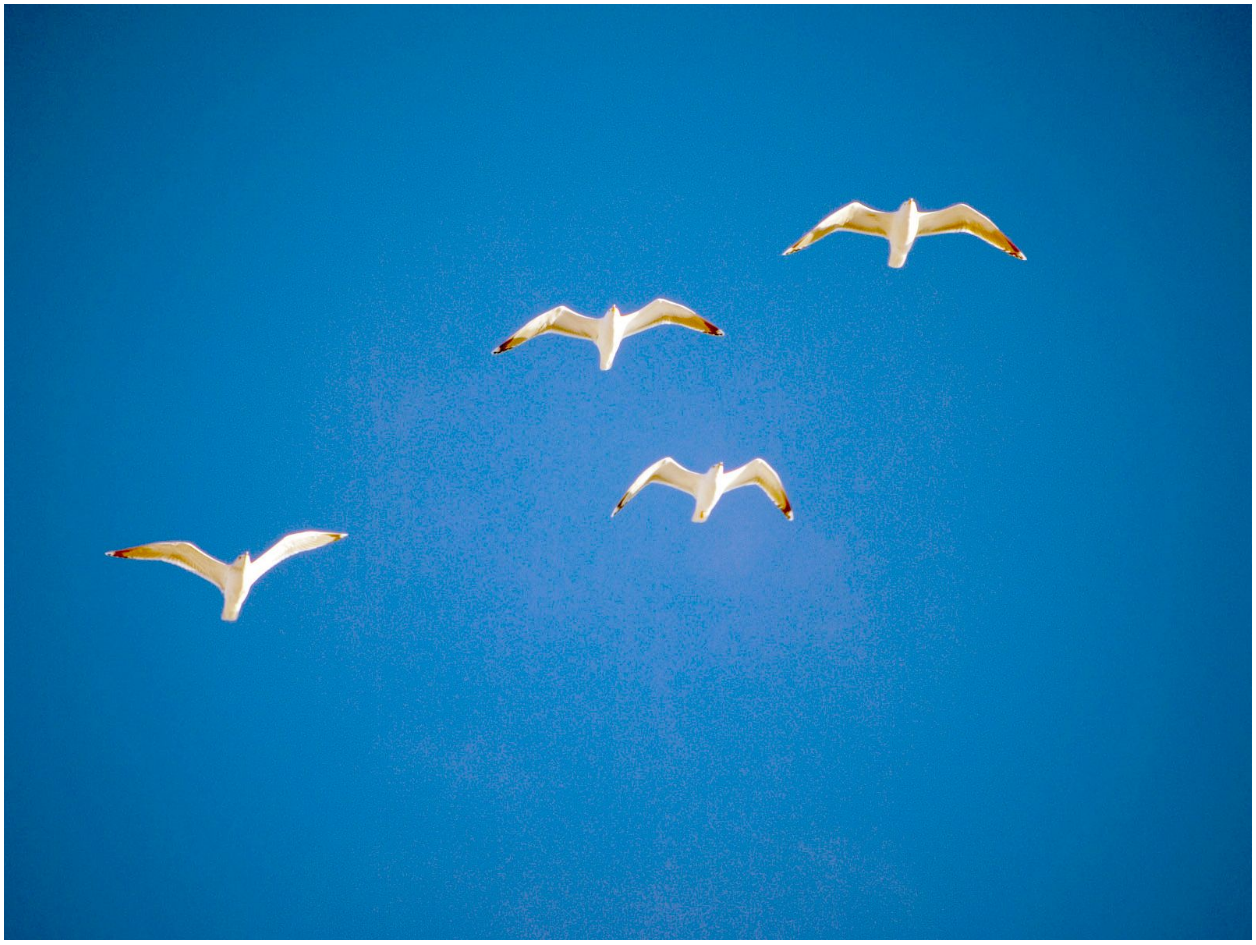

The Third Generation of IP Routing Suite

27.07.2020

Flock Networks Ltd
160 Kemp House,
London, EC1V 2NX

Overview

The first generation of IP routing suite was created around 40 years ago, as the concept of packet based forwarding replaced circuit switching. Its main design goal was efficiency, as it had to run in a very restrictive hardware environment. It had a monolithic design and could only handle tasks sequentially.

