Linux Fast Path Using Application-Specific Fast Path
# TABLE OF CONTENTS

Abstract .......................................................................................................................... 3
Abbreviations .................................................................................................................. 4
Market Trends/Challenges .............................................................................................. 5
Solution ............................................................................................................................ 7
Best Practices .................................................................................................................. 9
Common Issues ............................................................................................................... 10
Conclusion ..................................................................................................................... 11
References ...................................................................................................................... 12
Author Info ..................................................................................................................... 12
Abstract

Linux Fast Path (LFP) is a term used to describe a paradigm where an alternate data path is created within the Linux networking stack that enables faster processing of data packets. This alternate data path is specific to only particular application packet types which are deemed important by the network service provider. Hence, what it really implies is that performance efficiency improvements can be gained for certain applications by being able to treat these particular application packets in a different way. This is termed as Application Specific Fast Path. The Linux kernel achieves this improvement as it runs a special piece of code to attain this Fast Path with direct access to the hardware so it can bypass layers of generic software present in the stack. Also, it’s able to leverage the hardware more efficiently and typically in a customized fashion for certain applications. Although there are pros and cons involved with every technique developed for attaining performance improvements, the cost benefit analysis works in favor of the Linux-based ASFP.
### Abbreviations

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Acronyms</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LFP</td>
<td>Linux Fast Path</td>
</tr>
<tr>
<td>2</td>
<td>ASFP</td>
<td>Application-Specific Fast Path</td>
</tr>
<tr>
<td>3</td>
<td>Mpps</td>
<td>Mega packets/second</td>
</tr>
<tr>
<td>4</td>
<td>FP</td>
<td>Fast Path</td>
</tr>
<tr>
<td>5</td>
<td>OEM</td>
<td>Original Equipment Manufacturers</td>
</tr>
<tr>
<td>6</td>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>7</td>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>8</td>
<td>Berkeley System Distribution</td>
<td>BSD</td>
</tr>
</tbody>
</table>
Market Trends/Challenges

Linux is a generic operating system which caters to a wide range of user applications. Historically, it was used in a traditional server storage kind of enterprise application environment, having picked up its legacy from its elder brethren such as Unix System V, Berkeley System Distribution (BSD) Unix, etc. But that was a long time ago. Today’s Linux is a much more improved, faster and a more stronger operating system that is penetrating into almost everywhere that an operating system could actually fit, whether be it embedded applications used in various verticals such mobile phones, automobiles and medical devices or be it a niche carrier grade telecom system which has a much more stringent requirement in terms of performance, stability and features. In today’s fast-changing Internet age, speed is of prime importance, irrespective of the context of the application. Unfortunately, due to its monolithic design in terms of the network stack and the kernel, Linux doesn’t score very high when it comes to speed, especially if we refer it in the context of high bandwidth carrier grade data networking systems.

The main bottleneck to increasing packet processing capability lies in what is called as the Linux Networking Stack. This stack is a collection of layers which have been traditionally been growing in complexity as newer technologies and more features are being added by network service providers.

Let's start with a small illustration as to how much capability a networking stack has to process packets being given to it by the network interface driver.

For a 64 byte packet, we can calculate the total number of packets that will be processed by a 100BaseT Fast Ethernet interface.

We know a FE interface can work at 100Mbps, i.e.

\[ 100 \times 1,000 \times 1,000 = 100,000,000 \text{bps} \]

Also, if we consider a standard 64 byte packet, we can say the interface is processing 100,000,000/64 packets/second. But that's not all really. There is an 8 byte preamble and 12 byte inter frame gap as well that needs to go into the calculation. Hence, we have

\[ 100,000,000/64 \times 8 \times 12 = 0.1488 \text{Mpps} \]

Similarly, for a 1G interface, the number of packet processed per second value is 1.488 Mpps

The above values actually show what kind of packet speed the interface driver will be handling. But that's just one side of the story. The problem in the forwarding engine of the networking stack being slow is not so much a network interface driver related issue. The main bottleneck is the stack which uses precious CPU time to run the engine. For example, in the case of a 1GHz clock cycle based CPU, the number of
clock cycles taken by the CPU to process a 100Mbps bit stream from the Ethernet driver can be calculated in the following manner:

\[
1\text{GHz Frequency} = \frac{1,000,000,000 \text{ clock cycles/second}}{0.1488 \times 1000 \times 1000} \text{ OR } 6720 \text{ cycles/packet}
\]

For a 1G interface link, this value will reduce to just 672 cycles/packet. This just indicates that the CPU will need to process the packet 10x faster than it was doing for the FE link.

