Receive Packet Scheduling

Technical Guide

Introduction

This guide describes Receive Packet Scheduling and how to configure it. Receive Packet Scheduling is the mechanism by which packets requiring software forwarding are distributed to individual cores in multi-core CPUs.

The performance of software packet processing is directly related to the number of instructions per second the devices processor is capable of. This depends on the number of Instructions per Cycle (IPC) of the architecture and the core clock speed. Hence the simplest mechanism for improving performance has traditionally been to increase the CPU clock speed.

Unfortunately silicon fabrication limitations has meant that CPU speeds are no longer increasing and an alternative solution is required. That solution is multi-core processors. The ultimate goal of system designers is to balance processing load across each of the cores with the smallest possible load balancing overhead.

What's in this guide?

The guide begins with a brief look at how receive packet scheduling works and answers some Frequently Asked Questions.

This is followed by a description of the different scheduling methods and the pros and cons for each. AlliedWare Plus™ multi-core devices use the hash-based receive packet scheduling method by default. This is the most accepted method for splitting packet processing across multi-core processors. It also provides the best performance and ensures packet ordering for real world traffic.

The guide concludes with some simple configuration examples and suggestions on how best to use the **receive-packet-scheduler** command options: hash, balanced, and split.

Contents

Products and software version that apply to this guide

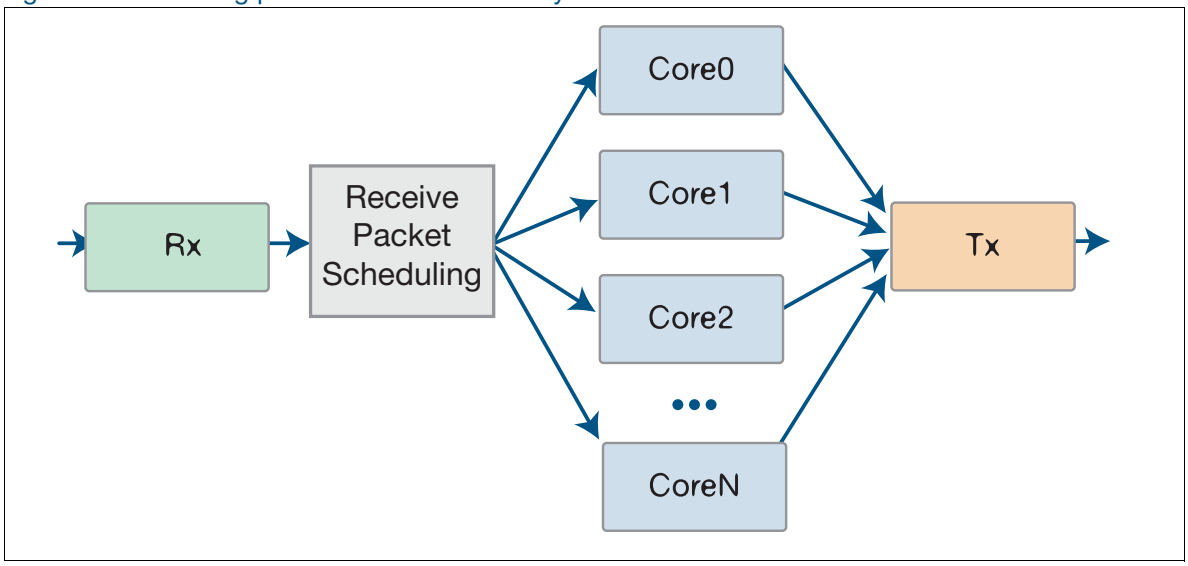
This guide applies to the following AlliedWare Plus AR series products running software version 5.4.8-2.1 or later.

- AR2050V
- AR3050S
- AR4050S

What is Receive Packet Scheduling?

Receive Packet Scheduling distributes packets to each core in a multi-core system.

Figure 1: Distributing packets to a multi-core system



AlliedWare Plus uses a flow hash-based scheme to ensure packets from the same flow are processed in order on the same core. This is generally accepted as the best compromise between efficiency and stability for the most common network traffic.

There are however a few scenarios where a different mechanism may be required. You can configure alternative packet scheduling algorithms to suit your traffic patterns. See, "Configuring Receive Packet Scheduling on AlliedWare Plus Devices" on page 9.

Note: It is very unlikely that there would be any need to change from the default receive-packet-scheduling scheme (hash) as it is the most suitable mechanism for real network traffic.