Around 20 years ago preemptive multitasking had become the norm and CPU's were moving to support multiple cores. In response the second generation of IP routing suite broke the existing monolith into components that could be run in parallel in independent processes.

Now with the maturity of the Rust language[1] there has been a revolutionary change in systems programming. The Rust memory safety guarantee enables the third generation of IP routing suite. The third generation IP routing suite has the efficiency of the first generation and the parallelism of the second generation. This combination enables a large increase in performance whilst reducing power consumption.

IP Routing Suite Components

Regardless of which generation, the main components of an IP routing suite are:

- RIB (Routing Information Base)
    - The RIB is the arbitrator of routes learnt from different sources.
    - The RIB receives routing updates from routing protocols and from the operating system kernel.
    - The RIB determines each best route and redistributes that route to the dataplane and to other components as specified in the configuration.
- Routing Protocols
    - An IGP (Interior Gateway Protocol) learns and advertises routes within a network / autonomous system.
        - OSPF and IS-IS have become the most commonly used IGP's.
    - An EGP (Exterior Gateway Protocol) learns and advertises routes from outside of the network / autonomous system.
        - BGPv4 has become the EGP de facto standard.
- Operations API
    - Used to configure the router.
    - Used to view state and stream telemetry data from the router.

These components have been abstracted as they each perform a distinct task. The components also need a high level of communication between each other as the output of

[1] https://en.wikipedia.org/wiki/Rust_(programming_language)

one component becomes the input for another. For example if BGP receives 10,000 routes from a neighbor then it must choose the best route to each destination and send each best route to the RIB. For each route received RIB must then determine its best route and send that to the dataplane and possibly redistribute a subset of the routes to OSPF. At the same time an operator might be using the API component to view the state of the OSPF component, and the RIB component might be streaming telemetry data via the API to a remote management system.

The RIB needs to reach a converged state, and program the dataplane in a timely manner, or IP traffic will be dropped or sent in the wrong direction.

It is the method by which components exchange information, that is the main indicator of which generation the IP routing suite belongs to.

The first generation of IP routing suite c.1980 onwards

Compared to today, the first generation of IP routing suite ran on very low end hardware. The expectation was that there would be a single 32 bit CPU core available, running at around 16MHz and available RAM limited to around 256KB. This led to a very basic and efficient design.

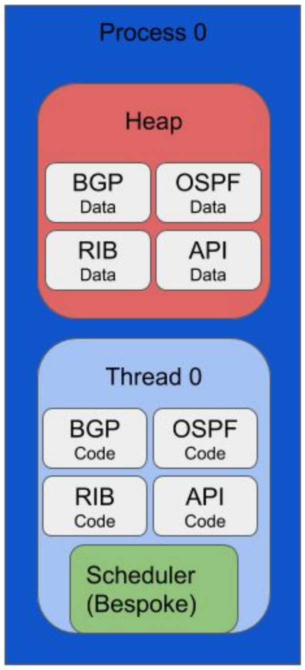

Figure 1: First Generation IP Routing Suite Overview

All the IP routing suite components ran in a single process and the process also contained a bespoke scheduler which decided when each component was allowed to run. The scheduler was light weight and relatively simple, following a “run to completion model”.

When a component was allowed to run, it ran for as long as it required or as long as it felt was reasonable. Then the running component had to cede control back to the scheduler.

Since the IP routing suite ran in a single process, there was only a single memory heap. All components could access all locations in memory. This had the advantage of allowing very fast messaging passing between the components. One component could simply pass the memory address of the message and the message size to another component, and the other component could access the message. The IP routing suite was most likely written in C, so the team of developers had the responsibility to manage the ownership of the message between and even within components. As the size and complexity of this monolithic implementation grew, large engineering challenges were encountered [2] [3].

Examples of the first generation of IP routing suite are Cisco IOS® [4] and BIRD[5].

