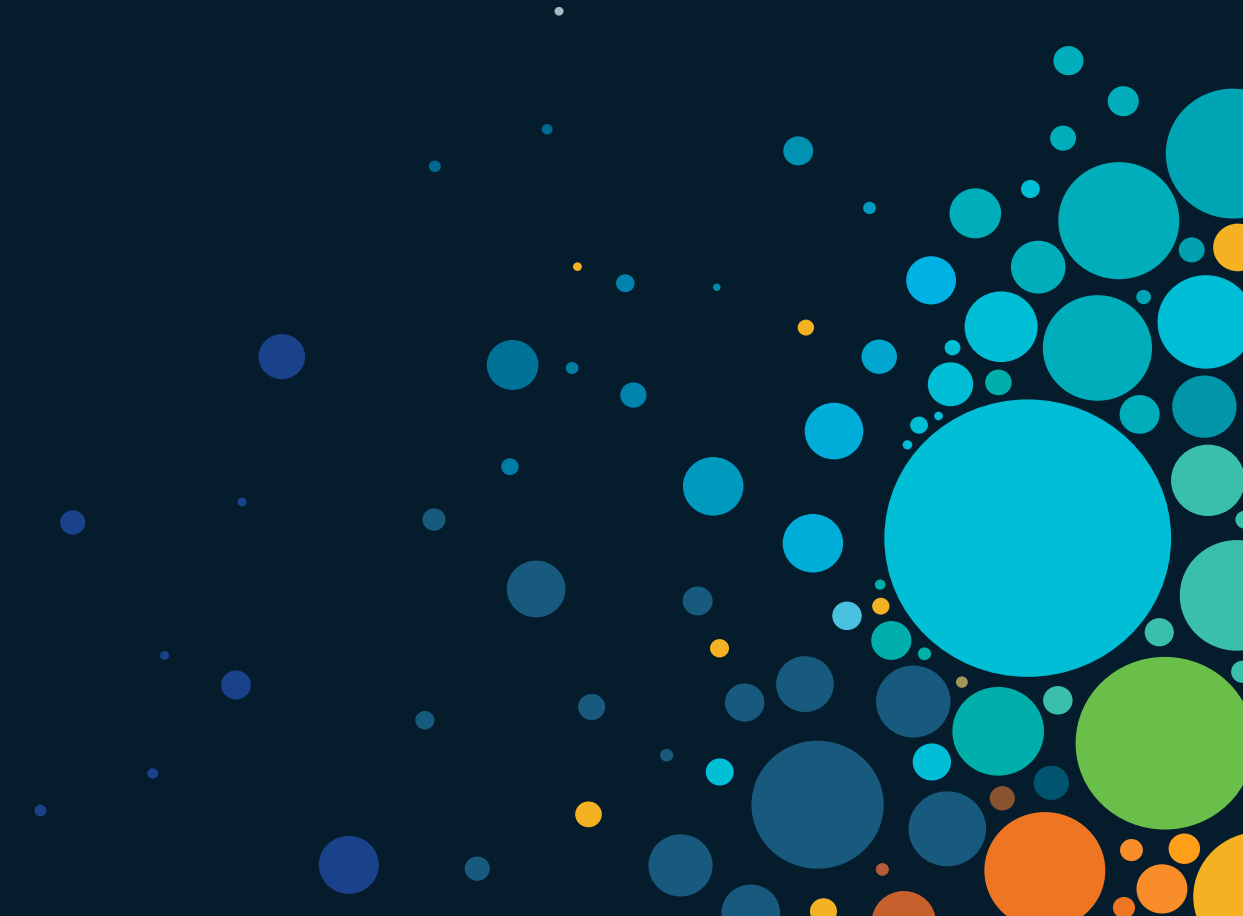
- Use “**`show proc cpu`**”:

  To get IOSd thread utilization

Network management access to x86 performance

- Use “**`sw-cio`**” output:
  - To get I/O (Rx/Tx) core utilization guidance — No SNMP/YANG method
  - To get data plane thread utilization
    - `ceqfpUtilProcessingLoad` — `1.3.6.1.4.1.9.9.715.1.1.6.1.14`
  - To get crypto thread utilization
    - `ciscoEntityPerformanceMIB` — `1.3.6.1.4.1.9.9.756`
- Use “**`show plat resources r0 cpu`**”:
  - To get control plane core utilization guidance
  - To get service plane core utilization guidance
- Use “**`show proc cpu`**”:
  - To get IOSd thread utilization

L2 and L3 forwarding



Wide range of L3 forwarding in the platform

- All L3 forwarding is via the data plane cores
- There is no L3 forwarding via the control plane threads
- Only traffic destined for the router itself will be punted to the control plane threads

L3 forwarding options

- IPv4 via CEF
- IPv6 via CEF
- MPLS via CEF
- L3VPN

Full hashing in Strict Flow-based platforms.

Maximum performance with flow diversity.

- Tunnels
  - GRE
  - IPSEC

If router is the tunnel source or destination, full hashing in strict flow-based platforms.
If router is a transit router for the tunnel, elephant flows could be a concern with small number of tunnels (outermost header hashing).

Maximum performance with multiple tunnels and flow diversity in the tunnel.