Frequently Asked Questions

Why does my performance test only use one core of my multi-core router?

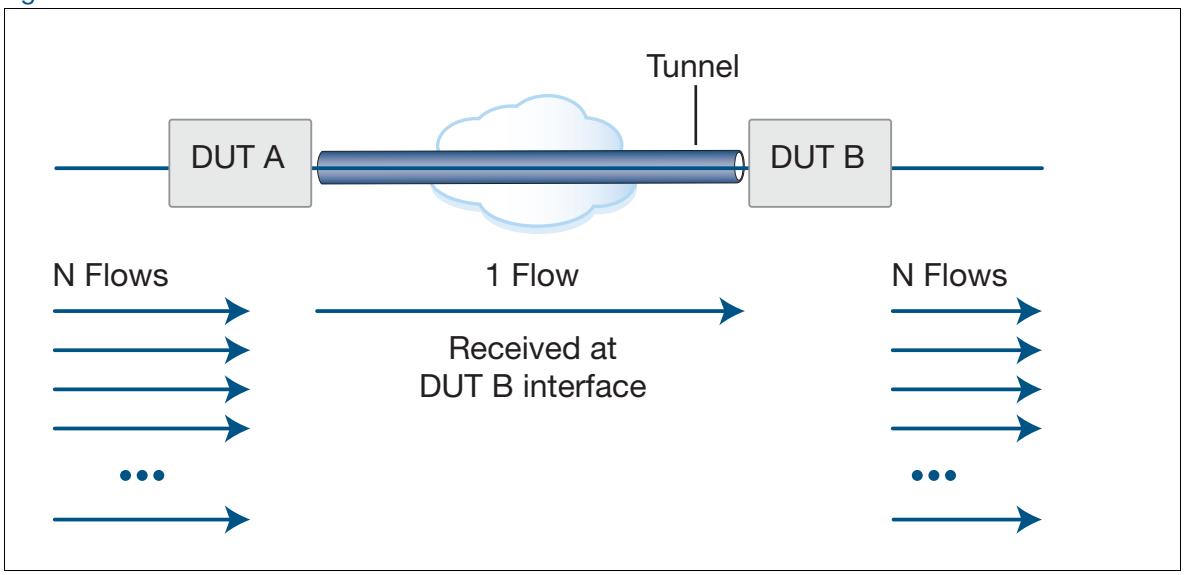
It is common for people testing the packet processing performance of software forwarders to naively create a test setup using a single packet flow. This gave the best performance when testing single core devices as it also avoided the overhead associated with local flow context storage. Often performance testing tools default to a single flow of traffic. On multi-core devices that test only utilizes one of the cores in the device and does not provide a good indication of how the device will perform in the real world.

Modern performance testing for multi-core devices requires a good balance of flows to the number of cores. If the detail of the flow hashing algorithm used is well known, then the test setup can be configured with a single flow per core. Commonly a test setup is created with at least 4 times the number of flows to the number of cores.

Why does my VPN tunnel only decrypt on a single core?

Historically IPsec performance was tested by using two identical devices as the endpoints of a VPN tunnel. The problem with this setup is that even if multiple flows of end user traffic are used, the DUT that is decrypting the IPsec traffic only sees a single encrypted packet flow and hence only processes those packets on a single core. Even though the device is capable of encrypting using multiple cores, due to the symmetry in the test, the decrypting device becomes the bottleneck and determines the performance result.

Figure 2: VPN Tunnel



The obvious answer to this is to make sure the decrypting device is receiving multiple encrypted packet flows by configuring multiple tunnels. This is common in the real world for fully-meshed VPNs where the device is connected to many remote sites or for central site devices that process the traffic from many branch offices. Another alternative is to list the single tunnel encryption performance separately to the single tunnel decryption performance.