The second generation of IP routing suite c.2000 onwards

20 years later thanks to Moor's law [6] the hardware available had become much more powerful. 64 bit CPUs became mainstream, running at around 1GHz. CPU's with multiple cores became common during this decade. Typical available RAM had increased one thousand fold to around 256MB.

The operating systems had been updated to utilize this new hardware. Support for preemptive multitasking was enabled by the inclusion of a preemptive scheduler [7]. Running processes concurrently was now the norm.

[2] https://en.wikipedia.org/wiki/Dangling_pointer
[3] https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/
[4] https://en.wikipedia.org/wiki/Cisco_IOS
[5] https://en.wikipedia.org/wiki/Bird_Internet_routing_daemon
[6] https://en.wikipedia.org/wiki/Moore's_law
[7] https://en.wikipedia.org/wiki/Preemption_(computing)

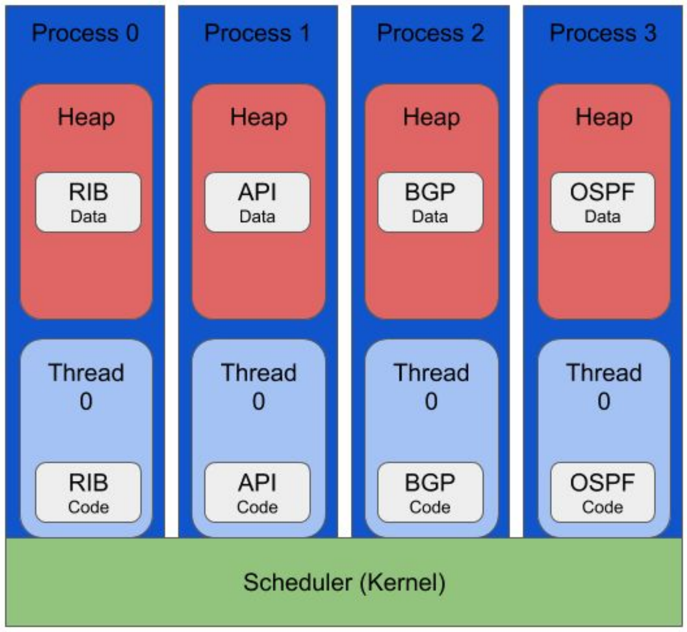

Figure 2: Second Generation IP Routing Suite Overview

The second generation of IP routing suite could now run components concurrently by placing each in its own process. These processes were now scheduled by the operating systems built in scheduler.

By running in a separate process, each component had its own memory heap, which was protected from inadvertent access by another component by the operating system using the memory protection built into the hardware. Message passing between components was now performed via Inter-Process Communication.

The second generation of IP routing suite is the most commonly implemented. Examples are Juniper Junos® [8], Cisco IOS XR® [9] and FRR[10].

[8] https://en.wikipedia.org/wiki/Junos_OS
[9] https://en.wikipedia.org/wiki/Cisco_IOS_XR
[10] https://en.wikipedia.org/wiki/FRRouting

The third generation of IP routing suite 2020 onwards

20 years later and the hardware has continued to improve. CPU clock rates have started to level off to around 2 - 4 GHz and performance gains are now made by increasing the number of CPU cores. CPUs with around 32 cores are becoming commonplace. RAM size has continued to grow exponentially and 64 GB is becoming commonplace.

There has also been a revolution in system programming language design. Traditionally, for performance reasons there have only been two widely used system programming languages, C and C++. C and C++ are not memory safe. That is, a C or C++ program can do arbitrary arithmetic on memory addresses and have direct access to the resulting address. This is very useful, flexible and efficient but leads to a design where components of an application need to be placed in different processes, so the operating system can provide memory protection between the components. This is the design we see in the second generation IP routing suite.

In the decade prior to 2020, the Rust language was invented and became the first memory safe systems programming language. The memory protection in Rust is guaranteed at compile time, so the resulting application still has a performance similar to C or C++. The memory protection inherent in Rust enables the third generation IP routing suite.

