System Issues for High End Security Appliances Design

Huailin Chen
www.valleytalk.org
huailin@gmail.com

Jin Shang
www.hillstonenet.com
jshang@hillstonenet.com

Dongyi Jiang
www.hillstonenet.com
dyjiang@hillstonenet.com

Abstract

The design and implementation of high-end security appliances is a major challenge requiring high throughput, excellent manageability, reliability, and the ability to debug the system.

In this paper, we outline the challenges we faced in the past decade when doing system design. We will share our solutions, the tradeoffs we made, and concluded with a successful design that was conceived mostly from a bottoms up evolution instead of a top down design. We argue that most of the problems we experienced are common to the whole telecom sector. We believe that system researchers should pay more attention to the problems that we experienced.

Keywords Operating System, Security, Appliance, Scheduling, Memory Management, Availability, Multi-Core, Integration

1. System Definition

A system is usually defined as a set of interacting or interdependent components forming an integrated whole [1].

An operating system (OS) is software that manages computer hardware and software resources and provides common services for computer programs. The operating system is an essential component of the system software in a computer system. Application programs usually require an operating system to function [2].

In industry, especially the communication sector, the semantics of system or operating systems are different than in an academic setting. A commercial system is specifically designed for use in switches, routers, firewalls and other devices. Academics tend to focus on kernel level functionality. Linux, FreeBSD, or VxWorks are examples of kernel level software. Most companies start with kernel level services and then modify and build upon these kernels to create their operating systems. Ciscos IoS, Junipers JUNOS, NetScreens ScreenOS, and Hillstones StoneOS are examples of operating systems built on kernels [3,4,5,6].

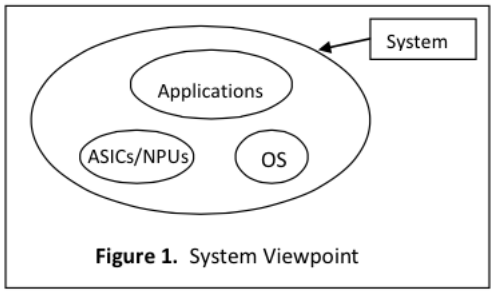

Figure 1. System Viewpoint

A stateful security appliance (firewall) typifies the challenges many industries face because it requires high high throughput performance, but also excellent management capability, high availability and debug-ability.

We have been working on the design and development of security appliances for the past 15 years. The appliances range from 10Gbps to 350Gbps. In this paper, we discuss some of the system issues we experienced and briefly share the solutions and tradeoffs we have made. One of the reasons for submitting this paper is to encourage the research community to pay more attention to the problems experienced by commercial enterprises.

This paper is organized as follows. In section 2, we provide a simple classification of security firewalls and explain why stateful systems have been a challenge from a system design viewpoint. In section 3, we will examine system issues and discuss the challenges that academic researchers might want to tackle.

2. Stateful Security Appliances

A security appliance is composed of a control plane, a data plane, a management plane, and a debug plane. Most people combine the control, management and debug planes together, and (for simplicity) only classify a system with control and data planes.

We further divide systems into high-end or low end with the

CONF 'yy, Month d–d, 20yy, City, ST, Country.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

following criteria: If the data plane is supported by dedicated line-cards, we call it a high-end appliance. It is a low-end appliance if the data plane and control plane share a CPU or a multicore/NPU chipsets.

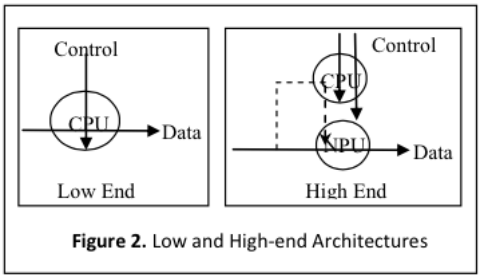

Figure 2. Low and High-end Architectures

In the above diagram, the dotted line represents the first packet of a stateful session. For most firewall/security products the first packet of a connection needs to be forwarded to the CPU so that a session can be established and installed to the ASICs and/or NPUs. This establishes that subsequent traffic belongs to this session and can pass- through the appliance without the CPUs involvement.

3. System Issues and Tradeoffs

3.1 Control Plane vs. Data Plane

For appliances, what does the system mean?