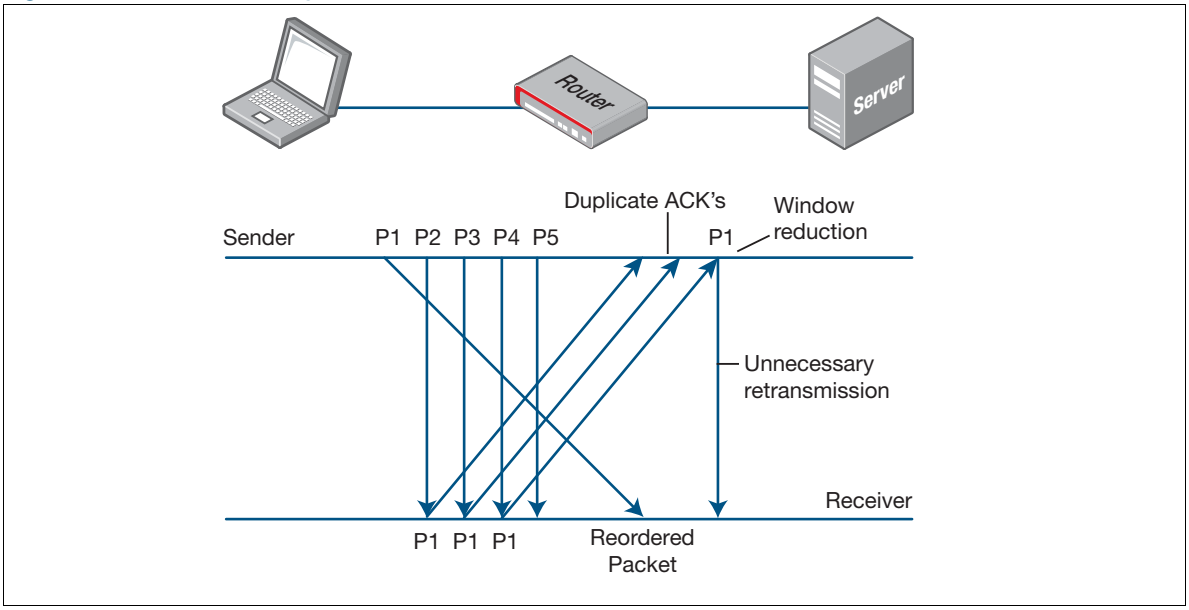
There is however one scenario where configuring multiple tunnels is not realistic in the real world. If a branch office VPN device routes all traffic via the central office and users predominantly perform downloads from the central site, then all of that downloaded traffic will be decrypted on a single core of that device.

Why is packet ordering important?

Most higher layer protocols can handle out-of-order packets, but it comes at a cost. Protocols that are robust to packet re-ordering attempt to hide the problem from the end user. There is usually a noticeable “network slowdown” as transmission windows reduce and are resent or end device stacks are bogged down reassembling packet streams.

Protocols that are not robust to out-of-order packets can fail catastrophically, usually by aborting a transfer or even worse by accepting the out-of-order packets assuming they are in the correct sequence. Hence while packet balancing across cores using round-robin or random distribution provides fantastic performance test results, it cannot be used in the real world.

Figure 3: **Packet ordering**



Why do packets from the same flow need to be processed on the same core?

Historically, intermediary devices only needed to look at Layer 2 or Layer 3 headers to decide what to do with a packet. It is now common for packet forwarding devices to use Deep Packet Inspection (DPI) to differentiate between higher layer applications. More often than not, DPI must see more than one packet to interpret the packet stream.

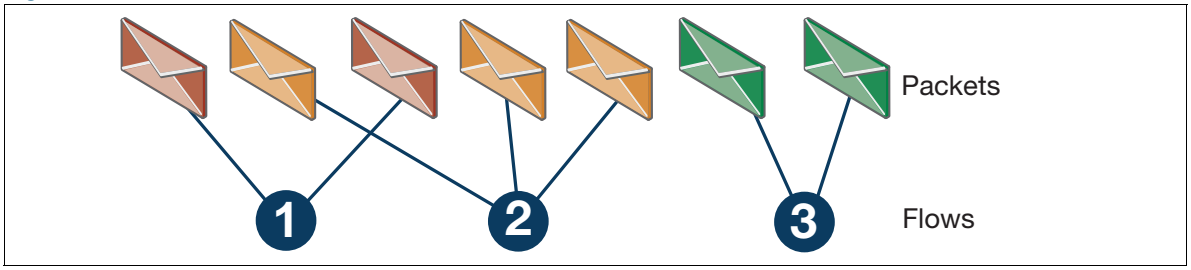
This requires that a context is maintained on the forwarding device for the entire stream of packets. While it is possible to share that context across multiple cores, the best memory/cache utilization is achieved when packets from the same flow are processed in order on the same core.

What are IP flows?

To meet the requirements of packet ordering and local context, packets classified as being part of the same stream must be sent to the same core for processing. An IP flow is defined as all packets that share the same 5-tuple, made up of the source and destination IP addresses, protocol, and source and destination ports.