L2 forwarding options

- With L3 underlay:
  - OTV
  - L2VPN
  - L2TP
  - EVPN
  - VxLAN

EVPN L3 gateway
Cross-leaf VLAN over L2 VNI

- Native L2 forwarding
  - L2 bridging
  - L2 port connect
  - L2 switching on C8300 platforms with switch modules

L2 forwarding options

- With L3 underlay:
  - OTV
  - L2VPN
  - L2TP
  - EVPN
  - VxLAN
- Native L2 forwarding
  - L2 bridging
  - L2 port connect
  - L2 switching on C8300 platforms with switch modules

Ingress L2 side traffic uses L2 for hashing. **Incoming remote OTV traffic hashed against remote OTV overlay IP address.**

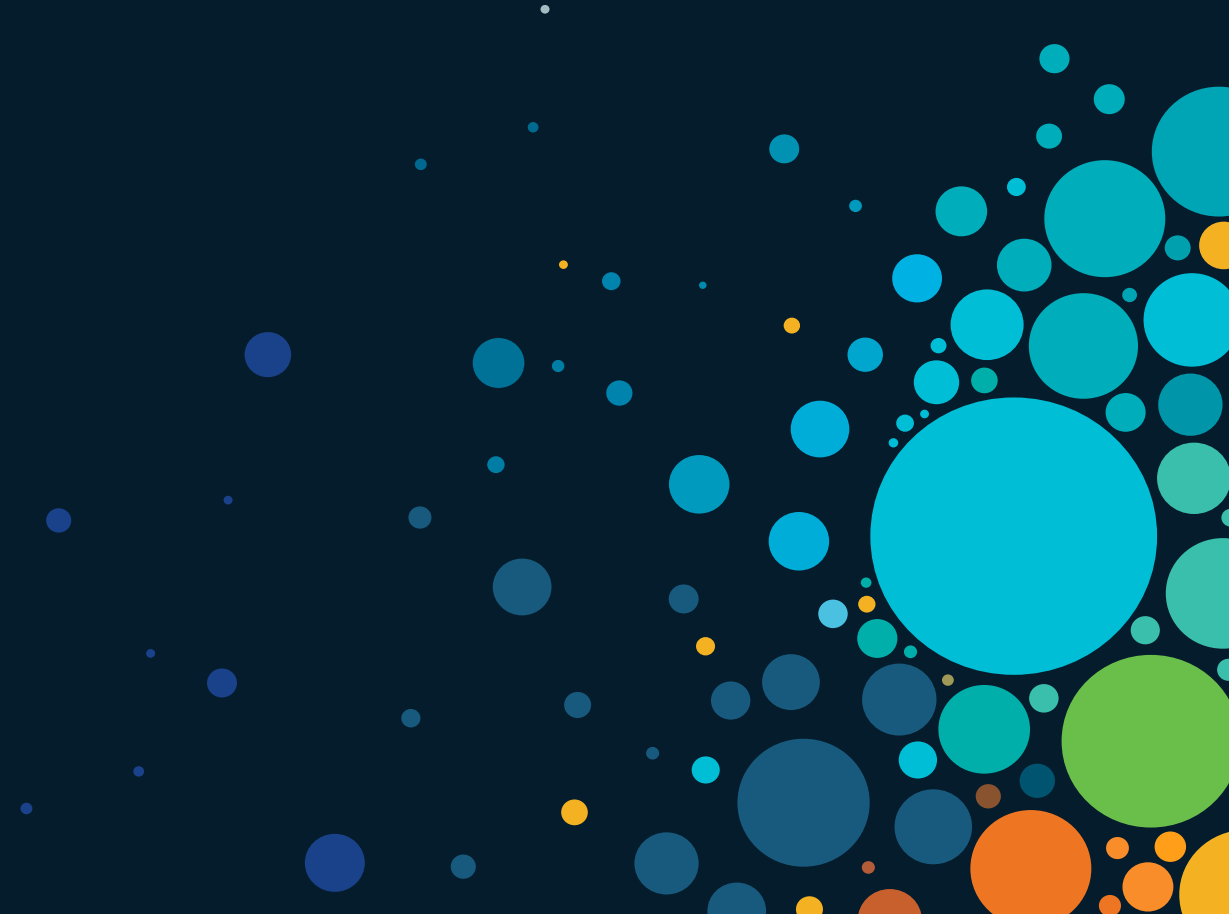
Ingress L2 side traffic uses L2 for hashing. **Incoming remote traffic hashed against remote IP encapsulation source address.**

Ingress L3 addressing for hashing of VxLAN encapsulated traffic.

L2 addresses always used for hashing to data plane cores.

Not part of the router dataplane.

Conclusion



Key takeaways

- Feature and functional compatibility across the portfolio
- Understand the forwarding mechanism for the platform before you dive into troubleshooting
  - When you suspect a bottleneck, use the sw-cio output to begin your troubleshooting process (except QFP platforms)
- C8500L requires a diversity of flows for maximum performance
  - Inner packet inspection is supported for GRE, IPSEC, and SDWAN tunnels
- IOS XE x86 platforms have flexibility for core allocation to suit your network needs

Complete your Session Survey

- Please complete your session survey after each session. Your feedback is important.
- Complete a minimum of 4 session surveys and the Overall Conference survey (open from Thursday) to receive your Cisco Live t-shirt.

- All surveys can be taken in the Cisco Events Mobile App or by logging in to the Session Catalog and clicking the "Attendee Dashboard" at https://www.ciscolive.com/emea/learn/sessions/session-catalog.html

Continue Your Education

Visit the Cisco Showcase for related demos.

Book your one-on-one Meet the Engineer meeting.

Attend any of the related sessions at the DevNet, Capture the Flag, and Walk-in Labs zones.

Visit the On-Demand Library for more sessions at ciscolive.com/on-demand.


cisco
The bridge to possible
Thank you
cisco Live!


CISCO Live!
ALL IN