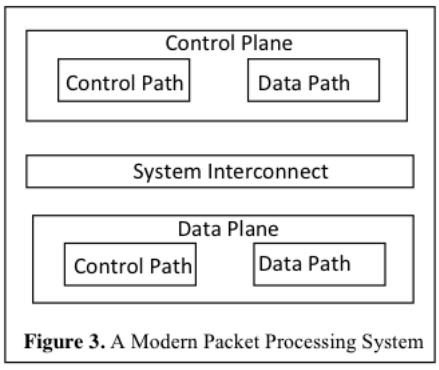

Figure 3. A Modern Packet Processing System

A telecom system is logically divided into two subsystems:Control Plane (CP) and Data Plan (DP). The control plane is usually composed of a set of processes plus either a proprietary OS or an open source OS, for instance, VxWorks, Linux, Freebsd, QNX, to name a few. The data plane usually consists of line cards or service cards. Each line or service card is designed with a control CPU and a few NPU or ASICs that are dedicated for packet processing.
Control Planes and Data Plans have both control paths and data paths. [7] Control path usually means the logics that are responsible for configuration, setting up initial status and event reporting. For illustration purpose, lets consider high-end appliances. The data plane refers to line cards or service cards.
The line card has two components: (1) the control cpu which is responsible for configuration and is the control path of the data plane. (2) NPU or ASICs that perform packet processing and is the data path of the data plane.
Similarly the control planes control path is responsible for configuration and database while the data path has dynamic routing processes PKI services and other work-flow that are tightly related to data planes.
Therefore, we define a system as follows:

System::=[Control Plane] [Data Plane]

Control Plane(CP)::=[CP Control Path] [CP Data Path]

Data Plane(DP)::=[DP Control Path] [DP Data Path]

Over the past years system architects have moved features from the data path in the control plane to the control path in the data plane. A typical example is Public Key Infrastructure (PKI). PKI is computing intensive and is the control path of IPSEC VPN. The IPSEC data path resides in the data path of the data plane. Consequently, if PKI can be moved to the control path of the data plane, the system should have better performance.
One of the reasons that this architecture makes sense is that the CPU capacity in the data plane has become very powerful and their performance in the line/service cards would essentially be wasted after system booted up.
Acceleration techniques are also being employed on the data plane where cpu intensive data path software is being moved into hardware.
In short, the industry is optimizing design topology to avoid communications overhead and congestion that system interconnects may introduce.
Data Path has two parts: Slow Path and Fast Path.

Data Path = (Slow Path, Fast Path)

Slow Path generally implies that the architecture cannot be accelerated. Fast path implies that the design is capable of improved performance during the next rev of the product.
We have found that this consensus understanding of system design has brought helpful insights when we refresh a product line, or when we convert a high end product to a middle or low-end product.

3.2 Process Based Protection

Process based programming provides many benefits and is the primary reason that telecom companies have been migrating from VxWorks to either Linux or Freebsd.
For most of our products we use multi-thread/single process or a hybrid multi thread/multi process model. Few companies in our industry use a single thread/multi process model.

Memory protection among user level processes plays a critical role in our designs. However, shared memory based multi-threading can be a problem. A bad pointer access from one thread can easily destroy the whole processs data, bss,and heap as well as other threads private data.
Even though POSIX and other OSs provide some stack protection different threads in the same process still share a common protection. For example, with TLS(Task Local Storage), a thread can have its own tbss and tdata sections. However, these sections have no protection at all when a malicious pointer access happens. [9, 10]

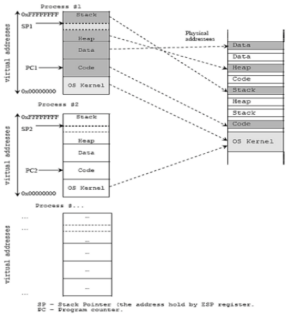

Figure 4. Process Based Protection

Memory corruption is responsible for most system instabilities issues and is one of the toughest issues to identify and fix.
In our proprietary OS we use a thread level local stack and a heap mechanism to provide granular protection. First, we split the heap space of a process into a Local heap and Global heap. All threads share Global heaps while local heaps (fixed size) are bound to a single thread. All local heap and stacks are guarded by MMU based protection.