This not only ensures end-user application packets are processed in order, but also conveniently provides a real-world higher layer mechanism for evenly distributing packets across cores. In real network traffic, there can be 100's if not 1000's of different IP flows being processed by a forwarding device at any time, which provides an ample spread of packets across cores. Commonly the values of the 5-tuple are hashed together to reduce the size of the flow data and provide faster core lookup.

Figure 4: Flow hash



Receive Packet Scheduling Methods

There are several different methods of receive packet scheduling, they include:

- Single core interrupt handling
- Multiple core interrupt handling
- Balanced packet sharing
- Hash packet scheduling
- Split-core packet scheduling

Let's look at the pros and cons for each method:

Single core interrupt handling

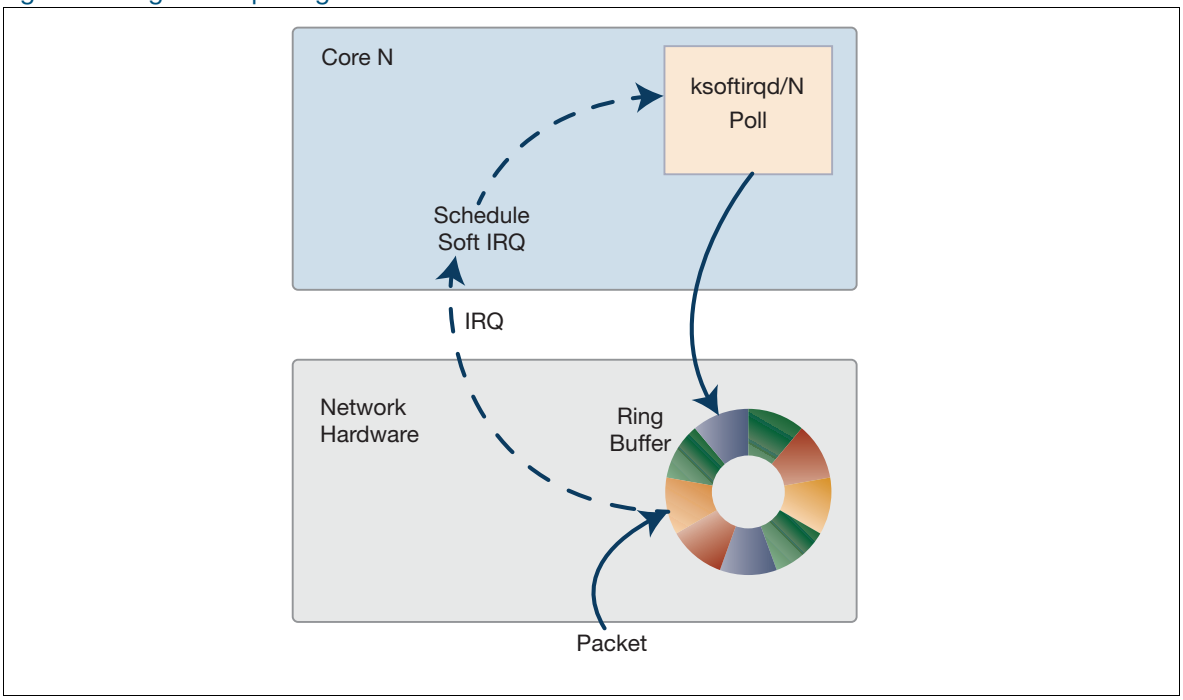
On software forwarders, the receive hardware generates a processor interrupt request (IRQ) when it receives a packet. The processor immediately stops what it is currently doing and services the interrupt.

In simple software forwarders, the core that services the IRQ:

- finishes the packet reception process
- actions the packet forwarding software
- and finally, triggers the send hardware to transmit the packet.

In single core systems there is only one core to process the IRQ, one core to process the packet, and hence no need to do anything more complicated at all. To avoid the overhead of being continuously interrupted, most systems turn off interrupts and revert to polling for new packets when the rate of generated interrupts increases to some threshold.

Figure 5: Single core polling



Multiple core interrupt handling

The simplest approach for sharing packet processing across multiple cores is to share the packet receive interrupts across the cores. In two core, two interface systems, each core can be configured to pick up packets from only one of the interfaces. If an equal number of packets was received on one interface as the other, then there should be a good balance of traffic across cores.

However, real-world traffic is never that symmetrical, and with some devices the number of interfaces does not match the number of cores!

Balanced packet sharing

Another simple approach is for the receive hardware to either randomly or round-robin notify cores of packets to be processed. This mechanism is implemented differently by various types of hardware, but the end result is that the received packets are balanced across the available cores.