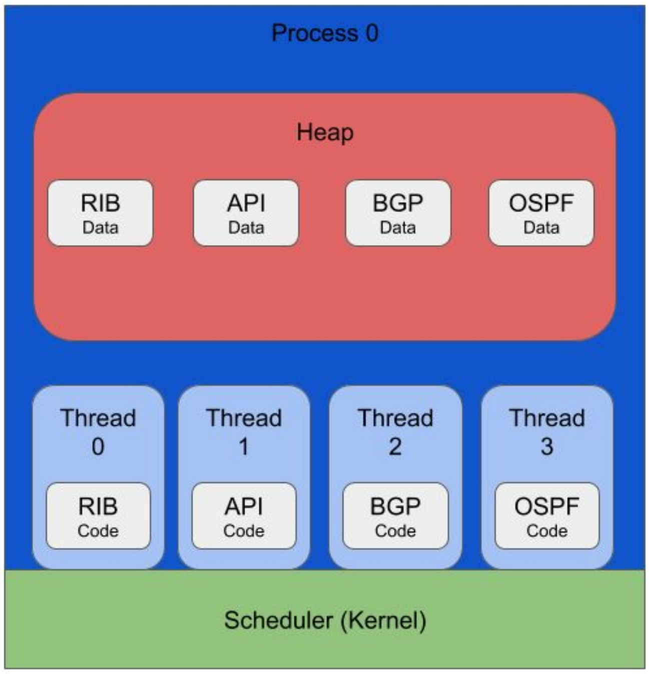

Figure 3: Third Generation IP Routing Suite Overview

Now that each component's heap no longer needs to be isolated by the operating system, all the components can be moved back into a single process. Parallel execution is achieved by placing each component into its own thread. Each thread can access the entire shared heap, and the Rust language provides memory protection between the threads. Each thread is scheduled by the operating system's scheduler.

The third generation IP routing suite combines the advantages of the previous two generations. From the second generation we get the advantages of parallel execution and operating system scheduling. From the first generation, since we only have a single memory heap, components can again pass messages by only referring to a memory address, rather than by copying data via Inter-process communication. Since IP routing suite components exchange so much messaging, the performance impact of this is huge.

Currently the Flock Networks IP routing suite is the only example of this third generation implementation.

Comparison of the three generations of IP routing suite

	First Generation	**Second Generation**	**Third Generation**
Component Scheduling	Bespoke	OS Kernel	OS Kernel
Parallelism	None	Process per component	Thread per component
Memory Protection	None	OS Kernel	Compile time
Message Passing	Pass memory address + size	Copy data via IPC	Pass reference to object

Message passing in detail

A common operation in an IP routing suite would be to learn a relatively large number of routes from a BGP neighbor and program the RIB with the subset of routes that is considered best. Below we show how each generation of IP routing suite would handle this update. We assume that of all the routes learnt from the BGP neighbor, 10,000 of them qualify to be sent to the RIB.

First Generation Message Passing

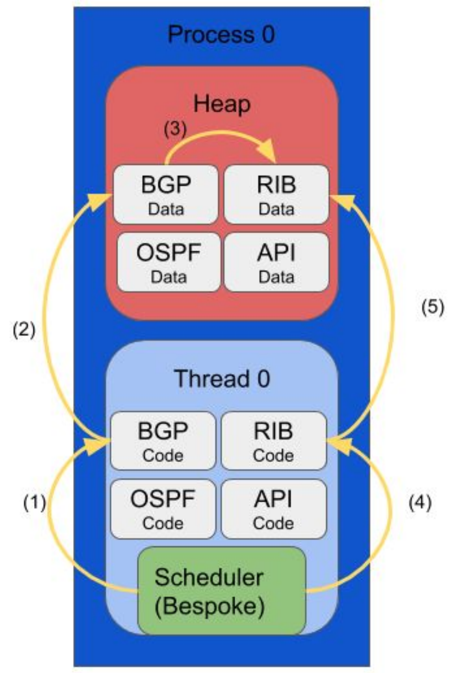

Figure 4: First Generation Message Passing Sequence

(1) The scheduler wakes the BGP component.