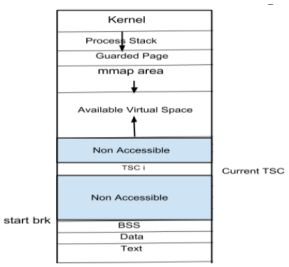

Figure 5. Thread Level Protection

Our proprietary heap protection affords system stability and reliability. Our experience has shown that most system crashes are due to bad pointer usage.

3.3 Scheduling Mechanism

Traditional OS scheduling mechanisms do not meet our requirements. The reason comes from the basic understanding of a system: What object is to be scheduled?
For traditional OSes, the scheduling algorithms are mainly concerned about CPU time. It is a Time Oriented Scheduler[7,8,9].
For firewalls, as well as other communication appliances, the basic object to be scheduled is the data packet. It is of throughput-oriented scheduling mechanism.
Figure 6 is an illustration that shows the scheduler from our viewpoint.
We think that the events in a system are only composed of traffic and keyboard being hit. Otherwise, the whole system is totally idle. System can contribute all computing capabilities to process through traffic if no CLI task was activated for running state.
When working under multicore chips for product design, most of our scheduling policy was using CPU affinity to lock software threads onto pre-allocated hardware threads. And a proprietary packet scheduler took over the whole system, having an out of order packet engine that accepts, process packets. And one or more dedicated threads will be responsible for the packet retirement.

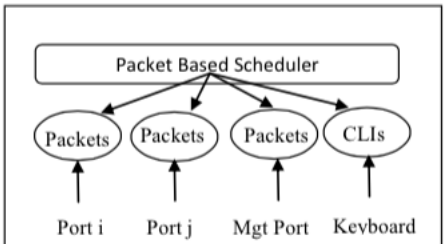

Figure 6. Packet Based Scheduling

3.4 Virtual Memory and Heap

For every system we had shipped so far, we did enable the MMU so that we can have advanced MMU based protection. But we were not fully using some features of modern OS VM subsystem.
Page on Demand and Copy on Write are two principles of VM subsystem [9,14]. However, they are not favored for telecom appliancs.
For a high throughput system, we cant afford to any non deterministic delay that page fault exception may introduce. The non-deterministic is not only about the MMU TLB, Cache parts, but also, the delay from kernel queuing.
For our system design, we have been trying to avoid these Page Fault issues at all. Our solutions are that, when system booted up, we will pre-allocate all DRAM and had the corresponding page table entries all ready. For example, which part is packet memory, which part is local heap, which is global heap were pre-allocated. All MMU entries or hardware page tables were setup already before the system jumped to process the first packet.
Regarding heap part, what we learned was that every task should maintain its own heap space. Different tasks could have different heap allocation algorithms. For telecom appliances, memory allocation/free performance were very critical. We think that glibc malloc with ptmalloc2 could not

best provide our needs. For glibc, for small memory blocks allocation, all threads will all go to the main arenas. Also, even though different arenas are protected by locks, threads are still sharing the heap space interleaving.

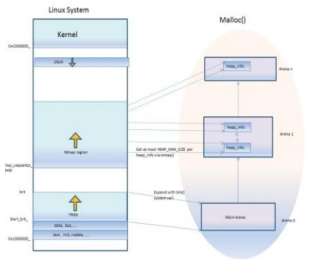

Figure 7. glibc arena management

We learned this when tuning BGP and OSFP performance. These routing tasks need lots of small memory blocks being temporally used. Very frequent malloc and free with small chunks, for example, 64Bytes,256 and 512 Bytes. If these actions all happened in a global heap, BGP performance got downgraded a lot.

The solution of ours was that we first gave BGP a big chunk of memory space. With that space, BGP task built its own heap and allocation mechanism. We had some memory cache mechanism being designed and implemented, so that BGP task will not put small memory blocks back to the heap and got merged. Instead, we will keep bunches of small blocks of memory so that we can feed BGP task very quickly. And we did find this memory usage improvement helped us gain BGP performance a lot.

3.5 Cache Miss and Thrashing

For firewall system, an OS contains several millions of lines code; it is a big runtime package.

We frequently struggled with performance challenges when products were to be released. A particular product line had to meet marketing performance metrics before went to market. During the past decade, a big lesson that we had was that 10-20 percentage performance gain could be easily brought back in the tuning stage, if kernel engineers applied cache analysis and corresponding adjustments. Typical methods were to arrange the code functions and data structures in the elf sections. Sometimes, we found a buggy line of code could even result in horrible cache misses.