Either way it is implemented, the reason this mechanism is not used in the real world is because it cannot guarantee packet ordering.

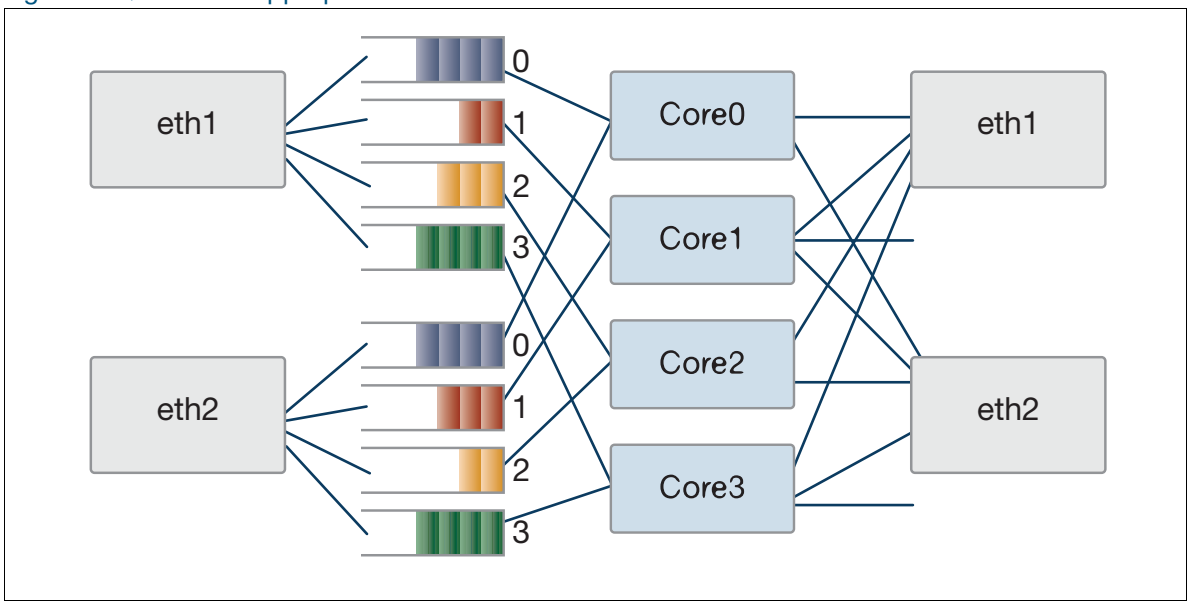
Note: The hardware on the AR series products balances packets across cores by hashing the first 128 bytes of the packet and using the bottom bits to determine the processing core. That means that if in test environments the first 128 bytes are identical between packets, all packets would end up on the same core.

Hash packet scheduling

To ensure packets from the same flow are processed on the same core, the packet reception process must find the IP header in the packet, hash the IP flow tuple, determine which core the packet should be processed on, and finally queue the packet for the appropriate core. If the packet is already on the correct core, then the packet can be processed as per normal. This process provides good balancing across cores, but comes with the added cost of parsing and hashing on the original core as well as a loss of data cache utilization if the packet is shifted to a different core.

Modern networking hardware has been enhanced to offload this process in an attempt to achieve zero overhead for hash based receive packet scheduling. Packets are parsed in hardware, a flow hash is generated in hardware, and the hardware provides multiple receive queues per interface. Each core need only pick up packets from the receive queues assigned to that core.