The Linux network stack is actually a whole lot of software layers catering to a wide variety of functionalities. It has been estimated by a lot of earlier work that the Linux network stack uses at least 2,000 CPU cycles per packet of processing. If we use this metric against the above calculated cycles/second metrics for a FE/gigabit link, it can easily be seen how a 1GHz CPU can hardly match to a 1G wire speed transmission rate, let alone a 10G transmit rate. In the case of an FE link, the packet received needs to be processed within 6,720 cycles. Out of this, if the stack uses 2,000 odd cycles, the packet can still have a cushion to be processed easily. But if we take 1G link packet processing, everything goes haywire as the 672 ≪ 2,000 and possibly the stack won’t be able to play catch-up with the driver speed.

There are ways to get around this to some extent by increasing the CPU frequency and increasing the number of cores. But we all know the limitations of increasing these factors and the performance improvement isn’t linear. Moreover, higher layers of the operating system hardly have any control over the hardware features such as the processor and accelerators. On top of this, add the context switching and locks in the multitude of layers in the stack cause more latency.
Solution

Each approach has some pros and cons, but the basic idea remains the same. Do the packet processing in a separate Fast Path and the normal path handle the control set-up.

From this paper’s scope, the discussion is limited to designing a Fast Path in software based on the Linux kernel. Moreover, the Fast Path we will develop will be an application-specific fast path. The aim of the Linux Application Specific Fast Path (LASFP) module is not to replace the complete Linux networking stack (possibly inhumane thinking!!) but to accelerate the packet processing of the most commonly used functionalities (read protocols and features).

The ASFP module will actually be able to fast track the processing of certain protocol packets which will be user controlled. The user will be able to decide which of the traffic being received by the device needs to be fast tracked and which can sustain certain delays.

The Linux ASFP module consists of four main components. These are as follows:

**ASFP Engine**

The engine is the brain of the module where all decision making happens. Packets received on the network interface are passed on to the ASFP engine rather than the normal path for receive processing. The ASFP engine then checks the packet against a packet flow which is retrieved from the ASFP Rules/Action DB. This packet flow is based on the L2/L3/L4 header information. If the packet received from the network matches one of the flows stored in the database, the packet is then sent to the ASFP Consume/Forward block for further processing, which is decided based on the action attached with the rule for that flow.

**ASFP Control**

During standard Linux processing, information is learned in the networking stack (ARP entries, L3 routes, IPsec security associations, etc.). This information is automatically and transparently synchronized to the ASFP Rules/Action database so the next packet from the same flow can be processed by the fast path. This synchronization is achieved by the ASFP Control Block. The continuous synchronization mechanism allows the system to process any kind of packet (even those that are not supported or not yet configured in the fast path) and transparently update information in the fast path.
ASFP Rules DB

The ASFP Rules database is a table containing the packet control information which is used to decide whether the packet will be processed in the ASFP module context or if it needs to be given to the Linux Networking stack for its regular processing. This database is continuously synced with the different packet control information being updated in the networking stack dynamically or through a control/management plane configuration.

ASFP API Subsytem

The API subsystem is set of APIs which can be used to sync/update the Rules/Action database with the networking stack. The purpose of the ASFP subsystem is to provide a generic interface to the ASFP packet engine.

Hence, the basic functionality of the ASFP module is to configure a set of rules and associated actions. The incoming packets are then checked against these rules to determine a match. If it does match, it is processed in the ASFP context. Otherwise, it’s given back to the networking stack. Eventually the ASFP Rules/Action database will be updated with the similar information present in the networking stack.
Best Practices

Thus, all this reasoning leads us to one basic inference, that is, inducing a separate path within the kernel to fast process data traffic in some manner such as to achieve transmission rates close to wire speed by consuming less CPU cycles. Fast Path is a method which isn't very new, as network equipment OEMs have been using it for a long time. Different vendors use different methods for achieving Fast Path. To name just a few of the popular ways to achieve this:

- ASIC based FP
- Network Processor based FP
- Control Plane and Data Plane segregation on multiple cores
Common Issues

The design of the Linux networking stack in monolithic to say the least. And ideally to solve the problem in hand, the best option is to rewrite it! Having said that, I think that’s an impossibility, although not from an implementation point of view, but keeping in mind that there are millions of routing devices using that software who would need to upgrade, not considering the attached defects that might arise due to the change in software.