Our findings were mainly as follows: When an appliance started to process packet, either through or self traffics, the system work-flow was generally composed of some fixed amount of functions. And the calling sequences mostly followed a pattern, for instance, some main data path functions for through traffic; some main control path functions for self traffic. Similar to data side, the data structures being involved were some global packet buffer, management data array and so on. We called these Systems Working Set.

For a system with our proprietary kernel, we tried to use gcc section extension to group together the working set into the same elf section. And then at linkage, we put those working sets (TEXT and DATA) onto some pre-allocated virtual memory ranges. We did find we have a good performance gain with usually 5-10 percentage; sometimes, up to 20 percentage.

Also, we carefully calculate the page coloring pattern based on different CPUs, for example, PowerPC 7447s. And we tried to move around those TEXTs or DATAs that we found could have cache thrashing. We found it was very hard to have a more than 30 percentage performance gain if only with cache usage fine tuning. Usually, if a system was having a very bad performance number, for instance, 50 percentage lower than expectation, the root cause was usually from architecture design.

For Linux or other commercial OSs, we found that user level applications could not directly access privileged cache instructions was really not convenient for us when doing high-end system design[17].

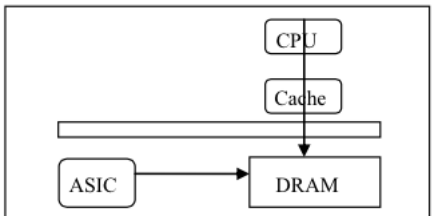

Figure 8. Cache Position under System View

For a high-end firewall system, usually, CPU side and ASIC or NPU side need share some blocks of DRAM, for example, session memory, some mgt control buffer. Some memory are non-cachable; some are cacheable with either write-back or write-through.

ASIC usually used DMA engine or some dedicated channel to read/write data to DRAM while CPU fetching a data need pass through cache and memory bus.

Lots of modern CPUs need an application to have privilege mode in order to do cache update, or cache invalidate operations. While we can understand this is for protection purposes, this limitation did introduce lots of inconveniences for the interactions of ASIC side and CPU side.

User level cpu cache control is surely a very friendly mechanism that we think research area should consider[18]. For our product design, we had been using our internal package for this purpose, and extremely accelerated our product implementation.

For advanced cache usage, including Write-Back, Write-Through and Cache Prefetch, Cache lock, we extensively tried use various combinations in our product design. And our experiences are: We did not find significant performance impact between write-back and write-through memory regions in our high-end firewall products. For cache lock(both instruction and data), we realized this could not contribute

any significant performance gain for the products we involved before. For cache prefetch, it was a very interesting scenario. We tried at least 5 products ranging from middle-end to high-end products, and found it was very difficult to know where to put the right prefetch instructions in order to have the performance gain. We had one very successful try for a middle rang product that was using previous IXP 23xx chipset. Our conclusion was that cache pre-fetch could be a big deal if the underlying memory bus was very slow, or the data was moving from a place that need bridge over. That was the case that we reaped the gain. For rest of products that did not fit this category, we donnot think cache prefetch matters.

3.6 Lock Usage

Spinlock and mutex are two locks that mostly got used for many appliances. During our product design, we felt any of commercial OS alone could not completely fit our need. We once tried to combine both Linux and FreeBSD lock features and provide the APIs to network engineers[19,20].

Some people had confusion about hardware lock engine and memory based locks. For example, might be hardware locks are better than memory based? Should we provide as much as possible hard locks when doing a ASIC or NPU design?

We ever put lots of testing over this part. For example, tested both hard locks provided by some chipsets and memory based spinlock. We did not think there was much difference, if not the same. And we eventually adopted memory based spinlocks for system usage. One of big reasons was that we could then provide lots of locks, and maintained a clean and consistency code base and APIs. Introducing and maintaining two different sets of lock mechanisms to engineers were indeed a pain for future.

Meanwhile, we believe that the critical part for a lock usage is the lock granularity, not the lock implementation itself overhead. And thus avoiding a big lock is always welcome. Lots of engineers did not pay enough attention on it, always having an assumption in mind that the overall system performance would not affect simply because their part of codes.