Figure 6: Queues for appropriate cores



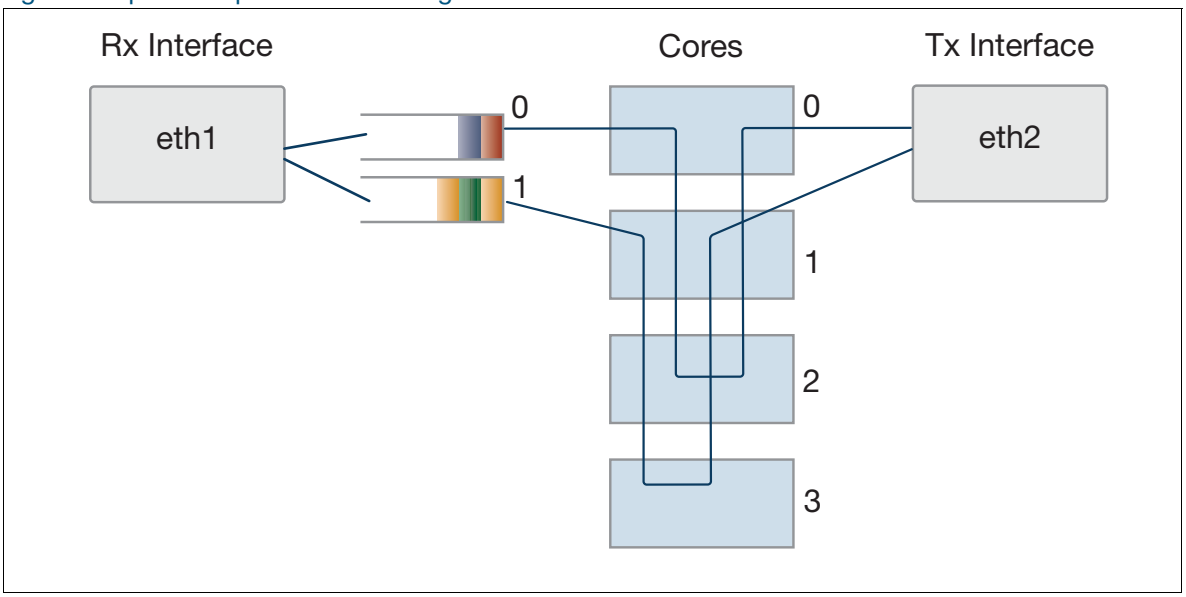
Real-world network traffic is made up of many different flows. Hence flow hash-based receive packet scheduling is ideal for almost all scenarios.

Split-core packet scheduling

When processing VPN packets the biggest computational process required is always the encryption/decryption process. This often equates to approximately 50% of all the CPU cycles required to forward a packet over a VPN tunnel.

If instead of hashing packets to all the available cores we only pass received packets to half of the cores then we can reserve the other half of the cores for the encryption/decryption process. Even with the overhead of scheduling the crypto processing across the other cores, this mechanism provides single tunnel decryption of approximately 80% of the potential multiple tunnel decryption capability of the device and only a small drop in the encryption performance.

Figure 7: Split-core packet scheduling



Configuring Receive Packet Scheduling on AlliedWare Plus Devices

The AlliedWare Plus AR series of devices forward packets in software. The AR series differ by their number of CPU cores, CPU speed, and the amount of available memory:

- AR1050V - 1 800MHz MIPS64 core, 512MB RAM
- AR2050V - 2 800MHz MIPS64 cores, 512MB RAM
- AR3050S - 2 800MHz MIPS64 cores, 1024MB RAM
- AR4050S - 4 1500Mhz MIPS64 cores, 2048MB RAM

With the exception of the AR1050V, all of these products contain multi-core processors and hence require some mechanism for processing packets on all available cores.

To configure packet processing across cores, use the command **receive-packet-scheduler**:

```
awplus(config)# receive-packet-scheduler {hash | balanced | split}
```

Let's look at each of the three scheduling options:

Hash AlliedWare Plus devices use hash-based receive packet scheduling by default. This is the most accepted method for splitting packet processing across multi-core processors. It provides the best performance and ensures packet ordering for real world traffic.

To set the hash scheme, use the commands:

```
awplus# configure terminal
awplus(config)# receive-packet-scheduler hash
```

Hash is the default setting for the command. If the receive packet scheduling scheme has been changed from the default it can be set back to hash using the command:

```
awplus(config)# no receive-packet-scheduler
```

Balanced Packets are balanced across cores as efficiently as possible providing the best performance for single flow scenarios. This comes at the cost of no guarantee of packet ordering.

To set the balanced scheme, use the following command:

```
awplus(config)# receive-packet-scheduler balanced
```

Note: Use this scheduling scheme only if you fully understand the consequences of out-of-order packets. It is likely to cause difficult to diagnose networking issues such as random network slow downs and unexpected application behavior.

Split Half of the CPU cores in a multi-core device are reserved for packet processing. These cores process packets using the default hash based scheme. The other half of the processing cores are reserved for the IPsec encryption/decryption process. After the encryption/decryption process is complete, the packets are forwarded by the same core the packet was received on ensuring packet ordering is maintained.

To set the split scheme, use the following command:

```
awplus(config)# receive-packet-scheduler split
```

Note: Use this scheme on a branch office router where all network traffic is routed via a VPN to a central site and where the direction of the data transfer is predominately downloads from the central site.