So the Fast Path is definitely the way forward, but there is a cost to that along with the limitations that come along. Some of the most commonly faced problems with implementing Fast Path using the ASFP methodology are:

1. The FP will work only for a limited set of network applications. If it were to support all the features, it would be similar to rewriting the stack altogether, in addition to requiring a complete redesign of the hardware that supports the stack.

2. Being close linked to the hardware in order to take advantages of the proximity, the FP feature will be very hardware-centric. This means that one particular implementation of the FP might not be a simple port to another device which uses a different hardware profile.

Typical Use Cases

**IP Forwarding**

IP forwarding is the most basic operation which every routing device is capable of. This feature allows a packet to be sent to the correct next hop that could either be another router or be the actual destination, based on a set of IP addresses contained in the packet received.

In the normal networking stack processing, the IP forwarding feature would need to consult the routing table to decide on the fate of the received packet. If this is the first packet toward that destination, the packet is passed to the networking stack due to the lack of ASFP Rules database containing a packet flow rule/action for that destination. Upon this ASFP Rules DB Miss, the ASFP Control Block updates the Rules database with this packet flow control information and hence, the next packet coming in will get processed in the ASFP module.

**NAT/Firewall**

NAT flows can be identified by the 5-tuple information. These flows could also be a potential use case for offloading it to the ASF module.
Conclusion

The Linux ASFP runs close to the hardware and has access to all the hardware features. Since it has a small code and memory footprint, it will be implemented in the L1 I-cache and L1 D-cache, thereby providing fast processing. It can be scaled to many more use cases besides the few shown above. Linux ASFP will use the Run-To-Completion model that reduces the context switching and results in better cache usage. It can also be customized for specific hardware, as the ASFP module sits very close the Ethernet driver and has complete control over it. When combined with other desirable features like Large Segment Offload, TCP Segment Offload, Large Receive Offload and optimizations such as Zero Copy Termination, and Read Copy Update Spinlocks to avoid packet processing locking issues, Linux becomes a feature rich and a very fast kernel capable of providing fast packet processing times with less CPU cycles, thereby delivering carrier grade switching times and high throughput.
References


Author Info

Gaurav Bansal works with the HCL ERS Practice team as a Senior Architect. He has approximately 18 years of software experience in the designing of carrier-grade telecommunication software with expertise in Layer 2, IP and embedded technologies. Currently he is involved in building HCL solutions in the Telecom and Video domains.
Hello, I'm from HCL’s Engineering and R&D Services. We enable technology led organizations to go to market with innovative products and solutions. We partner with our customers in building world class products and creating associated solution delivery ecosystems to help bring market leadership. We develop engineering products, solutions and platforms across Aerospace and Defense, Automotive, Consumer Electronics, Software, Online, Industrial Manufacturing, Medical Devices, Networking & Telecom, Office Automation, Semiconductor and Servers & Storage for our customers.

For more details contact ers.info@hcl.in

Follow us on twitter:  http://twitter.com/hclers


Visit our website:  http://www.hcltech.com/engineering-services/

About HCL

About HCL Technologies
HCL Technologies is a leading global IT services company, working with clients in the areas that impact and redefine the core of their businesses. Since its inception into the global landscape after its IPO in 1999, HCL focuses on ‘transformational outsourcing’, underlined by innovation and value creation, and offers integrated portfolio of services including software-led IT solutions, remote infrastructure management, engineering and R&D services and BPO. HCL leverages its extensive global offshore infrastructure and network of offices in 26 countries to provide holistic, multi-service delivery in key industry verticals including Financial Services, Manufacturing, Consumer Services, Public Services and Healthcare. HCL takes pride in its philosophy of 'Employees First, Customers Second' which empowers our 85,194 transformers to create a real value for the customers. HCL Technologies, along with its subsidiaries, has reported consolidated revenues of US$ 4.4 billion (Rs. 23499 crores), as on TTM ended Dec 31 ‘12.

For more information, please visit www.hcltech.com

About HCL Enterprise
HCL is a $6.2 billion leading global technology and IT enterprise comprising two companies listed in India - HCL Technologies and HCL Infosystems. Founded in 1976, HCL is one of India’s original IT garage start-ups. A pioneer of modern computing, HCL is a global transformational enterprise today. Its range of offerings includes product engineering, custom & package applications, BPO, IT infrastructure services, IT hardware, systems integration, and distribution of information and communications technology (ICT) products across a wide range of focused industry verticals. The HCL team consists of over 90,000 professionals of diverse nationalities, who operate from 31 countries including over 500 points of presence in India.

For more information, please visit www.hcl.com