Many engineers did not understand when they should use spinlock, when then should use mutex or rwlock and so on. And we did also see lots of bugs related to this part. One case was that one line buggy code led to a 2million USD deal screwed up, and more than twenty people spent whole weekend to find the root cause. And It was a Thanksgiving holiday.

We indeed support nested locks. And we did not suffer some bugs that people forgot to unlock the locks that they owned. We implemented several features to detect the unpaired locks so that the runtime software could issue a warning or even crashed the box when in QA testing build.

3.7 Queues and Various Congestions

For modern operating system, we realized the stack was an outstanding limit. And thats why lots of industry companies have been using customized stack, or even wrote their own to bypass the default stack.

With our experiences on this part, an important system issue was that default stack was lack of the support for congestion control.

For a firewall that supports routing mode, the cpu side usually need support BGP, OSPF and some dynamic routing processes. Those routing instances are highly replying on heartbeat messages to maintain neighbors. For example, BGP will treat its neighbors being lost if not able to receive several heartbeat messages.

We suffered several urgent customer issues from some big customers including a well-known service provider.

When a big amount of traffic was trying to pass the appliance, we noticed that BGP task started to complain the lost of heartbeat packets. However, when we remotely accessed to our customer network and had some debugging, we did find the heartbeat messages were already being sent to control CPU side.

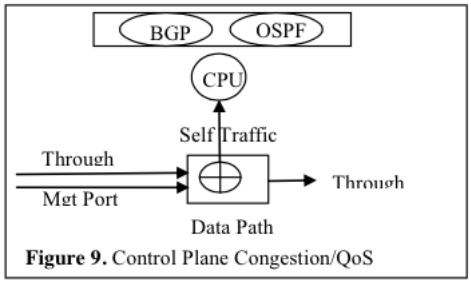

Figure 9. Control Plane Congestion/QoS

After dumped corresponding packet buffers and some painful investigations, the BGP keep-alive messages were simply congested within the queue to the CPU side. A typical HOL(Head of Line) issue happened in the control path. We realized that we should have always given these BGP keep-alive messages highest priority with special queues, so that the control path will always consume the special queue first in order not to miss critical messages. A big lesson we learned was: Multiple Queues are always better than single queue[21].

After we redesigned the queue structures, we always had a special queue from data path to control path, in which BGP and other dynamic routing keep-alive packets would be put. And this solved our customer issues very well.

The root cause of dynamic routing case also applied for the stability of our HA subsystem. As we know, for an Active/Passive or Active/Active HA environment, it would be a disaster if people found HA Master/Slave thrashing.

For most of our high-end appliances, HA was one of important features that customer wanted, especially for financial and service provider customers.

We did experience several master/slave thrashing issues. And the root causes were very similar. For HA system, mast and slaves need to report each other its status periodically. Whenever slaves thought the master was died, a new election among slaves would happen. Therefore, when the critical keep-alive messages were delayed unexpectedly, a chaos happened. Master and Slaves were keeping up and down.
From our fixes to all these above congestion issues, we felt that HOL had contributed lots of critical system issues during the past decade. Its strongly encouraged that system architect should pay more attention on this part, particularly, for the control traffic side, while data plane usually has some high-arch queue engines available for QoS guarantee.

3.8 Manageability

No matter how good the performance number of an appliance was, the device became useless if it could not be managed by people. The manageability issues were not trivial at all.
We summarized three outstanding manageability issues that we think it is worthwhile to be explained in this section.
Usually for a telecom appliance, there is a task called CLI, Command Task, or Console, which is used for people to use a command line/keyboard to setup the config, display statistics, push config, or any other management jobs. At most time, this task is IDLE until people hit the keyboard to input an CLI.
CPU needed to split its time to the cmd task. What could happen if the appliance was extremely busy processing packets at that time?
During our design, we had a long struggle with this issue. We found it was very hard to make a good tradeoff so that CPU can process console work and meantime the performance of packet processing would not drop too much, especially, when doing the firewall session ramp-up.
We did not know this issue until we had reports from field engineers. Our customer complained that the console could not response at all when heavy traffic was passing through. Ironically enough, a character just being hit even would not show up at all after 10 seconds. The root cause was straightforward: Console task could not get enough CPU time while timer-interrupt based packet processing had been using too much CPU circles.
To alleviate this issue, we first tried to cut the amount of packets to be processed every round. And it was apparently, it was very difficult to choose a right number. One of big reasons was that we could not afford to lose performance too much. Keeping a good ramp up rate and concurrent session numbers was very important for enterprise and service provider network.
Give that the granularity of cutting total amounts of packets to be processed was too coarse, we tried to cut traffic based on port, along with based on traffic type.
Issues got alleviated, but not fixed. We had been suffering this pain and customer issues for years, and finally, we designed a set of CLIs so that our customers could adjust the number and the ways of cutting traffic by themselves.
The situation got better or fixed until we started to work on our first multi-CPU and then later on multi-core based systems. We think having separated computing resources was the reason that this issue was gone.