(2) The BGP component learns routes from its neighbor and decides 10,000 route updates need to be sent to the RIB component.

(3) The BGP component writes a message into the RIB components memory, telling the RIB the starting memory address and size of the new route updates.

(4) The scheduler suspends the BGP component and wakes the RIB component.

(5) The RIB component notices the message from the BGP component and dereferences the start address to find the route updates. The RIB component installs these routes into the RIB table.

Points of interest:

- The size of the message passed at (3) is tiny. On a 64 bit CPU it will be an 8 byte pointer + 4 bytes of size information. The 10,000 route updates remain at the same memory address.
- At (4) the BGP component is suspended so any other incoming requests, such as BGP neighbor route updates, updates from the RIB or Operation API requests will need to be queued and will not be serviced until the BGP component is woken up again.

Second Generation Message Passing

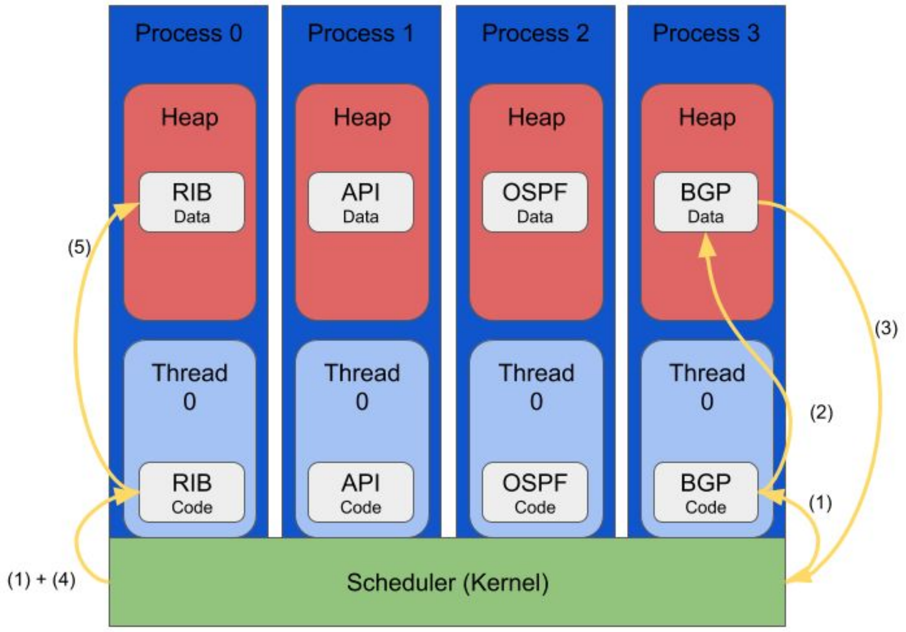

Figure 5: Second Generation Message Passing Sequence

(1) The scheduler wakes up the BGP component and the RIB component.

(2) The BGP component learns routes from its neighbor and decides 10,000 route updates need to be sent to the RIB component.

(3) The BGP component makes a system call into the kernel and copies the 10,000 route updates ‘into’ the kernel.

(4) The kernel informs the RIB component that it has data waiting to read.

(5) The RIB component copies the 10,000 route updates ‘out’ of the kernel. The RIB component installs these routes into the RIB table.

Points of interest:

- At (3) the BGP component interrupts the kernel and requires attention. The kernel may reply with a busy indication and the BGP component will have to hold the routes and try again 'later'.
- At (3) The message requires copying a fair amount of data. If we assume the route is IPv6 then as a minimum we have a network (16 byte address + 1 byte prefix) leading to an output interface id (4 bytes) + next hop (16 bytes) = 37 bytes. In practise route updates tend to travel with a fair chunk of metadata so are much larger than this. This makes the route update message a minimum size of 370KB.
- At (5) the data is copied again.