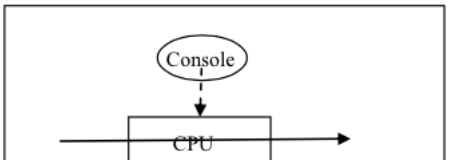

Figure 10. Died System when under flooding

We experienced similar situations when trying to push routing configuration and security policies to data plane from control plane. Once a time, we found that our system spent more than 10 minutes to get it done for installing half million routes. This situation could not get accepted by our customers.
Overall we think the lack of Control Plane QoS was the root cause for various congestion and manage-ability issues. Most of work has been done for data plane performance. However, we argue that control plane performance is critical as well [22].

3.9 Micro-Kernel and Performance Issue

Micro-kernel vesus monolithic kernel has been a debate for long time. A typical arguing point was that micro-kernel may hurt application performance too much. The good parts are that micro-kernel does provide better architecture and protection model by using message passing.
We had one chassis based high-end products with several IDS service cards. For the IDS module, we ported our codes on top of the QNX neutrino micro-kernel[24].
Our experiences were that the porting efforts were pretty straight forward because the legacy codes were mostly glibc style. We were able to get porting done within one-three months and had packets passed through. It would be very difficult for us to re-write our mostly share memory based legacy codes to the message passing based one. We picked the hybrid model and used mmap to keep legacy semantics. Our conclusion was that system architect should make a good tradeoff when handling micro-kernel architecture, when trying to handle millions of lines legacy codes. Our best practice was: Take advantage of micro-kernels advantages but avoid those native messaging services.
For performance concern, we spent lots of efforts to do the tuning. And we did not see big performance drop that we

were scary about at the beginning. 5 percentage around performance drop was totally acceptable in the industry when doing a product refreshing.
Overall, we did not recommend our community use micro-kernel technology if you had been using monolithic kernel for product line and had accumulated many legacy codes. We think that monolithic OS has been evolving and converging with micr-kernel or virtualization stuff these years. Thus the gaps between monolithic and micro-kernel are not significant any more.
As a system architect, you need to plan your produce line futures evolvement. Since most of chipset vendors always have limited resources to provide software SDK support for their silicons, they usually only focused on Linux part. Given this kind of situation, the selection of OS should be considered not only systematically as well as realistically.

3.10 Multicore Usage Pattern

We have been using commercial multicores for our product design and implementation and also involved in a multicore chip design for years.[25,26]
Below are some our experiences or lessons.
For the product that we designed before, we adopted a hybrid system model, instead of a pure pipeline, or a parallel model. And for risk control and simplicity reasons, we statically divided all available cores, hardware threads for dedicated usage. While we did know this might not be the best choice, we felt this was the most realistic approach.

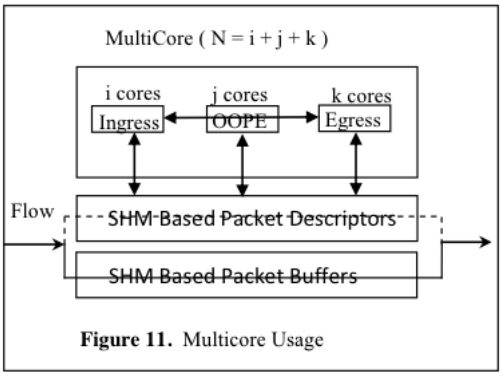

Figure 11. Multicore Usage

For the design and implementation that we had done, we explicitly split the N cores with i, j and k cores. The first i cores were solely used for ingress packet processing; the middle j cores were responsible for security/routing handling. Usually, the j cores were out of order engines, dedicated to pick up available packets ; The last k cores were used for doing the final checks and some sequences arrangements before sending packets out to egress ports.
We found our design pattern was able to satisfy most of our needs while keeping a system simple enough. Surely, The limitation of our design was the static partition of i, j and k cores. We did find this limit hurts our next generation system when doing the upgrades.
The best core allocation schema would be that the system can dynamically adjust the size of i, j and k, based on some dynamic performance factors .
We had been carefully watching the cache and memory bus issue. We did have a chance to collect some statistic data and realized that some HWT were competing the memory bus extensively when trying to obtain a shared spinlock (memory based). And the distribution of having the lock among the related HWTs was not strictly even indeed. After we compared the data and our conclusion was that we did not get a hit a scared starving issue yet and thus did not investigate this issue further.
We also participated in a multicore NPU design for several years. The chip was to be used internally for high-end firewall products.
We found that the co-design philosophy was very important for a silicon to be a successful tapout. Otherwise, huge of efforts including money could be easily wasted. Silicon engineers usually were lack of knowledge why some logics of a chip should be added, and how some parts were a must-have feature, while some were not. Ironically, software engineers would always like to ask chip designers one question: Could we have a big L2/L3 cache?
Usually, for a NPU silicon, the die is composed of two big parts: Core side and Engine side. And a system interconnect is used for connecting everything together. From our experiences previously, the system interconnect and cache coherence protocol stability were the two most critical parts.

4. Discussion

We have been working on real-time telecom appliance design for the past 20 years. We feel that Integration Complexity has been becoming one of the most difficult system design issues we face.
Any standalone sub-system of an core router, edge router, carrier switch, or a high-end firewall is not that difficult to design. But the systems became complex when we started to glue these pieces together. We encourage system researchers to consider the qualitative and quantitative aspect of systems design as part of their research.
We think it is very important for Service Chain integration when all major service providers move towards NFV (Network Function Virtualization). We expect the problems from multi-vendors integration will become very complex for system designers in the next decades.

Acknowledgments

We would like to thank Dr. Xianan Tang, Dr. Nick Zhang and Steve Karkula for their helpful reviews on drafts of this paper.

References

[1] http://en.wikipedia.org/wiki/System

[2] http://en.wikipedia.org/wiki/Operating_system

[3] http://en.wikipedia.org/wiki/Cisco_IOS

[4] http://en.wikipedia.org/wiki/Junos

[5] http://juniper.net/techpubs/software/screenos

[6] http://www.hillstonenet.com/

[7] http://en.wikipedia.org/wiki/Routing_control_plane

[8] http://tools.ietf.org/html/rfc4945

[9] Maurice J. Bach, The Design of the UNIX Operating System, Prentice-Hall, 1986

[10] http://en.wikipedia.org/wiki/Thread-local_storage

[11] Volker Seeker, Process Scheduling in Linux, University of Edinburgh, May 12, 2013.

[12] M. Tim Jones, Inside the Linux scheduler, IBM developer Works Technical Library, June 30, 2006

[13] Daniel P. Bovet & Marco Cesati, Understanding the Linux Kernel, October 2000

[14] http://en.wikipedia.org/wiki/Demand_paging

[15] http://en.wikipedia.org/wiki/Copy-on-write

[16] Justin Ferguson, Understanding the heap by breaking it - Black Hat

[17] Harvey G. Cragon, Computer Architecture and Implementation, Cambridge University Press, 2000

[18] Xiaoning Ding et al., ULCC: A User-Level Facility for Optimizing Shared Cache Performance on Multicores

[19] Silas Boyd-Wickizer, et al., An Analysis of Linux Scalability to Many Cores

[20] Marshall Kirk McKusick , An Overview of Locking in the FreeBSD Kernel

[21] http://en.wikipedia.org/wiki/Head-of-line_blocking

[22] Intel Corp, Intel Data Plane Development Kit (Intel DPDK)

[23] http://www.cs.vu.nl/ ast/reliable-os/

[24] QNX, QNX Neutrino System Architecture

[25] http://en.wikipedia.org/wiki/Multi-core_processor

[26] http://www.cavium.com/OCTEON_MIPS64.html