Third Generation Message Passing

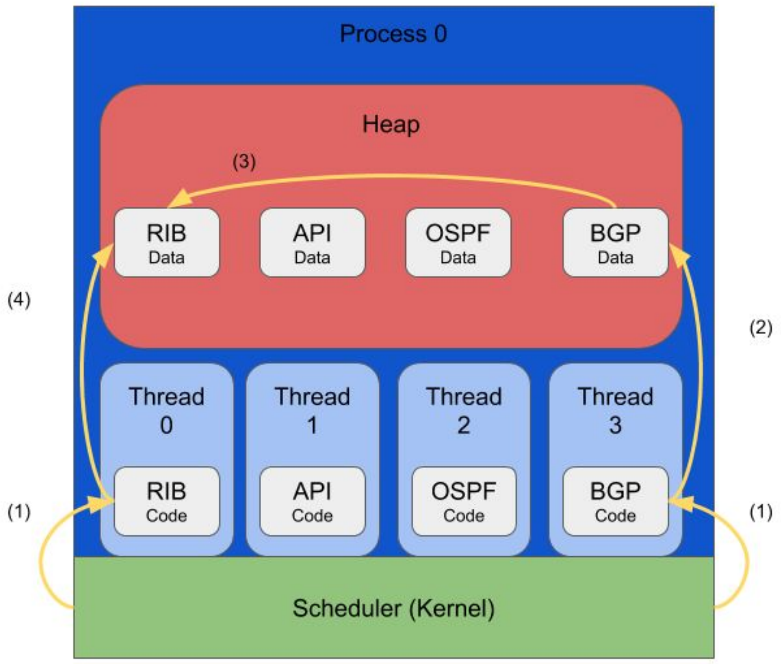

Figure 6: Third Generation Message Passing Sequence

(1) The scheduler wakes up the BGP component and the RIB component.

(2) The BGP component learns routes from its neighbor and decides 10,000 route updates need to be sent to the RIB component.

(3) The BGP component sends a message to the RIB component over a lock free channel. The message consists of a reference to the object holding the 10,000 routes. The reference consists of the address of the object and its size. Once the message is sent the BGP component can no longer access it or the associated route update data. This is a Rust compile time guarantee.

(4) The RIB component receives the message over the lock free channel. The RIB component uses the reference to the route update object and installs these routes into the RIB table.

Points of interest:

- The size of the message passed at (3) is tiny. On a 64 bit CPU it will consist of an 8 byte pointer + 8 bytes of size information. The 10,000 route updates remain at the same memory address.

Summary

The first generation IP routing suite has stood the test of time and is still used widely in production environments today. Its main benefit is that it is so efficient. Even though it is monolithic and can only run one component at a time, as CPU clock speeds exponentially increased from 1980 to around 2005 it gained a significant performance boost with each CPU upgrade. Now with CPU clock speeds levelling off and being replaced by multi-core, this performance increase has stopped.

The second generation IP routing suite is the current de facto standard. It is capable of utilizing the highest end hardware. However with its multiple processes and heavy weight messaging it is very inefficient.

The third generation IP routing suite with its compile time memory protection and lightweight message passing is by far the most efficient. As more CPU cores become available, increased performance will be easy to achieve by splitting each component up into multiple threads. In addition, it consists of a single process which makes it easier to deploy and operate. The low latency and low jitter message passing reduces the risk of component states becoming unsynchronized.

The Flock Networks implementation of the third generation IP routing suite also has these characteristics to enhance its performance further:

- Route updates are handled in batches, which minimises the number of times routes need to be computed and keeps the CPU instruction cache hot.
- All data structures are CPU data cache friendly. Modern CPUs can execute hundreds of instructions in the time it takes to fetch a single cache line from main memory[11]. If an application is not implemented in a cache friendly way, the CPU will repeatedly stall, waiting for the requested instructions or data.
- It is lock free. This means each thread of execution is never blocked by any other thread. When running on a multi-core processor, each component achieves independent parallel execution.

[11] https://en.wikipedia.org/wiki/CPU_cache

Author

Nicholas Carter

Founder

Flock Networks Ltd

ncarter@flocknetworks.com

flocknetworks.com